Scenario-based approaches are becoming ubiquitous in systems analysis and design but remain vague in definition and scope. A survey of current practices indicates we must offer better means for structuring, managing, and developing their use in diverse contexts.

Scenarios in System Development: Current Practice

Klaus Weidenhaupt, Klaus Pohl, Matthias Jarke, and Peter Haumer, RWTH Aachen

Scenarios present possible ways to use a system to accomplish some desired function. As scenario-based approaches attract increasing interest among requirements engineers, the literature on scenario methods, models, and notations proliferates. Object-oriented analysis and design use cases[1] highlight the benefits of creating concrete, use-oriented system descriptions prior to modeling function, data, and behavior. Proposed extensions and alternatives include adding structure to use cases;[2] the formal treatment of scenarios;[3] and the use of scenarios during documentation, discussion, and evolution of requirements.[4] Scenarios have also become popular in other fields, notably human–computer interaction[5] and strategic planning.[6]

Scenario use also pervades industrial practice, but comprehensive and expressive studies on the practical relevance of research techniques remain rare. Recent surveys are mostly broad in scope[7,8] or draw their conclusions from a single project.[9,10]

0740-7459/98/$10.00 © 1998 IEEE

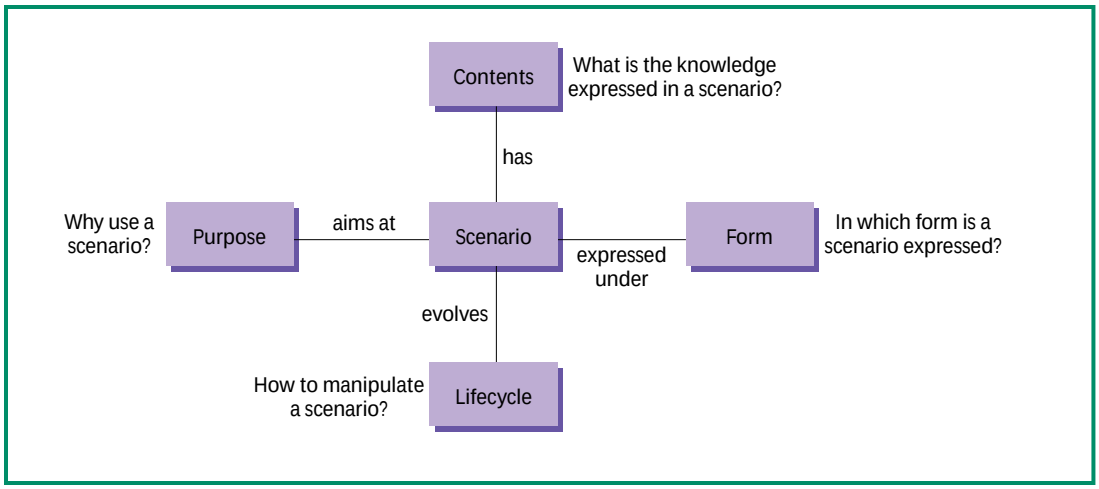

Figure 1. Four views on scenarios: form, purpose, content, and life cycle.

We in the European Esprit project Crews (Cooperative Requirements Engineering with Scenarios) are seeking a deeper understanding of scenario diversity, necessary to improve methodological and tool support for scenario-based requirements engineering. We follow a two-pronged strategy to gain this understanding:

- First, following the "three dimensions" requirements engineering framework developed in the precursor Nature project,[11] we developed a scenario classification framework based on a comprehensive survey of scenario literature in requirements engineering, human–computer interaction, and other fields.[12] We used the framework to classify 11 prominent scenario-based approaches.
- To complement this research framework, we investigated scenario applications in industrial projects through site visits with scenario user projects. The field's lack of theory compelled an exploratory study aimed mainly at understanding the diversity of scenario applications.

This article focuses on our site visits. We found that while many companies express interest in Jacobson's use case approach, actual scenario usage often falls outside what is described in textbooks and standard methodologies. Users therefore face significant scenario management problems not yet addressed adequately in theory or practice, and are demanding solutions to these problems.

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

Overview of Site Visits

We selected 15 projects in four European countries for site visits. Each 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 to one day. We recruited most sites directly or indirectly through software company representatives and independent consultants serving on the Crews Industrial Steering Committee.

We documented the site visits in minutes and summarized them in a technical report[13] highlighting each project's background, scenario characteristics, how scenarios were produced and used, benefits and problems or needs noted by the interview partners, and the main lessons learned from each visit.

Preparation

We used the Crews classification framework[12] to derive questions to characterize the scenarios used. This framework, developed from a comprehensive literature survey, views scenarios from four different angles: form, purpose, content, and life cycle (see Figure 1).

The *form* view pertains to a scenario's expression mode. Typical questions include whether a scenario is formally or informally described, and whether it is in a static, animated, or interactive form.

