Scenario Usage in System Development: A Report on Current Practice*†§

Klaus Weidenhaupt, Klaus Pohl, Matthias Jarke, Peter Haumer[1]

RWTH Aachen, Informatik V

52056 Aachen, Germany

{weidenh|pohl|jarke|haumer}@informatik.rwth-aachen.de

Abstract: *Scenario-based approaches are becoming ubiquitous in systems analysis and design but their definition and scope remain vague. Complementing the recently proposed CREWS classification framework for scenario-based RE, this paper reports on an exploratory survey of practice conducted through site visits with 15 projects in four European countries. The main findings include that: (1) the variety of purposes and uses of scenarios in the process is much greater than expected; (2) as a consequence, we must take scenarios much more seriously as important design artifacts, offering better means for structuring, management, and evolution. To handle these complex processes, users request more explicit methodological guidance and more adequate tool support.*

1. Introduction

Scenario-based approaches are attracting more and more interest in requirements engineering research and practice.

The research literature offers an increasing number of scenario-related methods, models and notations. The consideration of concrete system descriptions from a usage-oriented perspective – prior to abstract conceptual modelling of function, data, and behavior – has been highlighted in software engineering in the form of use cases within object oriented analysis and design [1]. A number of extensions and alternatives have been proposed, which, e.g., focus on adding structure to use cases [2;3], on the formal treatment of scenarios [4], on the use of scenarios during documentation, discussion and evolution of requirements [5] etc. In addition, scenarios are also popular in other fields, most notably human-computer interaction (e.g. [6]) and strategic planning (e.g. [7;8]).

---

* This work was in part founded by the European Community under ESPRIT Reactive Long Term Research 21.903 CREWS.

† A revised version of this paper appears in: IEEE Software, March, 1998.

§ An extended abstract of this paper appears in: Procs. Intl. Conf. on Requirements Engineering (ICRE 98), Colorado Springs, Colorado, April 1998.

1 The following members of the European ESPRIT project CREWS and the Scenario Working Group of the German Informatics Society have contributed to the empirical studies underlying this paper and related discussions: M. Arnold (FIDES Informatik Zürich, Switzerland), C. Ben Achour, C. Cauvet (Université Paris-Sorbonne, France), E. Dubois (FUNDP Namur, Belgium), M. Erdmann (University of Karlsruhe, Germany), M. Glinz (University of Zürich, Switzerland), P. Heymans (FUNDP Namur, Belgium), R. Knoll (RWG GmbH, Stuttgart, Germany), N.A.M. Maiden, S. Minocha (City University London, UK), B. Paech (Technical University of Munich, Germany), J. Ralyté, C. Rolland (Université Paris-Sorbonne, France), J. Ryser (University of Zürich, Switzerland), R. Studer (University of Karlsruhe, Germany), A. Sutcliffe (City University London, UK)

Scenario use is also becoming a pervasive phenomenon in industrial practice, but comprehensive and expressive studies on the practical relevance of the techniques proposed by research are still rare. Recent surveys are mostly broader in scope. Lubars et al. [9] and El Emam/Madhavji [10] report on the state of practice in requirements engineering in general but deal with the aspect of scenario usage only in passing. Other studies draw their conclusions from observations in a single project. For instance, Gough et al. [11] examine scenario usage at Philips medical systems development. Catledge and Potts [12] report on experiences in the industrial Centauri project, but observed that – in this project – scenario usage played only a surprisingly minor role.

The European ESPRIT project CREWS (Cooperative Requirements Engineering With Scenarios) aims at a deeper understanding of the diversity of scenarios, in order to help improve methodological and tool support for scenario-based requirements engineering. A two-pronged strategy is being followed to gain this understanding.

Firstly, following the example of the "three dimensions" framework of RE developed in the pre-cursor NATURE project [13], a scenario classification framework was developed based on a comprehensive survey of scenario literature in requirements engineering, human computer interaction, and other fields [14]. The framework was applied to classify eleven prominent approaches.

To complement this research framework, the state of applying scenarios in industrial projects was investigated through site visits with scenario user projects. Given the lack of theory in the field, this study had to be exploratory in nature, and was mainly aimed at understanding the diversity of issues prevalent in scenario-based approaches.

This paper focuses on the site visits made during this effort. Our findings indicate that, while Jacobson's use case approach has instigated many companies to promote scenario-based techniques, the actual usage of these techniques goes much beyond what is described in textbooks and standard methodologies. As a consequence, users face significant problems in scenario management not yet addressed adequately in neither theory nor practice. There is a large demand to solve these problems.

The rest of the paper is organized as follows. Section 2 provides an overview of the study design, of the projects covered in the site visits, and of their main scenario characteristics. Sections 3 and 4 summarize our main findings concerning the purpose and contents of scenarios as well as their structuring and management. Section 5 relates these findings to the literature and draws some conclusions.

2. Overview of Site Visits

Fifteen projects in four European countries were selected for site visits. Each site visit involved two to three people from the CREWS project and one or two members from the examined project, mostly project leaders or consultants. The duration varied from half a day up to one day. Most site visits were recruited directly or indirectly through representatives from software companies and independent consultants serving on the Industrial Steering Comitee (ISC) of the CREWS project.

The site visits were first documented in minutes and then summarized in a technical report [15] which highlights for each site visit the project background, the scenario characteristics, the way in which scenarios were produced and used, the benefits and problems/needs stated by the interview partners and our main lessons learned from the site visit.

