CREWS-SAVRE: Systematic Scenario Generation and Use[1]

N.A.M. Maiden, S. Minocha, K. Manning[2] and M. Ryan

Centre for Human-Computer Interface Design
City University, London, UK

Abstract

CREWS-SAVRE is a prototype software tool for systematic scenario generation and use. This paper reports on two interleaved strands of research and development of CREWS-SAVRE. The first is theoretical research into classes of exceptions in software-intensive systems. The second is the development of a software prototype which has been used to acquire requirements from current scenario users. This paper reports these user requirements and design implications for the current version of the prototype. The paper ends with a discussion of the advantages of integrating basic software engineering research with user-centred system design.

1: Introduction

There has been considerable recent interest in the use of scenarios to acquire and validate system requirements. However, few methods or software tools are available to achieve systematic scenario use, or even to generate useful scenarios in the first place. A recent analysis of scenario use by practitioners reveals that current commercial methods and software tools provide insufficient and unsystematic guidance for requirements acquisition and validation [1]. This paper presents a prototype software tool called CREWS-SAVRE (Scenarios for Acquisition and Validation in Requirements Engineering), developed as part of the European Union-funded ESPRIT 21903 'CREWS' (Co-operative Requirements Engineering With Scenarios) long-term research project. The CREWS-SAVRE software tool has been developed to encourage systematic scenario generation and use. This paper describes the innovative features of this tool and demonstrates them with an example of the tool's use.

Despite the recent interest in scenarios, development of new scenario-based methods and tools has been limited to a small number of academic institutions [2], research-oriented organisations [3], consultancies and method vendors [4]. Current software tools provide little prescriptive guidance for scenario generation and walkthrough. In most organisations, such tasks are still a craft undertaken by a few skilled individuals. To remedy this, CREWS aims to make scenario generation and use more systematic and hence more useful, widespread and cost-effective. To this end it synthesises existing theoretical research in disciplines such as computer science and cognitive science with software engineering research, described later in this paper, to inform design of the CREWS-SAVRE tool's functions and user guidance.

Basic research in software engineering has often resulted in products which ignore the requirements of their intended end-users [5]. To avoid this problem theoretical research has taken place in parallel with user-centred system design. An initial prototype of the tool was developed using intermediate results from the theoretical research and shown to requirements engineers who use scenarios for acquiring requirements for a range of software-intensive system types, from fast-attack submarines to complex medical systems. Where

---

[1] This research has been funded by the European Commission ESPRIT 21903 'CREWS' (Co-operative Requirements Engineering With Scenarios) long-term research project.

[2] Current address: Cairo Information Systems, Old Station, Station Rd., Great Chesterford, Saffron Walden, Essex, UK

possible these requirements were acquired in a systematic manner using SCRAM (the SCenario Requirements Acquisition Method), a method which itself uses scenarios to enrich the exploration of software prototypes. This "drip-feeding" of basic research results into a working prototype enabled the authors to acquire both end-user requirements for the product and obtain early feedback on the research results.

The paper is in four sections. The next section walks the reader through one usage scenario with the CREWS-SAVRE tool. Section 3 describes the theoretical research which underpins the design and implementation of the CREWS-SAVRE tool. Section 4 describes user feedback on and acquired requirements for the tool. Section 5 outlines future versions of CREWS-SAVRE, and discusses the integration of theoretical research findings into user-centred system design.

2: The CREWS-SAVRE Software Tool

The CREWS-SAVRE tool is a simple software tool which semi-automates generation of useful scenarios according to parameters entered by a user. A software automaton, known as the wizard, then enables and supports systematic walkthrough of generated scenarios to ensure that all scenario courses are explored and all system requirements are acquired and validated. During the walkthrough, the tool proposes generic requirement statements which the user can add to a requirement specification stored in commercial requirements management software tools. This paper focuses on a first prototype, developed to inform design of the subsequent versions of the tool.