The *contents* view concerns the kind of knowledge a scenario expresses; its scope can vary from system-internal to organizational, and it can cover normal or exceptional cases.

The *purpose* view captures the role a scenario aims to play in software development, such as describing system functionality, ex-

TABLE 1
PROJECT CHARACTERISTICS

Project	Application Domain	Size	Developer
1	Medical information system	Small	Software contractor
2	Network documentation and management	Small	Software contractor
3	Business information system (banking)	Large	In-house
4	Business information system (public authorities)	Large	Consultant
5	Business information system (insurance)	Small	Consultant
6	Satellite communication	Medium	Consultant
7	Medical systems	Medium	In-house
8	Air traffic control systems	Large	Software contractor
9	Systems engineering for warships	Large	In-house
10	Radio telecommunication	Large	Consultant
11	Management of train networks	Medium	Consultant
12	CS applications for government and banks	Medium	Software contractor
13	Water invoicing management system	Large	Software contractor
14	CASE tools for bank applications	Small	Software contractor
15	Process engineering	Medium	Software contractor

ploring design alternatives, or explaining a system's drawbacks or inefficiencies.

The *life cycle* view considers scenarios as artifacts existing and evolving in time and pertains to technical handling, evolution, and project management.

Although these four views proved useful to structure the real-world observations, the focus on each shifted during our investigation. For example, in practice, form issues play a far less important role than research literature would suggest, while usage and life cycle aspects are much richer than anticipated from the literature survey.

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

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

To avoid bias, we let interview partners talk freely about their overall development process, scenario usage, and experiences. We used the interview plan as a checklist to ask for missing information when needed. In addition, we asked interview partners to provide us with concrete project material explaining their scenario generation and usage.

Project characteristics

Table 1 characterizes projects in terms of application domain, size, whether the project was performed in–house or by a software contractor, and whether we got our insights from a consultant. We classified projects as small (less than 10 person-years), medium (between 10 and 50 person-years), and large (greater than 50 person-years).

Table 1 shows how widely the projects varied in terms of application domain and project size. Clearly, scenario usage is not restricted to a specific application domain or project size.

Scenario characteristics

Table 2 characterizes scenarios usage according to the four views (form, content, purpose, and life cycle). We characterize the relevance of table entries using three attribute values: "F" for frequent occurrence, "M" for moderate occurrence, and "—" for no occurrence of that aspect in the project.

We found three categories of scenario content. *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 *internal system* refers to internal interactions among system components.

Scenario form seems to correlate with content, and five basic representation types dominate. Twelve projects used natural language heavily: one used narrative text without any structure, eight used structured text following a template or table structure, and three used a combination. Stakeholders used natural language mostly for context and interaction scenarios: 12 of the 13 projects focusing on interaction and context scenarios relied on textual notations. This holds to a lesser extent for images used as illustration, mostly in context and interaction scenarios, such as screen-dumps of user interface forms. In contrast, three of four projects dealing with internal scenarios employed more formal diagrammatic notations such as object inter-

TABLE 2
SCENARIO CHARACTERISTICS

	Scenario Facet	Project 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Form	Narrative text	F	M	F	M	M	M	M	F	M	—	—	—	F	—	M
	Structured text	F	F	M	F	F	F	F	F	F	M	—	F	F	—	F
	Diagrammatic notations	—	—	M	F	—	M	F	F	—	F	F	F	F	—	—
	Images	F	F	M	M	M	M	—	M	M	—	—	—	M	—	—
	Animations or simulations	—	—	—	—	—	—	—	—	F	—	F	—	—	F	—
	Typical size (pages)	1-3	1-2	2-8	3-8	10-20	3-20	5-20	/	/	1-2	3-10	1	10-200	*	10-50
	System context	M	M	M	M	—	—	—	F	F	—	—	—	F	—	F
Content	System interaction	F	F	F	F	F	F	F	F	F	—	M	F	F	F	F
	Internal system	—	—	M	—	—	—	—	M	F	F	F	—	F	—	—
Purpose and usage	Concretization of abstract models	F	M	F	F	F	F	F	F	F	—	—	F	M	M	F
	Scenarios instead of abstract models	—	—	—	F	F	—	—	—	—	—	—	—	—	—	—
	Interdisciplinary development	F	M	F	F	F	F	F	F	F	—	—	F	F	F	F
	Scenario use with prototypes	F	F	F	F	M	M	—	—	M	—	—	F	F	F	—
	Complexity reduction	F	F	F	F	F	F	M	F	M	M	f	F	F	F	F
	Agreement and consistency	F	F	F	F	F	F	F	F	F	F	M	F	F	F	M
	Scenario use with glossaries	—	—	F	M	M	—	M	—	—	—	—	—	—	—	—
	Reflection on static models	F	F	F	F	F	F	F	F	—	—	—	F	F	F	F
Life cycle and management	Partial views	M	M	F	F	F	F	M	F	F	F	F	M	F	—	M
	Distributed scenario development	M	—	F	F	F	M	M	F	F	M	—	—	F	F	M
	Review	F	F	F	F	F	F	F	M	M	M	M	M	F	—	F
	Traceability issues	F	M	F	F	F	M	M	F	F	M	M	M	F	M	M
	Basis for test cases	M	—	M	M	M	M	M	—	M	M	M	M	M	M	M
	Evolution	F	M	F	F	F	F	F	M	F	M	F	F	F	M	F