2.1. Preparation

The CREWS classification framework [14] was used to derive a catalogue of questions for scenario characterization in the site visits. This framework originally was gained from a comprehensive literature survey and applied to classify eleven scenario approaches. It views scenarios from four different angles, concerning the form, purpose, content, and life-cycle of the scenarios used in the project (cf. figure 1).

The questions related to the *form* view deal with the expression mode of a scenario. Typical questions emerging from this view are: is a scenario formally or informally described, in a static, animated, or interactive form?

The *contents* view concerns the kind of knowledge expressed in a scenario. For instance, the scope of scenarios can vary from system-internal to organizational context, a scenario can cover normal cases or exceptional ones.

The *purpose* view is used to capture the role a scenario is aiming to play in the software development process, e.g. describing the system functionality, exploring design alternatives, or explaining drawbacks or inefficiencies of a system.

The *life-cycle* view considers scenarios as artifacts existing and evolving in time. Aspects of technical handling, evolution and of project management are of interest to this view.

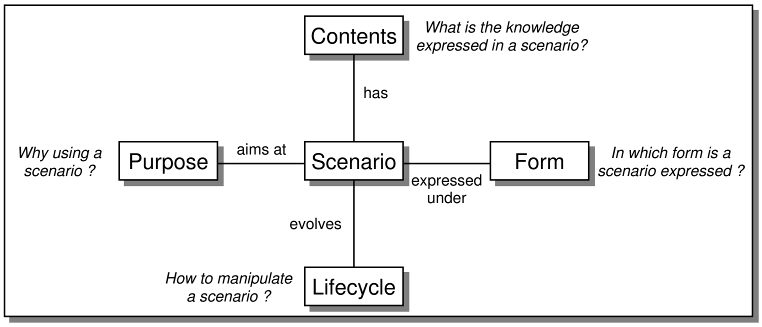

Figure 1 Four Views on Scenarios.

It was interesting to notice that although the general distinction between these four views proved to be useful to structure the observations from real world, the focus on the individual facets shifted during our investigation. For example, in practice form issues play a much more minor role than in research literature while the usage and lifecycle aspects are much richer than anticipated from the literature survey.

Besides the scenario characteristics, the interview plan also addressed two other topics:

- *project profile*: to better understand scenario use in a broader context and to identify potential correlations between project and scenario characteristics;
- *experiences*: to elicit the main benefits gained through the use of scenarios, and to capture the known open problems and future needs.

To avoid the danger of biasing the interviews to scenarios aspects captured by our interview plan, we let the interview partners talk freely about their overall development process, the usage of scenarios, and their experiences made, but used the interview plan as a checklist to ask for missing information about the various scenario usages and experiences identified.

In addition, we asked our interview partners to provide us with concrete project material used to explain their scenario generation and usage.

2.2. Project Characterization

Table 1 characterizes the projects evaluated during the site visits in terms of their the application domain, size and if the project was performed in-house, by a software contractor or if we got our insights from a consultant of the project. The project size was classified into the categories *small* (less than 10 person years), *medium* (between 10 and 50 person years), and *large* (greater than 50 person years).

As the table indicates, the visited projects vary widely in terms of application domains and project size. This can be seen as an indicator that the usage of scenarios is not restricted to a specific application domain or for a specific project size.

	application domain	size and duration	in-house/consultant/ software contractor
P1	medical information system	small	software contractor
P2	network documentation and management	small	software contractor
P3	business information system (banking)	large	in-house
P4	business information system (public authorities)	large	consultant
P5	business information system (insurance)	small	consultant
P6	satellite communication	medium	consultant
P7	medical systems	medium	in-house
P8	air traffic control systems	large	software contractor
P9	systems engineering for warships	large	in-house
P10	radio telecommunication	large	consultant
P11	management of train networks	medium	consultant
P12	c/s applications for government and banks	medium	software contractor
P13	water invoicing management system	large	software contractor
P14	CASE tools for bank applications	small	software contractor
P15	process engineering	medium	software contractor

Table 1 Project Characteristics.

2.3. Scenario Characterization

Table 2 depicts a characterization of the scenarios used in the various projects according to the four views (form, content, usage/purpose, management/life-cycle of scenarios) of the CREWS classification framework. To characterize the relevance of table entries, we use three attribute values: “X” means significant occurrence, “O” moderate occurrence, and "–" no occurrence of that aspect in the project. In the following, we summarize some main observations on results listed in the table.