Each scenario describes hypothetical situations in the environment in which the required system will be used. It acts as a 'test-script' for the requirements specification, to acquire new system requirements or to validate the completeness and correctness of existing ones. CREWS has developed theories of normal and alternative courses for scenarios to inform design of the tool to make scenario walkthroughs more systematic. These theories, described later, are operationalised in two databases and a dialogue controller whose design is informed by the CREWS-SAVRE process model. The first database contains patterns of normative behaviour which belong to all instances of categories of problem domain [6]. The second contains classes of non-normative events and states for all problem domains. The CREWS-SAVRE tool uses the contents of these two databases to provide the basis for 'intelligent' guidance for scenario generation and use. The tool has been implemented using Microsoft Visual Basic 4.0 under Microsoft Windows NT 4.0. The next section demonstrates the main functions and 'look-and-feel' of this prototype during a retrospective scenario analysis of part of 1992 London Ambulance Service (LAS) Computer Ambulance Despatching (CAD) system.

2.1: The LAS-CAD System

In 1992 the LAS was looking to computerise core activities such as incident management. The aim of the CAD system was to automate as much of the call-taking and resource identification, mobilisation and management functions as possible. A complex and integrated software and hardware system was designed and implemented. It included a computer-based gazetteer, a resource allocation system and an automatic vehicle location system. However, following the system's introduction, a number of social and technical problems arose. The entire system descended into chaos and the LAS reverted to the manual despatching system. The next section demonstrates the advantages of the CREWS-SAVRE tool for systematic scenario generation and use. It describes how the tool can generate scenarios which describe communication between the control room operator and ambulance crew members, and how a tool-led scenario walkthrough might have identified some of the now-recognised CAD system failures.

2.2: Generating and Using Scenarios for the LAS-CAD System

A fictitious requirements engineer called Roger is using the CREWS-SAVRE tool to generate scenarios for the LAS-CAD system. CREWS scenarios describe hypothetical situations in the environment in which the required system will be used. Each describes events, actions and agents in the LAS's operational system. Figure 1 shows the first stage of interaction, in which Roger generates a use case. CREWS-SAVRE makes an important distinction between use cases and scenarios. It treats a use case as a collection of actions and the temporal rules that govern how the actions may be linked together. In contrast a scenario is one sequence of events, the ordering of which is tied to the start- and end- events for actions in the use case. Therefore it is possible to have one or more scenarios for a use case, each defining one possible sequence of actions "through" the use case.

Roger can either specify use cases from scratch, retrieve reusable use cases from the database, or choose NATURE object system models [6] from which to generate a use case. In our example Roger chooses to generate a new use case from reusable actions in these models, which he first browses and retrieves from the database of over 250 such models, structured into 13 basic hierarchies. For example, there are a large number of object sensing models to model sensing different numbers of objects in different environments for different purposes. Roger can use keyword searches or simple data base browsing to retrieve models pertinent to the LAS-CAD problem domain, then reuse descriptions of actions, events, agents, objects, state transitions and states from these models.

Figure 1 shows that two reusable models have been retrieved by Roger. The first, object messaging (OM), describes actions, events, objects and states common to message communication between sender and receiver agents. The second, object reading (OR), describes events, actions, objects and states common to reading information from an object device. Roger instantiates these models using the simple action template shown in Figure 1 to describe the normal events (e.g. start to send), actions (read mobilisation decision) and agents (e.g. sender-device) during control-room to ambulance-crew communication.

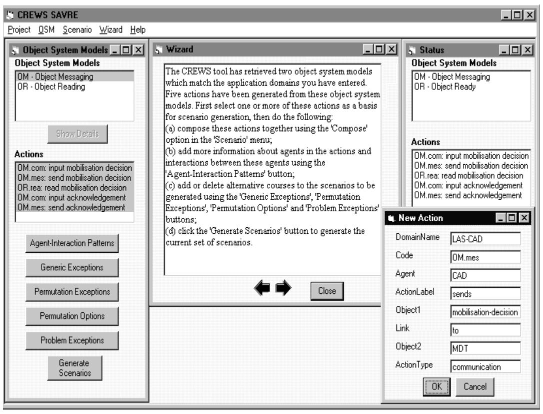

Figure 1: Selection of use cases and composition of actions using temporal links [13] as a basis for scenario generation.