* 80% of the screens of the end product

action or message trace diagrams and animations.

Beyond their expected purpose for requirements elicitation and validation (not mentioned in the table), other usage aspects played a major role. For example, 13 projects depended on scenarios to make requirements concrete by lowering the abstraction level; two even changed their process to a scenario-based approach after encountering severe problems with traditional abstract model development. Nearly all projects stressed the role of scenarios in reaching partial agreement and consistency of the requirements specification.

Scenarios also helped reduce complexity and enforce interdisciplinary development. Somewhat surprisingly, 12 projects used dynamic scenarios to elaborate static models, such as object model population and validation. We also observed some unusual but interesting interactions of scenarios with

prototypes and glossaries, which we describe later.

Management issues included how to impose partial views on a scenario, handle distributed scenario development, enable quality assurance through scenario reviews, make scenarios traceable along the whole process, reuse scenarios as test cases, and evolve scenarios. We elaborate on these later.

Customers and users prefer to talk about concrete scenarios rather than abstract models.

To summarize, we found much more diverse scenario usage than you would expect based on the Unified Modeling Language view of scenarios as instances of use cases. Eleven of the 14 projects (P1-P8, P12-P14) claimed they followed a use case approach, but all 11 extended the textbook version significantly to make it work.

Scenario Usage in Practice

While scenarios enable interdisciplinary learning in requirements engineering, they also serve as means for divisions of labor, with significant consequences for project management and artifact integration. Table 2 provides a glimpse of the diverse scenario purposes and uses we found in practice; we summarize their benefits and shortcomings here.

Use scenarios when abstract modeling fails

Thirteen projects used scenarios to make concrete abstract models. Projects P4 and P5 first neglected systematic consideration of concrete system usage. When these projects failed to develop abstract conceptual models such as class models due to the complexity of the problem domain, they turned to scenarios to elicit and document customer requirements, successfully in both cases.

In project P4, after half a year it became evident that object model complexity and communication overhead within the requirements engineering team could not be managed anymore. Team members did not yet sufficiently understand the business processes to be supported, and because the customer did not understand the abstract models, validating them became almost impossible.

Developers therefore stopped defining the class model and established a scenario-based approach. They first used scenarios to divide the application domain into 15 topics, such as personal data capture and fines. They then assigned each topic to a development team of five to seven stakeholders. To understand each topic, they again used scenarios, which helped integrate the domain experts into analysis, since they found it easier to talk about concrete scenarios than abstract models. In addition, the focus of project management shifted from ensuring consistency across the whole project to achieving good partial understanding of individual topics.

Project P5 initially modeled their business processes using Petri nets. After defining more than 120 processes this way, interrelating a new model with the existing models and assuring consistency became almost impossible. Project managers therefore decided to stop formalizing the business processes, instead employing a less formal scenario-driven modeling approach. Just 27 scenarios captured all potential system usage, primarily because the Petri net-defined business processes were much more fine-grained than the scenario descriptions.

Scenarios enforce interdisciplinary learning

Scenario-based requirements engineering calls for interdisciplinary development with intensive and continuous developer–customer or developer–user interaction. We attribute this to three things.

First, writing scenarios requires detailed knowledge that only domain experts can provide and validate. The projects we evaluated mostly developed scenarios in an evolutionary manner: Brainstorming for potential scenarios led to topic-centered interviews at individual workplaces for elaborating individual scenarios in detail. In most projects, prestructured interviews became essential to keeping discussions with the domain experts focused and reasonably short.

Second, developers commonly describe current and future system usage scenarios using the language of the problem domain, vital to establishing good communication with nontechnical domain experts.

Third, customers and users prefer to talk about concrete scenarios rather than abstract models, since the scenarios more closely match their perception of the problem domain (except if formally defined). The intensive cooperation during scenario

creation and usage ensured the customer's early contribution to the developers' work.

Scenarios require the coexistence of prototypes

In two-thirds of the projects, scenario generation and usage interrelated with rapid prototyping or even building first versions of the new system. Particularly in projects P1-P4 and P12-P14, interview partners stated that combining both approaches yielded symbiotic effects. That is, without prototyping, the value of using scenarios would drop almost to zero, and vice versa. Project P1 developers considered the coexistence of a prototype and typical usage scenarios vital to selling the overall project to the customer and convincing the customer that the new system would meet his needs.