project/ scenario facet		P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15
form	narrative text	X	O	X	O	O	O	O	X	O	—	—	—	X	—	O
	structured text	X	X	O	X	X	X	X	X	X	O	—	X	X	—	X
	diagrammatic notations	—	—	O	X	—	O	X	X	—	X	X	X	X	—	X
	images	X	X	O	O	O	O	—	O	O	—	—	—	O	—	—
	animations / simulations	—	—	—	—	—	—	—	—	X	—	X	—	—	X	—
	typical size (measured in pages)	1-3	1-2	2-8	3-8	10-20	3-20	5-20	/	/	1-2	3-10	1	10-200	*	10-50
content	system context	O	O	O	O	—	—	—	X	X	—	—	—	X	—	X
	system interaction	X	X	X	X	X	X	X	X	X	—	O	X	X	X	X
	system internal	—	—	O	—	—	—	—	O	X	X	X	—	X	—	—
purpose and usage	concretization of abstract models	X	O	X	X	X	X	X	X	X	—	—	X	O	O	X
	scenarios instead of abstract models	—	—	—	X	X	—	—	—	—	—	—	—	—	—	—
	scenario use with prototypes	X	X	X	X	O	O	—	—	O	—	—	X	X	X	—
	complexity reduction	X	X	X	X	X	X	O	X	O	O	X	X	X	X	X
	agreement and consistency	X	X	X	X	X	X	X	X	X	X	O	X	X	X	O
	scenario use with glossaries	—	—	X	O	O	—	O	—	—	—	—	—	—	—	—
	reflection on static models	X	X	X	X	X	X	X	X	—	—	—	X	X	X	X
lifecycle and manage-ment	partial views	O	O	X	X	X	X	O	X	X	X	X	O	X	—	O
	distributed scenario development	O	—	X	X	X	O	O	X	X	O	—	—	X	X	O
	review	X	X	X	X	X	X	X	O	O	O	O	O	X	—	X
	traceability issues	X	O	X	X	X	O	O	X	X	O	O	O	X	O	O
	basis for test cases	O	—	O	O	O	O	O	—	O	O	O	O	O	O	O
	evolution	X	O	X	X	X	X	X	O	X	O	X	X	X	O	X

Table 2 Scenario Characteristics.

* 80 % of the screens of the end-product

Three categories of scenario *content* were found. *System context* refers to descriptions of the broader environment in which the system is embedded, *system interaction* covers how the system interacts with its environment, and *system internal* refers to internal interactions between components of systems.

The *form* of scenarios seems to be correlated with the content. Five basic representation types dominated in the projects (see table 2). Twelve projects made heavy use of natural language, either as *narrative text* without any structure or more preferably as *structured text* following a more or less rigid template or table-structure. In eight of these projects structured text predominated, three of them made equal use of narrative and structured text, and only one project preferred free-form prose. In any case, natural language was mostly used for context and interaction scenarios: twelve of the thirteen projects focussing on interaction and context scenarios relied on textual notations. To a less extent, this also holds for *images* (e.g. screendumps of user interface forms) which are used for illustration mostly in context and interaction scenarios. In contrast, three out of four projects dealing with internal scenarios employed more formal *diagrammatic notations*, such as object interaction or message trace diagrams, and *animations*.

Beyond their expected purpose as means for requirements elicitation and validation (not mentioned in the table), lots of other *usage* aspects play a major role. For example, *concretizing* requirements by lowering the abstraction level was seen as an important purpose of scenarios in thirteen projects; two of them even changed their process to a scenario-based approach after encountering severe problems with the traditional development of abstract models. The role of scenario as means for reaching partial agreement and consistency of the requirements specification was stressed in nearly all projects. Another prevalent use of scenarios was complexity reduction of the requirements engineering task and enforcing an interdisciplinary development process. Somewhat surprisingly, twelve projects emphasized the reflection of dynamic scenarios on static models, i.e. the population and validation of object models. Some not widespread, but interesting interactions of scenarios were observed with prototypes and glossaries. More details about these usages are described in section 3.

The *management* issues that played a major role in most projects were: how to impose *partial views* on a scenario, how to handle *distributed scenario development*, how to enable quality assurance through *scenario reviews*, how to make scenarios *traceable* along the whole process, how to re-use scenarios as *test cases*, and how to *evolve* scenarios. These are elaborated in more detail in section 4.

Summarizing, the diversity of scenario usage observed in the projects was much greater than one would expect from the UML-incited understanding of scenarios as instances of use cases. More than two thirds of the projects (P1 – P8, P12 — P14) claimed that they followed a use case approach, but all of them extended the textbook version significantly to make it work.

3. Scenario Usage in Practice

Table 2 provides only a glimpse at the diversity of scenario purposes and uses found in practice. On the one hand, scenarios serve as an important enabler of interdisciplinary learning in RE. On the other hand, scenarios lead to a different division of labor with significant consequences for project management and artifact integration.

3.1. Use Scenarios When Abstract Modelling Fails

The use of scenarios to concretize abstract models is prevalent to a more or less extent in thirteen projects. In two of them, P4 and P5, which at first neglected the systematic consideration of concrete system usages, the development of abstract conceptual models, e.g. class models, failed due to the complexity of the problem domain. As an alternative approach both projects then applied a scenario-based approach to elicit and document the customer requirements. This approach turned out to be successful in both cases.

In project P4, after half a year, it became evident that object model complexity and communication overhead within the developing team could not be managed any more. The business processes to be supported were not yet sufficiently understood. The customer did not understand the abstract models to such an extent that validating them became almost impossible.

The definition of the class model was therefore stopped and a scenario-based approach was established. They first used scenarios to divide the application domain into 15 topics, such as personal data capture, fines, etc. Each topic was then assigned to a developing team consisting of 5–7 stakeholders. To understand each topic, they again used scenarios. This led to a better integration of the domain experts into the analysis process since they found it much more convenient to talk about concrete scenarios instead of abstract models. In addition, the focus of project management shifted. Instead of ensuring consistency across the whole project, now more emphasis was put on achieving good partial understanding in each of the individual topics.