CREWS-SAVRE's wizard then guides Roger to develop and generate one or more scenarios from the use case using the command buttons shown on the left-hand side of Figure 1. At each stage Roger selects one or more options from the lists made available by the tool. The command buttons enable Roger to add and remove exception classes which give rise to different forms of alternative courses in the generated scenarios:

- the 'Generic Exceptions' command button enables Roger to add simple alternative courses which arise from exceptions about events, actions, objects and states, for example a scenario can include events which repeat or do not occur;
- the 'Permutations Exceptions' button enables Roger to add alternative courses which arise from combinations of events, actions, objects and states, for example two events which occur at the same time;
- the 'Permutation Options' button allows Roger to add more complex, tailorable alternative courses about frequencies and other temporal links between events and actions;
- the 'Problem Exceptions' button enables Roger to add alternative courses to explore familiar but non-normative events and states involving human and machine agents, their communication and their interaction. Figure 2 shows selection from a subset of the available machine-machine communication exception classes. Roger chooses all exception classes for machine-machine communication, so that he can explore communication failures between the control room and ambulances in the generated scenarios.

The selected exception classes provide the input parameters for the scenario generation mechanism. Roger then presses the 'Generate Scenarios' button, and the tool generates a collection of scenarios for the use case. Now let us walk through one generated scenario with Roger.

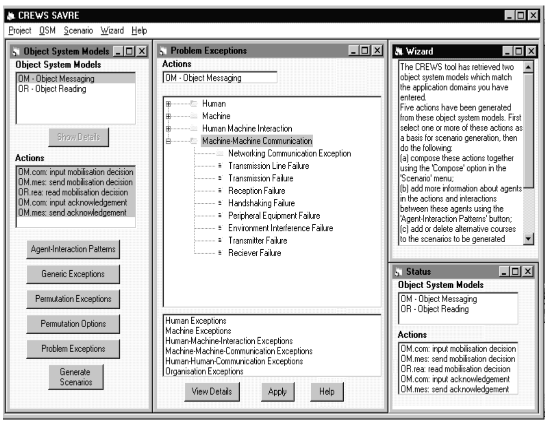

Figure 2: Selecting exception classes for inclusion in use cases.

Figure 3 shows CREWS-SAVRE's standard presentation during scenario walkthroughs. The top left-hand corner shows process guidance given by the wizard. The bottom left-hand corner shows the normal course of the generated scenario as a

sequence of events which start or end actions in use cases. The type of each action (P for physical, C for cognitive, M for communicative) is also shown. Roger uses the "Back<<" and "Next>>" buttons to navigate through the scenario and select events. On the right-hand side are alternative courses generated automatically from Roger’s earlier selection of generic, permutation and problem exception classes. These courses are presented in the form of what-if questions. For each selected event, the tool presents alternative courses linked by the tool to that event, and advises Roger to decide whether each alternative course is (a) relevant, and (b) handled in the current requirements specification. Figure 3 also shows part of the requirement specification to be validated.
In Figure 3 Roger is exploring event-5 (start [ambulance-crew reads information about the mobilisation decision from the mobile data terminal]). The tool proposes 5 alternative courses which were generated from the generic exceptions selected earlier. Roger is examining G1, what if event-5 does not occur. Each alternative provides a ’seed’ event or state that Roger can explore using techniques such as cause-consequence and fault-tree or event-tree analysis. Roger decides that this course is relevant but not handled in the current specification. For example in 1992, when an ambulance crew attended an incident, it might not be able to communicate with the control room because communication devices were in the ambulance. As a result Roger adds one or more requirement statements to the specification. The wizard guides him to explore the four other alternative courses G2-G5 in the same way.