Typically, developers integrated scenarios and prototyping as follows. In early project stages, domain experts created a first set of scenarios to communicate application knowledge and their system vision to system engineers (such as requirements engineers, system architects, and designers). Based on these, system engineers developed a requirements specification, for example, a class model and behavior model, and then used this to develop prototypical implementations. Prototypes varied from simple paper-based user interface forms for validating the class model to a comprehensive scheme for validating real-time aspects of the system to be built.

The initial scenarios served to validate the prototypes and, indirectly, the requirements specification. Evaluating the prototypes led to detection of misunderstandings between the domain experts and the system developer, for example, if the system developer made the wrong abstractions based on the initial scenarios. Resolving such misunderstandings becomes easier with scenarios as a common basis for communication.

Equally important, validating prototypes against the initial scenarios let the domain experts validate the initial scenarios themselves to detect missing functionality, overspecifications, errors, and even unintended side effects. For example, in project P2 the developer mistakenly inferred a 1:1 association between two domain classes from an imprecise scenario statement. Although the domain expert reviewed the class model, no one recognized this error until the customer played with a user interface prototype that implemented the 1:1 association as two single-selection list boxes for establishing references between objects of the two classes. Comparing the prototype with the scenario, the domain expert detected the missing possibility to interrelate three or more objects.

Unearthing the gaps helped developers improve the scenarios or adapt the prototype or specification to the new detected requirements. This established an evolutionary system development process in which the domain experts worked closely with the software developers. This process depends, however, on consistency among requirements specification, scenarios, and prototypes—a major problem for which no systematic approach exists.

Scenarios reduce complexity

Our findings support the claim that the use case approach enforces a usage-oriented decomposition of the requirements analysis problem from the very beginning. Considering only one business process or task at a time reduces the number of system aspects the stakeholder must cope with simultaneously. We therefore see scenarios as structuring devices not only for requirements engineering but for the whole system development process. For example, projects P3-P6 and P10 used scenarios to divide work between designers, programmers, and even system testers.

To integrate the different use cases, several projects followed a sequential strategy. For example, project P5 restricted the system's first version to a single scenario. First, developers defined at a coarse-grained level the set of scenarios (business processes) the system should support. A management decision selected the most important scenario. Developers then analyzed that scenario and developed, tested, and installed a specification and first version of the system. They then chose the next scenario and repeated the procedure until the system functioned sufficiently. While this development approach caused some reworking, knowing that further scenarios must be integrated led to a reuse- and maintenance-oriented system design and implementation.

Scenarios and prototypes complement each other in a symbiotic manner.

Project P6 pursued a similar strategy, classifying scenarios as primary and secondary to simplify the system characteristics to deal with at a given time.

This also helped to define delivery stages.

Project P12 experimented with complexity measurements attached to the scenarios. Developers used these measurements to guess the development cycle's length, for example by assessing the number of involved objects, presence of subsystem interaction, number of actions, and so on. Management used this information to select the scenarios to be considered first, or those to be decomposed.

Scenarios facilitate agreement about system support for particular business processes.

The literature often emphasizes the role of scenarios in exception identification and exception handling. We observed this only in projects P8, P9, and P11, mostly due to their safety-critical nature. Other projects explicitly denied the need for exceptional scenarios, one argument being that reviewing exceptions would unnecessarily complicate discussions and distract domain experts from the main system goals. One interview partner contended that too detailed a discussion of exception scenarios might endanger a system's marketability. We find this standpoint fraught with danger—neglecting important exceptions in requirements engineering could result in customer dissatisfaction and higher costs in the long term.

Scenarios facilitate partial agreement and consistency

Stakeholders in a system under development have different goals; even their perceptions of reality vary significantly. Bringing all stakeholders together and reaching an overall agreement on the system usually proves too time-consuming or even impossible. Similarly, assuring that the system to be built conforms to all aspects of all existing systems in an organization often proves infeasible. Nearly all projects nonetheless used scenarios to drive the agreement process and establish partial consistency among existing systems. Apparently, reaching partial agreement and consistency suffices in practice, especially in large and complex projects (P4, P5, P8, P9, P10, and P13).

In contrast to overall complexity reduction—using scenarios to restrict system functionality—here scenarios reduced the scope of discussions and agreement processes. For example, using scenarios facilitated agreement about the performance of a particular business process and the support the system provides for this. Scenarios served also as means for discussing alternative solutions, grounding discussions and negotiations on real examples, and supporting trade-offs among design alternatives.