Project P5 initially used a Petrinet-based business process modelling tool. After defining more than 120 business processes with Petri-nets, interrelating an additional model consistently with the existing models, and assuring consistency became almost impossible. Therefore, the project's management decided to stop the formalization of the business processes. Instead they employed a more informal scenario-driven modelling approach. Interestingly, all potential usages of the system could be captured using 27 scenarios. The main reason for this complexity reduction was that the business processes defined as Petrinets were much more fine-grained than the processes described in the scenarios.

3.2. Scenarios Require the Coexistence of Prototypes

In two third of the projects, scenario generation and usage were heavily interrelated with rapid prototyping or even building first versions of the new system. In particular, in the projects P1 – P4 and P12 – P14, our interview partners stated that the combination of both approaches yielded symbiotic effects, i.e. without using prototyping, the value gained from using scenarios would drop almost to zero and vice versa. In project P1, the coexistence of a prototype and typical usage scenarios was considered as vital for selling the overall project to the customer, i.e. the usage scenarios together with the prototype convinced the customer that the new system is going to meet his needs.

Typically, the integration of scenarios and prototyping was achieved as follows. In early project stages, a first set of scenarios was developed to communicate application knowledge as well as the system vision from the domain experts (e.g., customer, user) to the system engineers (e.g., requirements engineer, system architects, designer). Based on the set of scenarios, the system engineer developed a requirements specification, e.g. class model and behavior model. This specification was then used to develop prototypical implementations. Those prototypes vary from simple paper-based user interface forms for validating the class model to comprehensive implementation for validating real time aspects of the system to be built.

The initial scenarios were then reused to validate the prototypes and also, indirectly, the requirements specification. This evaluation of the prototypes led to the detection of misunderstandings between the domain experts and the system developer, e.g. the fact that the system developer has made the wrong abstractions based on the initial set of scenarios. Such misunderstandings can be resolved more easily since the scenarios build a common basis for communication.

Equally important, the validation of the prototypes against the initial set of scenarios was seen as ideal means for the domain experts to validate the initial set of scenarios themselves. Missing functionality, over-specifications, errors, and even unintended side effects were detected. For example, in project P2, the designer mistakenly concluded from an unprecise statement in a scenario, that there exists an 1:1 association between two domain classes. Although, the class model was reviewed by the domain expert, this error was not recognized until the user had played with a user interface prototype. Here, the 1:1 association was implemented as two single-selection list boxes for establishing references between objects of the two classes. When testing the prototype with the scenario, the domain expert detected the missing possibility to interrelate three or more objects.

Based on the detected gaps, the scenarios were improved or the prototype and/or the specification were adapted to the new detected requirements. Thus, an evolutionary system development process in which the domain experts work closely with the software developers had been established.

To make this process feasible, consistency must be somehow ensured between the three components. This was seen as major problem for which no systematic approach exists.

3.3. Scenarios as Means for Complexity Reduction

Our findings support the claim of the use case approach that it enforces a usage-oriented decomposition of the requirements analysis problem from the very beginning. The consideration of only one single business process or task at a time reduces the number of system aspects the stakeholder involved in the process must cope with simultaneously.

Scenarios are seen as a structuring device not only for requirements engineering but also for the whole system development process. For example, the projects P3 – P6, and P10 used scenarios to divide work between designers, programmers and even system testers (see also Section 4.1).

To integrate the different use cases, a sequential strategy was followed in several projects. For example, in project P5 the first version of the system was restricted to a single scenario. First, they defined at a coarse grained level the set of scenarios, i.e. business processes, the system should support. By a management decision, the most important scenario was selected. The selected scenario was then analyzed, a specification and a first version of the system was developed, tested and installed. Then the next scenario was chosen and the same procedure was repeated until the system offered sufficient functionality. While this development approach of coarse caused some rework, the knowledge that further scenarios must be integrated led to a reuse- and maintenance-oriented system design and implementation.

A similar strategy was pursued in project P6, where the classification of scenarios into primary and secondary ones was used to reduce the complexity of the system characteristics to deal with at a given time. Moreover, it helped to define delivery stages.

Project P12 experimented with complexity measurements attached to the scenarios. These formulas were used to guess the time it will take to process them through the development cycle, e.g. by assessing the number of involved objects, presence of subsystem interaction, the number of action etc. This information was used in management decisions, e.g. to select the scenarios to be considered first, or the ones which should be decomposed.

In the literature, the role of scenarios in exception identification and exception handling is often emphasized. Interestingly, this role could be observed only in the projects P8, P9 and P11, mostly due to their safety-critical nature. In other projects, the need for exceptional scenarios was explicitly denied. One argument was that discussing exceptions would unnecessarily complicate discussions and distract domain experts from the main system goals. One interview partner, emphasizing the marketing aspects of the requirements process, even had reservations that a too detailed discussion of exception scenarios might endanger the saleability of the system[3].

[3] We think, that this standpoint is fraught with danger. Neglegting important exceptions in the requirements engineering phase likely results in customer dissatisfaction and, thus, will cause higher costs in the long term.

3.4. Scenarios as Means for Reaching Partial Agreement and Consistency

The stakeholders affected by the system development have different goals and aims about the future system. Even their perceptions of current reality vary significantly. Bringing all stakeholders together and reaching an overall agreement on the system is too time consuming or even impossible. Similarly, assuring that the system to be built is consistent with all aspects of all existing systems in an organization is often infeasible.