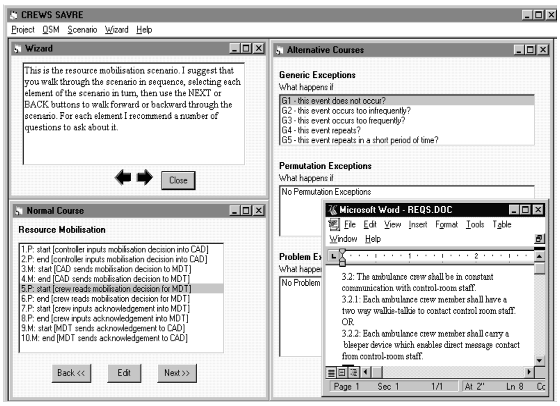

Figure 3: Scenario walkthrough, exploring an alternative course arising from a generic exception. Possible additional requirements are discovered from exploring this course include giving crew-members portable communication devices such as walkie-talkies.

Figure 4 shows another step in the scenario walkthrough. Roger again examines event-3, but this time he explores alternative course PE6, what if environmental interference occurs during the action. This time he decides that this course is relevant and not handled by requirements in the current specification. The reason for this is that London has communication blackspots. In 1992 these led the automatic vehicle location system to fail, the ambulance database to become incorrect and ambulance allocation to become sub-optimal. Furthermore, for this alternative course, the tool is also able to propose one or more candidate generic requirements shown in Figure 4. Roger can select one or more of these requirements and add them to the current requirement specification as he sees fit. In this example he chooses requirement GR2 "tolerate the environmental interference where possible - add toleration functions to other parts of the system", and edits it to fit the LAS-CAD problem domain, that is control-room staff shall be able to reposition ambulances manually if the system becomes incorrect.

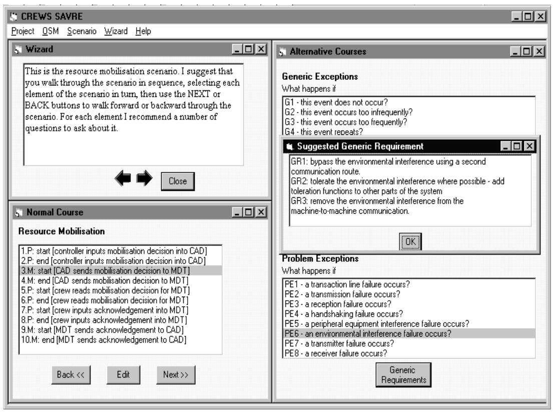

Figure 4: Scenario walkthrough, exploring an alternative course arising from a problem exception. At this stage the tool proposes three possible generic requirements to resolve or avoid the alternative course selected by Roger for investigation.

This simple example demonstrates some of the main functions of the CREWS-SAVRE tool. These functions include precise use case specification, automatic scenario generation, a software automaton which enforces systematic scenario walkthrough, and a database of exception classes for generating useful alternative courses. The next section describes the theoretical research which enables the CREWS-SAVRE tool to provide such functionality.

3. Theoretical Research Underpinning CREWS-SAVRE's Tool Design

CREWS is developing two theories of normative and non-normative problem domain behaviour relevant to socio-technical system use:

- generic domain models describing normative events and actions in different categories of problem domain. The ESPRIT 6353 'NATURE' basic research action posits a set of object system models which describe different patterns of objects, behaviours, structures and states of instances of about 250 categories of problem domains [6]. Object system models enable the CREWS-SAVRE tool to predict useful normal courses in scenarios;
- exception classes enable CREWS-SAVRE to predict useful alternative courses in scenarios, based on existing research from areas such as human error from cognitive science, machine failures from safety-critical systems and computer engineering, and interaction problems from human-computer interaction.

This paper outlines the current state of these exception classes and semantics for use case specification, and how this ongoing research informs CREWS-SAVRE's design.