Dealing with a concrete business process scenario, the stakeholders could detect where the use of individual, conflicting taxonomies suggested disagreement, even though they agreed in principle. Scenarios also enabled the detection of different perceptions. In one business process, for example, one stakeholder assumed that another needed evaluation of the customer data he produced, whereas that stakeholder was wondering why she was even getting this information.

Link scenarios and glossaries

In project P3 (and to a lesser extent in P4, P5, and P7), intertwining scenarios with a project-wide glossary established a common understanding of terms used among different stakeholder groups such as developers, domain experts, and managers. When developing a new scenario, the developer employed key terms already defined in the glossary and established a reference to the corresponding glossary item. For terms not yet in the glossary, the developer introduced a new glossary item with a short, general definition.

We noted an interesting bidirectional relationship between scenarios and the glossary: The glossary items related to (parts of) one or more scenarios in which the item defined plays an important role. Technically, developers realized this by establishing a hypertext infrastructure in a project-wide intranet that linked corresponding scenario parts and glossary items.

This permitted defining items in abstract terms and relating them to a broader usage context. The relations established between terms and scenarios served to explain the definitions by a set of concrete usages represented in the scenarios. Those relations helped the developers and the domain expert adjust their interpretation of the key terms used and thereby reach a common, project-wide understanding. Moreover, these relations helped new project members become familiar with the project terminology.

In addition, the relationships established between scenarios and the glossary served as access paths for the scenarios themselves. For example, a stakeholder

interested in the use of a certain artifact in all business processes, such as a customer's creditworthiness file, could use the glossary–scenario relations to access all relevant scenarios such as "check creditworthiness," "decide on granting a loan," and so forth.

Reflect on static models

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

We identified two ways scenarios helped define and validate structural models. Projects P1, P3-P6, P12, P14, and P15 employed a sequential development chain starting with informal scenarios gradually transformed into conceptual structural models. For example, P1 identified domain objects and relations from scenarios' textual representations, then transformed them into a prestructure such as class-responsiblity-collaboration (CRC) cards customers or users could still understand. They then used the structural descriptions to define a more formal object model.

P2, P7, P8, and P13 developers created scenarios and structural models in parallel and independently, thereby establishing two descriptions of the future system. They cross-checked those descriptions to identify inconsistencies. Detecting conflicts and gaps led to improvement of both the object models and the scenarios.

Scenarios as Artifacts

In all projects, the developers viewed scenario creation, documentation, and evaluation as a major effort. Even developing normal, nonexceptional scenarios required significant effort, for several reasons. For one, domain experts normally do not reflect on their daily work. If more than one expert is involved, their statements often differ; statements might even change from day to day as the experts become more explicit about their current way of working and their visions of what the system will be.

Scenarios thus proved not to be the simple things many assume they are but complex artifacts that evolve over time and must therefore be managed. In addition, most developers recognized the need to structure scenarios according to certain criteria. We discuss some of these below.

Partial views on scenarios

Some scenarios affect many stakeholders or are so complex that a single stakeholder is only interested in part of them. In project P13 we found scenario descriptions up to 200 pages long; the need for different views on a single scenario also emerged in projects P3-P6, P8, P10, and P11. We identified the need for three such views:

- *Manager/developer views*. Whereas a manager often needs only to understand a scenario on an abstract level, 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 one providing enough information for managers and a detailed scenario description for domain experts and developers. Though developed in parallel, keeping these models consistent often became a major problem.
- *Partitioned scenarios for work distribution*. Scenario views formed a basis for distributing work within and among development teams. This can best be illustrated using message trace diagrams. In the radio telecommunications project P10, objects expressed in an MTD could be divided into parts defining subsystems such as base stations and several types of mobile stations. Project managers used such divisions to assign responsibility for the subsystems (parts of the MTD). The messages exchanged between objects of different parts described the interfaces between those subsystems. Interactions between objects in a single subsystem only interest the developers responsible for implementing, validating, and testing this subsystem. Defining such views is fairly easy for MTDs but more difficult for scenarios represented as prose or structured text. Despite the lack of a formal approach to defining such views (partitions) using structured or unstructured text, developers established such views in many projects.
- *Scenarios divided based on the underlying*

Scenarios are complex artifacts that evolve over time and must therefore be managed.

business process. Especially with large business processes, certain stakeholders are interested only in particular tasks or activities performed in those processes. In most cases such views were established informally. Only in project P9, where warfare scenarios were simulated, did developers define such views formally to enable the display of relevant information to the right stakeholders during scenario animation.

Managing distributed scenario development

In large projects and in the case of a complex problem domain (P4, P5, P8, P9, P10, and P13), several teams developed the scenarios in parallel. This complicated management of the distributed scenario development and the resulting scenarios.

Project P4 developers devised an interesting strategy for managing distributed scenario development, creating hundreds of scenarios individually across spatially distributed teams. They first divided the problem domain into 15 topics (business process parts). Each "topic team" then defined the scenarios for each assigned topic, and a scenario management team managed the developed scenarios.