Nearly all projects used scenarios to drive the agreement process and to establish partial consistency between existing systems. It turned out that reaching *partial* agreement and consistency is sufficient in practice, especially in the large and complex projects (P4, P5, P8, P9, P10, and P13).

In contrast to overall complexity reduction, where scenarios are used to restrict system functionality, here scenarios were used to reduce the scope of discussions and agreement processes. For example, scenarios were used to achieve agreement about the performance of a particular business process and the support for this process provided by the system. Scenarios served also as means for discussing alternative solutions, grounding discussions and negotiations on real examples, and for supporting trade-offs among design alternatives.

Dealing with a concrete business process scenario, the stakeholders were able to detect situations where the use of individual, conflicting taxonomies suggested the existence of a conflict, even though the stakeholders in principle agreed. Conversely, scenarios also enabled the detection of different perceptions. In one business process, for example, one stakeholder assumed that the evaluation of the customer data he produces is needed by another stakeholder, whereas that stakeholder was wondering why she gets this information.

3.5. Bi-directional Relationship between Scenarios and Glossaries

In project P3 (and to a less extent in P4, P5, and P7) the intertwining of scenarios with a project-wide glossary was used as a means to establish a common understanding of the terms used between different stakeholder groups such as developers, domain experts, and managers. When developing a new scenario, the developer had to consider and use the key terms already defined in the glossary and establish a reference to the corresponding glossary item. For terms not yet included in the glossary, the developer had to introduce a new glossary item with a short, general definition.

The interesting observation, however, was that the relationship between scenarios and the glossary was a *bi-directional* one. The glossary items were related to (parts of) one or more scenarios in which the item defined plays an important role. Technically, this was realized by establishing a hypertext infrastructure in a project-wide intranet in which corresponding scenario parts and glossary items were linked to each other.

As a result the terms were not only defined in the glossary by an abstract definition, but also related to a broader usage context. The relations established between the terms and the scenarios could be used to explain the definitions by a set of concrete usages represented in the scenarios. It was especially those relations which helped both the developers and domain expert to adjust their interpretation of the key terms used and thereby reach a common, project wide understanding. Moreover, these relations provide excellent means for making new project members familiar with the terminology used in the project.

In addition, the relationships established between scenarios and glossary served as access path for the scenarios themselves. For example, a stakeholder interested in the use of a certain artifact in all business processes, e.g. the "creditworthiness file" for a certain customer, could use the relations between the glossary and the scenarios to access all relevant scenarios, e.g. the scenarios "check creditworthiness", "decide on granting a loan" etc.

3.6. Reflection on Static Models

Although scenarios are originally intended to bring dynamic aspects into the requirements specification, 12 projects used scenarios to define and validate *static* (object) models. Among others, scenarios were used to check the completeness of object models, to populate object models by deriving new objects and/or by identifying constraints such as cardinalities of associations or plausibility conditions on attribute values.

We identified two different ways of using scenarios to define and validate structural models. The projects P1, P3 – P6, P12, P14, P15 employed a sequential development chain starting with informal scenarios which were gradually transformed into conceptual structural models. For example, in project P1 domain objects and relations were identified from the textual representation of the scenarios which were then transformed into some kind of pre-structure, such as class-responsiblity-collaboration cards (CRC cards), still understandable by customers or users. The structural descriptions were then used to define a more formal object model.

In the projects P2, P7, P8, and P13, scenarios and structural models were developed in parallel and independently from each other. Thereby two descriptions of the future system were established. Those descriptions were then used to identify inconsistencies by cross-checking. The detected conflicts led then to an improvement of both the object models and the scenarios.

4. Structuring and Managing Scenarios as Artifacts

In all projects, the creation, documentation and evaluation of scenarios was seen as a major effort itself. Even the development of normal, non-exceptional scenarios required a significant effort for several reasons. For example, domain experts normally do not reflect their daily work. If more than one expert is involved, their statements often differ; statements even change from one day to the other as the experts become more explicit about their current way of working and their system visions, etc.

All projects therefore saw scenarios not as the simple things they are often assumed to be but as complex artifacts that evolve over time and that therefore must be managed. In addition, the need for structuring scenarios according to certain facets was recognized.

4.1. Partial Views on Scenarios

Some scenarios affect many stakeholders or are so complex that a single stakeholder is only interested in part of them. For example, in project P13 we found scenario descriptions to be up to two hundred pages long! The need for different views on a single scenarios was also recognized in various other projects (P3–P6, P8, P10, P11, P13). We identified the need for three different kinds of such views:

- *manager/developer views:* Whereas it is sufficient for the manager to understand a scenario on a coarse grained (abstract) level, the developers and domain experts require a more detailed scenario description. To enable such views on a single scenario some projects used two kinds of scenario models: a graphical scenario model providing enough information for managers, and a detailed scenario description used by the domain experts and developers. These models were often developed in parallel. Keeping them consistent was seen as a major problem;
- *partitioning of scenarios for work distribution:* Views on scenarios were also established as a basis for distributing work within and among development teams. This can best be illustrated using message trace diagrams (MTDs). For example, in the radio telecommunication project P10, the objects expressed in the MTD could be divided into parts, defining different subsystems such as base stations and several types of mobile stations. The messages exchanged between objects belonging to different parts describe the interfaces between those subsystems. Such divisions were used in project management to assign responsibility for the subsystems (parts of the MTD). As a consequence, interactions between objects belonging to a single subsystem are of interest only to the group of developers made responsible for implementing, validating, and testing this subsystem.
  The definition of such views is fairly easy in the case of MTDs. It is much more difficult if scenarios are represented e.g. as prose or structured text. Despite the difficulties caused by not having a systematic or formal approach of defining such views (partitions) on scenario represented using structured or unstructured text, such views have been established in many projects;
- *dividing scenarios based on the underlying business process:* Especially in the case of large business processes, certain stakeholder are only interested in certain tasks or even activities performed in those processes. Mostly, such views were informally established. Only in project P9 in which warfare scenarios were simulated, such views have been formally defined to enable the display of relevant information to the right stakeholders during scenario animation.