3.1: Use Case Semantics

Unlike existing methods such as OOSE [4] and SOMATiK [7], CREWS treats a use case as a collection of actions and rules that govern how actions may be linked together. Each action has a start event, one or more end events, one or more involved agents, and one or more possible sub-actions. The action-link rules include strictly sequential (A then B), sequential but with the second action starting only after the first has started but necessarily not finished (A and meanwhile B), concurrent where there are no rules about ordering of the start and end events of the action (A and B), and alternatives between use cases (A or B). Concurrent start, concurrent end and parallel links have been added to version-2 of the CREWS-SAVRE tool. Returning to our example, Roger specifies the larger "Resource Mobilisation" use case as a strict sequence of 5 actions (OM.com THEN OM.mes THEN OR.rea THEN OM.com THEN OM.mes). Roger can also define mappings between objects in these 5 actions using a set of object-mapping rules. For example, the CAD agent in the OM.com action is the same as the CAD agent in the OM.mes action.

Use cases and scenarios: as already stated, the distinction between a use case and a scenario is central to the CREWS-SAVRE approach to scenario generation. Each scenario has a specific ordering of events that should satisfy the rules for the action-link rules from which it was derived. Therefore, more than one scenario can be generated from a single use case. The sequential ordering of events makes it simpler to present scenarios in a text format and to walk through scenarios in a systematic way.

How generation works in the CREWS-SAVRE tool: an algorithm has been developed to generate all possible normal course scenarios from a single use case. For each use case, the algorithm uses the action-link rules between actions to determine all possible normal sequences of events which start and end these actions. Each possible sequence is a candidate scenario which the user can accept as a useful scenarios or reject as not useful. A combinatorial explosion in the number of scenarios which are generated can be avoided via careful use of the action-link rules and warning messages given to the user during scenario generation. Figure 5 shows scenario generation from a simple use case with 3 actions. The action-link rules define a strict sequence between actions 1 and 2, and a partial sequence between actions 1 and 3, where action 1 starts before action 3 but can finish at an undetermined time. From this use case the algorithm generates 4 scenarios. The difference between each is the timing of event E3 to end action 3. Separating the generation of the normal and the alternative courses of the scenario in this way reduces the risk of a combinatorial explosion because the more numerous alternative courses are handled in a different way during the scenario walkthrough process.

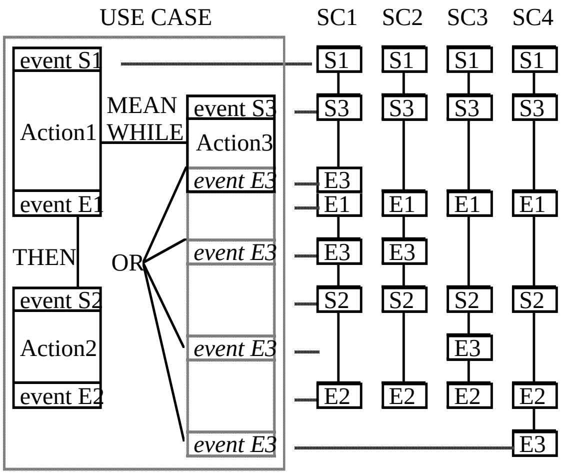

Figure 5: A graphical depiction of scenario generation from a simple use case. The use case definition leads to the 4 possible scenarios (SC1-4) shown. The difference between each is the timing of event E3 to end action 3.

3.2: Classes of Exceptions

Scenarios are useful for exploring exceptional situations. CREWS has synthesised a first version of exception class hierarchies from considerable research of human error in cognitive science, interaction failures in human-computer interaction and machine failures. The CREWS-SAVRE tool accesses the database of these classes during scenario generation and walkthrough.

Generic and permutation exception classes: CREWS defines an exception as:

> "a state or event that is necessary but not sufficient for the occurrence of an undesired or non-normative behaviour of the required system. If the exception is regarded as being sufficient for the occurrence of such a behaviour, it is said to be the cause".

CREWS distinguishes between generic, permutation and problem exception class types. Generic and permutation exception class definitions are based on the properties of events, actions, objects and states. For example, an event is an instantaneous real world phenomenon with properties such as totals and frequencies of occurrence, so CREWS defines generic exception classes about event frequencies and occurrence. An action is an operation that changes or maintains the state of the system or the outside world, with properties such as information inputs and information outputs, so CREWS defines exception classes for abnormal action inputs and outputs. Examples of alternative courses generated from such generic exception classes are shown in Figure 3. The current database contains 25 generic exception classes and 15 permutation exception classes.