Users need better tools to manage the scenarios and their relations.

A topic team could only reuse (part of) a scenario developed by another topic team if it was a public scenario. The team could also create a "raw scenario," which detects action sequences in their scenarios that either constitute a scenario of general interest or belong to another topic. After roughly specifying the raw scenario, the topic team passed it to the scenario management team, who identified the topic team responsible for elaborating the scenario.

Once fully defined, a scenario passed to the management team, who published it in the central library as a public scenario. For each public scenario, the management team maintains references to the team responsible for the scenario and the teams using the scenario. Based on this information, bilateral or multilateral meetings between the topic teams helped them adjust intertopic dependencies and assure consistent propagation of changes.

Reviewing scenarios

Similar to other software artifacts, scenarios are important project results that must be reviewed to establish high quality. Projects P1-P7, P13, and P15 established walkthroughs or inspection processes. Sometimes those involved in scenario creation conducted the review, sometimes domain experts not involved in scenario creation did it. In P4 and P7, experts from other domains were involved, such as quality assurance people, managers, and system maintenance people. Unfortunately, we could not obtain quantitative measurements about the use and impact of reviews because such information was not captured during the projects.

Scenario evolution

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

- *Top-down decomposition.* At the beginning of a project, developers often defined scenarios on an abstract level to get an overview of the system and its functionality. Later, they further elaborated these scenarios. A typical example is the definition of scenarios according to the abstractions used for defining business processes. First, the developers model the business processes affected, supported, or automated by the new system in a set of scenarios. Next, they refine them by tasks performed in each business process, and refine the relations defined between the business processes represented in the first scenarios to the task level. Finally, they enrich them by activities performed for each task.
- *From black-box to white-box scenarios.* Project P9 used black-box scenarios to represent the interaction of the system with its environment and interactions between objects (business processes, stakeholders, systems, and so on) in the environment. Once sufficiently understood, the development team extended these to white-box scenarios that also represented interaction between system components and thus information about system-internal aspects. Integrating white-box and black-box scenarios defined a set of complex scenarios describing the interaction of the environment with specific subsystems.
- *From informal to formal scenario definitions.* Often, developers first express scenarios in free-form prose, then impose a template structure to add more knowledge and detect inconsistencies and gaps. They then transform the structured texts into more formal representations such as message trace diagrams. This transformation adds more knowledge and helps detect inconsistencies and gaps. The safety-critical project P11 even transformed its MTDs into a formal protocol specification language. Keeping the different

scenario representations consistent proved difficult, since each transformation and correction of inconsistencies and gaps caused content changes. In addition, only a subset of stakeholders could understand the different representation formats—for example, adding a new interaction during the definition of a MTD or changing an existing interaction required back-propagating the changes to the textual scenario definitions for customer or user validation. Especially in large projects with a large set of scenarios, such changes are difficult to manage.

♦ *Incremental scenario development.* In most projects, a scenario's first version typically encapsulates knowledge at different levels of detail. Validating the scenarios through review techniques or checking the scenario against other domain models improves the scenario as more knowledge is added or parts are revised.

In all four types of scenario evolution, our interview partners face these problems:

♦ identifying the right level of granularity and abstraction during scenario development and usage,

♦ keeping the various scenario versions and the different representations used consistent, and

♦ supporting the management of changes across the different types and versions of scenarios, especially if a scenario encapsulates knowledge of different levels of abstractions.

Deriving test cases from scenarios

Nearly all developers we interviewed mentioned the need to base system tests on the scenarios defined with the customer during requirements engineering and system design. This means supporting the system developer in proving to the customer that the implemented system meets the requirements.

However, we found that current practice rarely satisfies this demand. Often, the scenarios developed during requirements engineering and system design were out of date by the time system testing began. Most projects therefore lacked a systematic approach for defining test cases based on scenarios. As mentioned earlier, the coverage requirements of test cases may also conflict with the goal of complexity reduction, which implies a small number of scenarios.

Traceability

Many developers also mentioned the need for better traceability support, with traceability seen as a prerequisite for establishing life-cycle-wide use of the scenarios defined during requirements engineering. Out-of-date scenarios were inconsistent with current design prototype versions and thus could not be used as a basis for test cases.

Traceability enables integration of change, helping users keep scenarios up to date. Developers should establish traceability between levels of scenario abstractions, views on scenarios, scenario versions, scenarios and prototypes, scenarios and the specification, and scenarios and test cases. Establishing traceability requires understanding the relations between the artifacts produced during the development project and the scenarios.

Use scenarios to relate system functionality to business process, and vice versa.