4.2. Managing Distributed Scenario Development

In large projects and in the case of a complex problem domain, the scenarios were developed by several teams in parallel (projects P4, P5, P8, P9, P10, P13). Managing the distributed scenario development and keeping the resulting scenarios consistent was seen as a major problem. An interesting strategy for managing the distributed development of scenarios was found in project P4 in which hundreds of scenarios were developed individually across spatially distributed teams.

In this project, the problem domain was first divided into fifteen topics (business process parts). Each topic was then assigned to so-called *topic team* responsible for defining the scenarios concerning the assigned topic. In addition, a *scenario management team* was made responsible for managing the developed scenarios.

More precisely, a topic team is only allowed to reuse (part of) a scenario developed by another topic team if the scenario is a *public scenario*, or if the team has created a so-called *raw scenario*. A *raw scenario* is created when a topic team detects action sequences in their scenarios which may either constitute a scenario of general interest or which belongs to another topic. After a rough specification of the raw scenario, the topic team passes the raw scenario to the scenario management team. The management team identifies the topic team responsible for elaborating the scenario.

When a raw scenario is fully defined, it is passed to the management team who publishes it in the central library as a *public scenario*. For each public scenario references to the team responsible for the scenario and the teams using the scenario are maintained by the management team. Based on this information, bi-lateral or multi-lateral meetings between the topic teams help adjust inter-topic dependencies of the scenarios, and assure consistent propagation of changes.

4.3. Reviewing Scenarios

Similar to other software artifacts, scenarios are important project results that have to be reviewed to establish high quality. Thus, the projects P1 – P7, P13, and P15 established some kind of walkthrough and/or inspection process.

The review was sometimes conducted by the people involved in scenario creation, sometimes by domain experts not involved in the scenario creation process. In the projects P4 and P7 even experts from other domains were involved, such as quality assurance people, managers, or system maintenance people. Unfortunately, we were not able to obtain quantitative measurements about the use and impact of reviews, since such information was not captured during the projects.

4.4. Scenario Evolution

In all projects the definition of a scenario was not a one-shot activity. Instead, the scenarios evolved over time. We found four types of evolution in the various projects:

- *top-down decomposition:* At the beginning of a project, scenarios are often defined on a very abstract level to get an overview of the system and its functionality. Later on, these scenarios are further elaborated. A typical example is the definition of scenarios according to the abstractions used for defining business processes. First, the business processes affected, supported or automated by the new system are modelled in a set of scenarios. Next, the scenarios are refined by tasks performed in each business process. In addition, the relations defined between the business processes represented in the first set of scenarios are refined to the task level. Finally, the scenarios are enriched by activities performed for each task;

- *from black-box to white-box scenarios:* Black-box scenarios were used in project P9 to represent the interaction of the system with its environment and the interactions between objects (business processes, stakeholders, systems, etc.) existing in the environment. Once these scenarios have been sufficiently understood, the development team extended them to white-box scenarios which also represent also interaction between system components and thus information about system-internal aspects. By integrating white-box and black-box scenarios, a set of complex scenarios is defined which describes the interaction of the environment with specific subsystems;
- *from informal to formal scenario definitions:* Quite often scenarios were first expressed using free-form prose. Once sufficiently understood, a template structure was imposed, to add more knowledge and detect first inconsistencies and gaps. As a third step, (parts of) the structured texts were transformed into a more formal representation, e.g. message trace diagrams. Again this transformation went along with adding more knowledge and solving the inconsistencies and the gaps detected. In the safety-critical project P11, the MTDs were even transformed into a formal protocol specification language.

  Keeping the different scenario representations consistent was seen as major problem, since each transformation causes changes to the content of the scenarios (due to the inconsistencies and gaps detected). Adding to the consistency problem was the fact that the different representation formats could only be understood by a subset of the stakeholders involved. For example, if a new interaction was added during the definition of a MTD or an existing interaction was changed, the changes must be back-propagated to the textual scenario definitions for validation by the customer or user. Especially in large projects with a large set of scenarios, such changes are difficult to manage;
- *incremental scenario development:* In most projects the first version of a scenario typically encapsulated knowledge at different levels of detail. When validating the scenarios, e.g through review techniques or checking the scenario against other domain models, the scenario was subsequently improved, i.e. more knowledge was added or parts were revised.

The problems our interview partners were faced with in all four types of scenario evolution can be summarized as:

- identifying the right level of granularity and the right level of abstraction during scenario development and usage;
- keeping the various versions of a scenario and the different representations used for representing the scenario consistent;
- supporting change management across the different types/versions of scenarios; especially if a scenario encapsulates knowledge of different level of abstractions.