Problem exception classes: CREWS has undertaken a more extensive classification of problem exceptions. It has developed 8 orthogonal classifications, each of which has a particular focus and literature. The first five classifications draw on CREWS definitions of actions and agents. An action can involve one or more human agents and/or one more machine agents, and a communication action can be between two or more human agents, two or more machine agents, or a combination of two or more human and machine agents. This minimum set of possible communication actions enables us to define 5 problem exception classifications:

- human agents: people often give rise to numerous alternative courses in a scenario. Our classified exceptions cause such courses from existing taxonomies of human error from cognitive science [8] and human factors research [9]. However, human exceptions can also be described by non-cognitive factors, for example physical exceptions (anatomical, sickness, fatigue, age);
- machine agents: machine themselves also give rise to numerous alternative courses, for example power failures and device failures. Such failures are classified as machine exceptions;
- human-machine interaction: alternative courses are generated from our classification of interaction failures from human-computer interaction and consequences from poor interface design [9];
- human-human communication: alternative courses often arise from communication breakdowns between people. Exception classes have been derived from theories and experience from computer-supported collaborative working [10];
- machine-machine communication: these exception classes give rise to alternative courses, see Figure 4.

Other exception class hierarchies include information-model exception classes (e.g. the information is incorrect, out-of-date etc.), work-domain exception classes (e.g. poor lighting, noise etc.) and organisational situations (e.g. policies, social climate, management attitudes). All of these classes enable the tool to generate useful alternative courses in a systematic manner. Figures 3 and 4 show 15 such alternative courses. The current data base contains just over 100 exception classes and the authors expect this number to rise over the duration of the project.

How alternative courses are generated in the CREWS-SAVRE tool: the tool uses a small number of rules to generate all permissible links between alternative courses and events in the normal course of a scenario using the type model shown in Figure 6. Each event starts or ends an action which involves agents and which can change the state of objects. The CREWS meta-model defines permissible types for each action (physical, cognitive, communicative), agent (human, machine, composite) and exception class (human, machine, different types of communication, etc.). Sixteen rules use these types to select alternative courses for each event (ev), action (ac), agent (ag) and problem exception class (pe) in a scenario, for example:

- IF ((ev1 starts ac1) OR (ev1 ends ac1)) AND ac1(type=cognitive)
  THEN (pe(type=human) applies-to ev1).

We believe that these 16 rules provide an important basis for constraining the generation of alternative courses for each event.

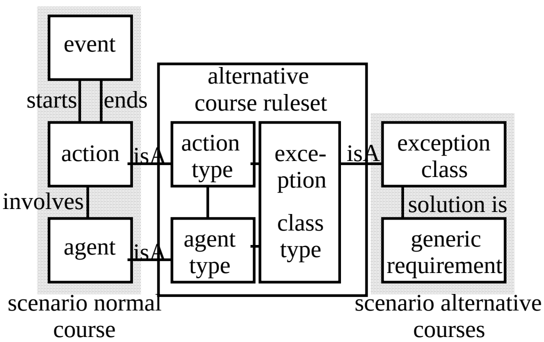

Figure 6: Event, action, object and exception class types and links modelled in the CREWS-SAVRE tool.

Proposing generic requirements: generic requirement statements such as those shown in Figure 4 are linked to exception classes. Generic requirement statements are mitigation and defensive mechanisms which handle the occurrence of one or more exception classes. The CREWS-SAVRE tool retrieves generic requirement statements for each inferred exception class. CREWS is developing a large set of such requirement statements to provide the tool with the capability to provide advice about possible solutions to problems identified during the walkthrough. Three examples are shown in Figure 4.