Users also need better tools to manage the scenarios and their relations. We observed a frequent lack of appropriate tool support and the inability to manually assure consistency between scenarios, and between scenarios and other artifacts. Hardly any two projects used the same tools for scenario management, except for the word processor! This indicates that no generally accepted tools exist.

IMPLICATIONS

Both the success stories from practice and the pitfalls and shortcomings of current methods pose new challenges for research. The site visits clearly highlight scenarios as pervasive artifacts used throughout a system's life cycle. They serve manifold purposes and must therefore be managed with more care than is usually discussed in the literature. The implications of these observations concern both practitioners and researchers.

Recommendations for the practitioner

Our investigation unearthed many excellent ideas for scenario use, among them the use of scenarios as "bridges":

♦ *To business processes.* Use scenarios to relate system functionality to business process, and vice versa. Concentrating on one usage aspect at a time and expressing relations between the system and business processes help developers manage the complexity of the application domain.

♦ *Between customer/user and developer.* Use scenarios as a communication medium between domain experts (customers, users) and the software and requirements engineers. Scenarios—in con-

junction with prototypes and glossaries—serve to transfer knowledge between both groups of people and to explain and illustrate terminology.

♦ *Between architectural and implementation components.* Use scenarios not only during requirements engineering but also during design and implementation to describe, test, and validate interfaces of various architectural and implementation components.

Scenarios are useful during design and implementation to describe, test, and validate components.

♦ *Between developers.* Use scenarios to distribute work among software engineers. For example, scenarios help divide the overall system, and therefore design tasks, into subsystems during the design phase. Scenarios can serve a similar purpose during implementation and testing.

♦ *Between software development phases.* Use scenarios, along with abstract models, to transfer knowledge across software development phases—for example, requirements engineering and design, or requirements engineering and system testing. Scenarios capture valuable background and environment information that enables the stakeholders of later phases to better understand the desired system.

♦ *Between structure and behavior.* Use scenarios to highlight the dynamics between various static components of a system defined using, for example, the Unified Modeling Language.[14] In well-understood domains, scenarios can even help users predict the influence of a system on its environment by, for example, envisioning changes in the environment caused by using the new system.

Challenges for research

While focusing strictly on scenario roles may clarify specific issues, it also tends to obscure the deep intertwining these bridging functions imply. 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 OO software engineering approach,[15] but all broadened the concept significantly, as some authors in the literature also suggest. For example, some system-internal and context scenarios involved extensions that Jacobson explicitly excluded (but others[16,17] predicted as necessary). Others generalized the structuring mechanisms offered or linked scenarios to design-level prototypes (as reported for the Danish Great Belt project[18]).

Most agree scenario development should be integrated with prototypes to validate the scenarios or to validate the prototypes based on scenarios. While this integration is treated formally in literature,[3,19] we observed a more informal use of scenarios in conjunction with prototypes and other software artifacts. Researchers must seek to better understand the relationships of scenario methods to other techniques and extend the scope of scenarios.

Evolution and management

The management of scenarios in the various projects and the classification of research contributions[12] illustrated similarities in managing the evolution of software artifacts and scenarios. The main difference is the customer's continuous involvement in scenario management, which requires keeping user-understandable scenario representations 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

Establishing comprehensive support for managing traceable scenario development and usage requires a better understanding of the relations between the scenarios and the other software artifacts.

Process guidance

Most project developers had to decide when to develop which kind of scenario, at which level of abstraction, and when to stop—that is, when the development and use of scenarios paid off. At all sites this emerged as a major problem in applying scenarios to real-world projects, and most developers saw scenario creation more as a craft than an engineering task.

The richness of scenario usage and management problems we encountered indicates the importance of comprehensive process guidance. This remains a distant prospect until we further investigate extensions and tailored applications of scenarios and provide more adequate tool support. ❖

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 Université Paris-Sorbonne. We are grateful to the members of the industrial advisory committee of the Crews project, most notably its chairman Peter Hruschka, for their support in establishing contacts to scenario projects.