4.5. Deriving Test Cases from Scenarios

The need to base system tests on the scenarios defined with the customer/user during requirements engineering and system design was mentioned in nearly all projects (except P2 and P8). This should support the system developer to prove to the customer that the implemented system meets the requirements.

However, the current practice rarely satisfies this demand. The main problem was that the scenarios developed during requirements engineering and system design were out of date at the time the system was going to be tested. Therefore most projects lacked a systematic approach for defining test cases based on scenarios. As mentioned earlier, the coverage requirements of test cases may also be in conflict with the goal of complexity reduction which implies a small number of scenarios.

4.6. Traceability

In all site visits, the need for better traceability support was mentioned. Traceability was seen as a prerequisite for establishing life-cycle wide use of the scenarios defined during requirements engineering. It was often the lack of traceability which caused scenarios to be out of date and thus inconsistent with, e.g., the current version of design prototypes. Since scenarios were out of date, they were not used, e.g., as a basis for test cases.

Traceability is seen as major source for enabling change integration, and thus for keeping scenarios up to date. At the very least, traceability should be established between different levels of scenario abstractions, views on scenarios, different versions of scenarios, scenarios and the prototypes, scenarios and the specification, and scenarios and test cases.

Establishing traceability requires a better understanding of the relations between the various artifacts produced during the software development and the scenarios. In addition, better tools are needed to manage the scenarios and their relations. A lack of appropriate tool support was observered and manual assurance of consistency between the scenarios, and between scenarios and other artifacts, was seen as impossible.

The problems with tool support are highlighted by our observation that hardly between any two projects, even in the same organization, there is only one tool in common: the word processer! This indicates that there are no generally accepted tools for these issues.

5. Implications

In the previous sections we have highlighted a number of success stories from practice as well as several pitfalls and shortcomings of current methods and techniques which place new challenges on research. The prevalent conclusion from the site visits is that scenarios are pervasive artifacts which should be used throughout system's lifecycle, which serve for manifold purposes, and must therefore be managed with more care than usually discussed in the literature. The implications of these observations concern both the practitioner and the researcher.

5.1. Recommendations for the Practitioner

Our investigation brought many excellent ideas for scenario use. These ideas may encourage practitoners to begin use of scenarios in their daily work as bridges:

- *to business processes:* Use scenarios to relate system functionality to business processes and vice versa! The concentration of one usage aspect of the system at a time and the expression of the relations between the system and business processes helps to manage the complexity of the application domain.
- *between customer/user and developer:* Use scenarios as communication medium between domain experts (customers and users) and the software/requirements engineers! Scenarios – in conjunction with prototypes and glossaries – serve for transferring knowledge between both groups of people and for explaining and illustrating the different terminologies used.
- *between architectural and implementation components:* Use scenarios not only during requirements engineering, but also in the design and implementation phase to describe, test and validate interfaces of various architectural and implementation components!
- *between developer:* Use scenarios to distribute work between software engineers! For example, scenarios help to divide the overall system into subsystems at the design phase for which a group of people can be made responsible for defining the detailed design. Scenarios can be used in the implementation and test phases in a similar way.
- *between software development phases:* Use scenarios, in addition to abstract models, for transferring knowledge across software development phases, e.g. requirements engineering and design, or requirements engineering and system testing! Scenarios capture valuable background and environment information which enables the stakeholders of later phases to understand the desired system better.
- *between structure and behavior:* Use scenarios to highlight the dynamics between the various static components of a system defined using, e.g., UML [16] or OMT [17]! More surprisingly, in well understood domains, scenarios pave the way to predict the influence of a system on its environment, i.e. to envision changes in

the environment caused by using the new system.

5.2. Challenges for Research

While a strict focus of scenario roles may clarify specific issues, it also tends to obscure the deep intertwining implied by these bridging functions, and thus the resulting management implications. These implications raise management problems not yet adequately addressed by research.

Extensions and interrelation with other techniques: About two thirds of the visited projects claimed to follow the OOSE [18] approach, but all of them broadened the concept significantly, as also suggested by some authors in the literature. For example, extensions involved system-internal and context-scenarios explicitly excluded by Jacobson (but predicted as necessary e.g. by [18;19]), generalizing the structuring mechanisms offered as also suggested by [3], and linking to design level prototypes, as reported for the Danish Great Belt project by [20]. The need to integrate scenario development with prototypes which can be used to validate the scenarios or to validate the prototypes based on scenarios was also stated by [4;21]. While these advocate a formal representation to generate scenario animations or prototypes we observed a much more informal use of scenarios in conjunction with prototypes, e.g. to validate user interface forms based on scenarios described in CRC cards. Summarizing, researchers are challenged to better understand the relationships of scenario methods to other techniques and extend the scope of scenarios.

Evolution and management: The management of scenarios observed in the various projects and the classification of research contributions [14] showed, that there are many similarities between managing the evolution and changes of software artifacts and scenarios. The main difference is the continuous involvement of the customer in scenario management which places additional demands on keeping user-understandable scenario representations always consistent with formally analyzable ones. This makes evolution management for scenario-based approaches even more difficult than for more formal parts of the software process.

Traceability: Traceability was mentioned in all site visits as a vital prerequisite for being able to manage the evolution of the scenarios and their relations to other software artifacts. To establish a comprehensive support for managing traceable scenario development and usage, a much better understanding of the relations between the scenarios and the other software artifacts must be established.