4: User-Centred Systems Design for the CREWS-SAVRE Tool

Prototype development of the CREWS-SAVRE tool has provided its authors with the chance to evaluate the tool, elicit feedback on the research and acquire user requirements for future versions. The tool was shown to 8 different groups of stakeholders using designer-led walkthroughs [11], academic presentations and informal discussions. The two formal walkthroughs were undertaken with Philips Research Laboratories (10 stakeholders present), who use use-cases to develop both medical systems and commercial appliances, and Logica UK (2 present) who are using scenarios to acquire requirements for a European air traffic control system. Less formal demonstrations took place with the Rational Software Corporation (2 present) and developers of a scenario authoring software tool for an emergency planning system (3 present). Three formal presentations of the prototype were made to 7 members of the CREWS industrial steering committee, to 2 system designers from a large aerospace organisation and to 7 members of a large defence systems engineering organisation. The scenario sequence shown during all presentations was similar to that shown in Figures 1-4. A facilitator prompted those present for feedback, criticisms and user requirements at different stages in the walkthrough of scenario generation in a lending library application.

Overall the feedback was very positive. All participants recognised the need for a software automaton to enforce systematic scenario generation and use. Most confirmed that the direction of research in CREWS is relevant to their practice and that the tool has features which can improve their requirements engineering processes. The following user requirements were acquired. The paper flags each requirement as already included in the shown first version of the CREWS-SAVRE tool (√1), at least planned for the current second version (√2) or not envisaged beforehand for the tool (x). Requirements for use case specification and scenario generation were:

- the tool shall generate scenarios which can scale to large and complex problem domains (x);
- the tool shall include thematic links between scenarios, so that interrupt events in one scenario are also start events in a second, different scenario (x);
- the tool shall provide graphical depictions of the use case structure and size prior to scenario

generation. Simple graphs can show exception class hierarchies linked to sequences of events, actions and agents in the normal course of the scenario (x);

- the user shall be able to attach domain-specific properties to agents and objects involved in actions. This will, in turn enable more refined selection of alternative courses. For example, electrical power failure is only asked about a machine agent which has a 'electrical' value for its power property (√2);
- the tool shall provide comprehensive configuration management and version control facilities to manage complex scenarios throughout the life time of the system (x);
- the system shall support full traceability both between use cases and scenarios and from use cases/scenarios to requirement statements (√1&√2);
- the tool shall link exceptions to the scenario as a whole as well as to specific events, actions, agents and courses in the scenario (x);
- the tool shall use the exception class hierarchies to generate domain-specific checklists for validating requirements, with or without a scenario to walkthrough (√2);
- the database of exception classes shall be tailorable to specific applications, for example to model common and important exceptions in fighter cockpit system usage (√2).

The following user requirements were acquired about scenario walkthroughs:

- the tool shall allow the user to record contextual information and comments about a scenario, for example about scenario users, the scenario author, background assumptions for the scenario, etc. (x);
- comments can be typed according to granularity, for example does this comment relate to the current event or is it a general comment about this scenario? (√2);
- the system shall output different forms of scenario, for example system specification for the designers, design alternatives, test cases for the test engineers, usage scenarios for authors of user guides, etc. (√1);
- the user shall be able to see changes in the state of the system as the user walks through the scenario. One means of achieving this requirement is presentation of inferred object states before and after each event in the scenario (√2);
- the user shall be able to adjust the granularity of the current view of the scenario whilst it is being presented and analysed (√2);
- the tool shall provide tailorable, parameterised guidance for the user depending on the user type and the resources available (e.g. time, people etc.) for the scenario walkthrough (x);
- the system shall provide rich question-and-answer templates for problem exceptions to enable a thorough investigation of their causes and effects during a scenario walkthrough (√2);
- the tool shall be integrated with a decision support system to advise the user about which risk analysis or other technique to use when predicting and investigating alternative courses (√2);
- the user shall be able to 'drag and drop' alternative courses into the normal course of the scenario if the users deem that the frequencies of these courses are sufficiently great (x).

The number of "x"s and "√2"s in the list indicate the value of undertaking user-centred system design alongside theoretical research. We are integrating these and other more detailed user requirements also elicited during these sessions into the current second version of the tool.