REFERENCES

1. I. Jacobson et al., *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, Reading, Mass., 1992.
2. B. Regnell, K. Kimbler, and A. Wesslén, "Improving the Use Case-Driven Approach to Requirements Engineering," *RE 95: Proc. Int'l Symp. Requirements Eng.*, IEEE Computer Society Press, Los Alamitos, Calif., 1995, pp. 40-47.
3. P. Hsia et al., "Formal Approach to Scenario Analysis," *IEEE Software*, Vol. 11, No. 2, Mar. 1994, pp. 33-41.
4. C.Potts, K. Takahashi, and A. Anton, "Inquiry-Based Requirements Analysis," *IEEE Software*, Vol. 11, No. 2, Mar. 1994, pp. 21-32.
5. J. Carroll, "The Scenario Perspective on System Development," *Scenario-Based Design: Envisioning Work and Technology in System Development*, J. Carroll, ed., John Wiley & Sons, New York, 1995, pp. 1-18.
6. T. Bui, D. Kersten, and P.-C. Ma, "Supporting Negotiation with Scenario Management," *Proc. 29th Hawaii Int'l Conf. System Sciences*, Vol. III, IEEE Computer Soc. Press, Los Alamitos, Calif., 1996, pp. 209-218.
7. M. Lubars, C. Potts, and C. Richter, "A Review of the State of the Practice in Requirements Modeling," *RE 93: Proc. Int'l Symp. Requirements Eng.*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1993, pp. 2-14.
8. K.E. Emam and N. Madhavji, "A Field Study of Requirements Engineering Practices in Information Systems Development," *RE 95: Proc. Int'l Symp. On Requirements Engineering*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1995, pp. 68-80.
9. P. Gough et al., "Scenarios—An Industrial Case Study and Hypermedia Enhancements," *RE 95: Proc. Int'l Symp. Requirements Eng.*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1995, pp. 10-17.
10. L. Catledge and C. Potts, "Collaboration During Conceptual Design," *Proc. 2nd Int'l Conf. Requirements Eng.*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1996, pp. 182-189.
11. K. Pohl, "The Three Dimensions of Requirements Engineering," *Information Systems*, Vol. 19, No. 2, 1994.
12. C. Rolland et al., "A Proposal for a Scenario Classification Framework," to appear in *Requirements Eng. J.*, Vol. 3, No. 1, 1998.
13. Crews Team, "Scenario Usage in Industrial Practice: A Summary of 15 Site Visits," Tech. Report, Crews Report Series No. 97-10, Aachen, Germany, 1997.
14. Rational Software Corp., *Unified Modeling Language*, http://www.rational.com, 1997.
15. I. Jacobson, "The Use-Case Construct in Object-Oriented Software Engineering," *Scenario-Based Design: Envisioning Work and Technology in System Development*, J. Carroll, ed., John Wiley & Sons, New York, 1995, pp. 309-336.
16. K. Kuuti, "Work Processes: Scenarios as a Preliminary Vocabulary," *Scenario-Based Design: Envisioning Work and Technology in System Development*, J. Carroll, ed., John Wiley & Sons, New York, 1995, pp. 19-36.
17. K. Pohl and P. Haumer, "Modeling Contextual Information about Scenarios," *Proc. 3rd Int'l Workshop Requirements Eng.: Foundation for Software Quality (REFSQ 97)*, Presses Univ. de Namur, Namur, Belgium, 1997, pp. 187-204.
18. M. Kyng, "Creating Contexts for Design," *Scenario-Based Design: Envisioning Work and Technology in System Development*, J. Carroll, ed., John Wiley & Sons, New York, 1995, pp. 85-107.
19. E. Dubois, P.D. Bois, and F. Dubru, "Animating Formal Requirements Specifications of Cooperative Information Systems," *Proc. 2nd Int'l Conf. Cooperative Information Systems (CoopIS 94)*, 1994, pp. 101-112.

About the Authors

Klaus Weidenhaupt is a research assistant in the Information Systems group at the Aachen University of Technology, Germany, where he received a diploma in computer science in 1995. His research interests include scenario-based requirements engineering, process modeling, process-centered engineering environments, and flexible CASE tool architectures.

Klaus Pohl is a senior researcher in the Information Systems group at Aachen University of Technology, where he obtained a PhD in 1995. His research interests include requirements engineering, process-integrated engineering environments, traceability, and process improvement. He has been a workgroup leader for the Crews research project and for the Esprit Basic Research Action on novel approaches to theories underlying requirements engineering (Nature). He initiated the international workshop series on Requirements Engineering: Foundations of Software Engineering (REFSQ) and serves on the editorial board of The Requirements Engineering Journal. He is the vice-chair of the GI requirements engineering interest group and a member of GI, IEEE, and the IFIP requirements engineering working group 2.9.

Matthias Jarke is professor of information systems and chairman of the informatics department at Aachen University of Technology. He earned a PhD from the University of Hamburg in 1980. His research interests include development and usage of meta-information systems for cooperative design applications. He has coordinated three European Esprit projects in this field: Daida (knowledge-based information system environments), Nature, and Crews. He is Editor-in-Chief of Information Systems and was program chair of the 1997 International Conference on Very Large Data Bases.

Peter Haumer is a research assistant in the information systems group at the Aachen University of Technology. His research interests include requirements engineering with scenarios, hypertext, and multimedia as well as object-oriented modeling and component-based design. Haumer received a diploma in computer science from University of Aachen in 1995.

Contact Weidenhaupt at Lehrstuhl Informatik V, RWTH Aachen, Ahornstr. 55, 52056 Aachen, Germany, e-mail weidenh@informatik.rwth-aachen.de.