Process guidance: Most projects were faced with the problem of deciding when to develop which kind of scenario, at which level of abstraction, and when to stop, i.e. to decide when the development and the use of scenarios pays off. In all site visits this was seen as a major problem in applying scenarios in real projects. Thus the development of scenarios was mainly seen as a craft instead of an engineering task. Even for projects following the full use case approach proposed in the OOSE method textbook [1], our interview partners stated that OOSE leaves too much room for (mis)interpretation and does not provide sufficiently detailed guidelines. Therefore, the projects had to develop their own concretizations of the use case approach in terms of the content, purpose and position in the overall process, form and structuring, size and granularity, and the relation to other requirement artifacts. The richness of scenario usage and management problems seems to indicate that comprehensive process guidance is still far away.

Acknowledgments: This work was supported in part by ESPRIT Reactive Long Term Research Project 21.903 (CREWS) which involves RWTH Aachen, City University London, University of Namur (Belgium), and Universite Paris-Sorbonne. We are grateful to our project partners for their participation in the data collection, and to the members of the industrial committee, most notably its chairman Peter Hruschka, for their support in establishing contacts to scenario projects.

References

[1] I. Jacobson, M. Christerson, P. Jonsson, and G. Oevergaard, *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison Wesley, 1992.

[2] B. Regnell, K. Kimbler, and A. Wesslén, "Improving the Use Case Driven Approach to Requirements Engineering," in *RE '95: Proc. Intl. Symposium on Requirements Engineering*, 1995.

[3] B. Regnell, M. Andersson, and J. Bergstrand, "A Hierarchical Use Case Model with Graphical Representation," in *Proceedings of ECBS'96, IEEE Second International Symposium of Computer-Based Systems*, 1996.

[4] P. Hsia, J. Samuel, J. Gao, Y. Toyoshima, and C. Chen, "Formal Approach to Scenario Analysis," *IEEE Software*, pp. 33–41, Mar. 1994.

[5] C. Potts, K. Takahashi, and A. Anton, "Inquiry-based Requirements Analysis," *IEEE Software*, vol. 11, pp. 21–32, Mar. 1994.

[6] J. Carroll, "The Scenario Perspective on System Development," in *Scenario-Based Design: Envisioning Work and Technology in System Development* (J. Carroll, ed.), John Wiley and Son, 1995.

[7] R. Blanning, "A Decision Support Framework for Scenario Management," in *Proc. 3rd Intl. Conf. on Decision Support Systems*, 1995.

[8] T. Bui, D. Kersten, and P.-C. Ma, "Supporting Negotiation with Scenario Management," in *Proc. 29th Annual Hawaii Intl. Conf. on System Sciences*, vol. III, pp. 209–218, 1996.

[9] M. Lubars, C. Potts, and C. Richter, "A Review of the State of the Practice in Requirements Modeling," in *RE '93: Proc. Intl. Symposium on Requirements Engineering*, (San Diego, CA, USA), pp. 2–14, 1993.

[10] K. E. Emam and N. Madhavji, "A Field Study of Requirements Engineering Practices in Information Systems Development," in *RE '95: Proc. Intl. Symposium on Requirements Engineering*, 1995.

[11] P. Gough, F. Fodemski, S. Higgins, and S. Ray, "Scenarios — an Industrial Case Study and Hypermedia Enhancements," in *RE '95: Proc. Intl. Symposium on Requirements Engineering*, (York, England), pp. 10–17, 1995.

[12] L. Catledge and C. Potts, "Collaboration During Conceptual Design," in *Proc. 2nd Intl. Conf. on Requirements Engineering*, (Colorado Springs, Colorado, USA), pp. 182–189, 1996.

[13] K. Pohl, "The Three Dimensions of Requirements Engineering," *Information Systems*, vol. 19, no. 2, 1994.

[14] C. Rolland, C. B. Achour, C. Cauvet, J. Ralyté, A. Sutcliffe, N. Maiden, M. Jarke, P. Haumer, K. Pohl, E. Dubois, and P. Heymanns, "A Proposal for a Scenario Classification Framework," tech. rep., CREWS Report Series No. 96–01, 1996.

[15] CREWS Team, "Scenario use in european software organizations: Results of 15 site visits and questionnaires," tech. rep., CREWS Report Series No. 97–10, 1997.

[16] Rational Software Corporation (available on the WWW: http://www.rational.com), *Unified Modeling Language*, 1997.

[17] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Model and Design*. Prentice Hall, 1991.

[18] K. Kuuti, “Work Processes: Scenarios as a Preliminary Vocabulary,” in *Scenario-Based Design: Envisioning Work and Technology in System Development* (J. Carroll, ed.), pp. 19–36, John Wiley and Son, 1995.

[19] K. Pohl and P. Haumer, “Modelling contextual information about scenarios,” in *Proc. 3rd Intl. Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ ’97)*, 1997.

[20] M. Kyng, “Creating Contexts for Design,” in *Scenario-Based Design: Envisioning Work and Technology in System Development* (J. Carroll, ed.), pp. 85–107, John Wiley and Son, 1995.

[21] E. Dubois, P. D. Bois, and F. Dubru, “Animating Formal Requirements Specifications of Cooperative Information Systems,” in *Proc. 2nd Intl. Conf. on Cooperative Information Systems (CoopIS 94)*, 1994.