5: Discussion and Future CREWS Work

The prototype version of the CREWS-SAVRE software tool has enabled its authors to evaluate and elicit feedback from potential users about the tool's theoretical foundations and functionality. Scenario users recognise the need for a software automaton to enforce systematic scenario generation and use. Most confirmed that the direction of research in CREWS is relevant to their practice and that the CREWS-SAVRE tool has features which can improve their requirements engineering processes. The authors believe that CREWS-SAVRE is one of the first "knowledge-based" scenario development tools, a belief borne out from its demonstrations.

However, the current mechanistic approach to scenario generation can generate an unmanageable number of useless scenarios. To avoid this, the authors propose to integrate CREWS's domain-oriented approach with goal-oriented scenario generation around obstacles, which are similar to CREWS exceptions, which impede the achievement or maintenance of a goal [2, 12]. We shall use goal refinement approaches to direct scenario generation around critical exceptions identified first from the tool's database. Another weakness with the mechanistic generation is the lack of a theme or storyline to the scenario which other authors [12] consider important. To this end we are examining the design of an authoring tool as part of CREWS-SAVRE, with which the user defines the main themes and events of the scenario or use case, then CREWS-SAVRE automatically generates scenarios around this central theme.

Finally, we believe that integrating basic research with user-centred system design, as reported in this paper, has been a surprisingly effective way of undertaking basic research. Within CREWS we advocate a gradual evolution of the research, methods and software tools, rather than developing revolutionary ideas and products often found with the research-then-transfer approach [5]. The authors are also looking to make CREWS-SAVRE more domain-specific prior to its evaluation during case studies. Potts argues that "much of the research-then-transfer work that seeks to develop completely general tools is naive". If future case studies are successful they will demonstrate the importance of gradual evolution of research prototypes, as advocated by Potts.

Acknowledgements

We thank all who gave feedback on the CREWS-SAVRE tool. The research is funded by the European Commission ESPRIT 21903 'CREWS' long-term research project.

References

1. Weidenhaupt K., Pohl K., Jarke M. & Haumer P., 1998, 'Scenario Usage in System Development: A Report on Current Practice', IEEE Software, March 1998.
2. Potts C., Takahashi K. & Anton A.I., 1994, 'Inquiry-Based Requirements Analysis', IEEE Software 11(2), 21-32.
3. Gough P.A., Fodemski F.T., Higgins S.A. & Ray S.J., 1995, 'Scenarios - an Industrial Case Study and Hypermedia Enhancements', Proceedings 2nd IEEE Symposium on Requirements Engineering, IEEE Computer Society, 10-17
4. Jacobson I., Christerson M., Jonsson P. & Overgaard G., 1992, 'Object-Oriented Software Engineering: A Use-Case Driven Approach', Addison-Wesley.
5. Potts C., 1993, 'Software Engineering Research Revisited', IEEE Software, 19-28.
6. Maiden N.A.M. & Sutcliffe A.G., 1994, 'Requirements Critiquing Using Domain Abstractions', Proceedings of IEEE Conference on Requirements Engineering, IEEE Computer Society Press, 184-193.
7. Graham I., 1996, 'Task Scripts, Use Cases and Scenarios in Object-Oriented Analysis', Object-Oriented Systems 3, 123-142.
8. Reason J.T., 1990, 'Human Error', Cambridge University Press.
9. Nielsen J., 1993, 'Usability Engineering', Academic Press.
10. Shneiderman B., 1992, 'Designing the User Interface: Strategies for Effective Human-Computer Interaction', Addison-Wesley.
11. Sutcliffe A.G., 1997, 'A Technique Combination Approach to Requirements Engineering', Proceedings 3rd IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, 65-74.
12. Potts C., 1995, 'Using Schematic Scenarios to Understand User Needs', Proceedings DIS95: Designing Interactive Systems, Ann Arbor, 247-256.
13. Allen J., 1983, 'Maintaining Knowledge about Time Intervals', Communications of the ACM 26(11).