ScenIC:
A Strategy for Inquiry-Driven Requirements Determination

Colin Potts

College of Computing, Georgia Institute of Technology

potts@cc.gatech.edu

Abstract

ScenIC is a requirements engineering method for evolving systems, derived from the Inquiry Cycle model of requirements refinement and using goal refinement and scenario analysis as its primary methodological strategies. ScenIC rests on an analogy with human memory: semantic memory consists of generalizations about system properties; episodic memory consists of specific episodes and scenarios; and working memory consists of reminders about partially complete analysis activities. The methodology is to raise method-specific issues and resolution guidelines when activated by the state of episodic or semantic memory. The paper presents a summary of the ScenIC strategy and guidelines.

Introduction

ScenIC is a method to support the requirements-driven evolution of software. It builds on the Inquiry Cycle [1, 2], a model or requirements and design refinement. This paper presents a summary of ScenIC, its strategy and models.

Human Memory as a Model

A project's requirements documentation or models is its*memory*. Psychologists have long studied memory, categorizing it along functional and temporal dimensions. A persistent theme has been the distinction between working and long-term memories. Human memory suggests useful parallels with software development, because the *functions* of the corresponding memory systems are analogous. Human working memories are used during a task and are then forgotten. Project working memories consist of analysis task information. In ScenIC, these are pending issues and decisions. Within long-term human memories, a distinction is also made between semantic memories, which capture vocabulary, generalizations and principles, and episodic memories, which recollect events, situations and cases [3]. Project semantic memories are rules and generalizations that apply to all use situations. In ScenIC, these are goal, task and actor specifications. Project episodic memories are specific behaviors, traces, cases, and scenarios that seem noteworthy. In ScenIC, these are purposeful episodes and scenarios.

Attention Management and the Inquiry Cycle

Human skill involves directing attention. Correspondingly, a methodology or analysis strategy guides the allocation of attention to requirements and design issues. Previous studies of development projects have shown that project-level distraction leads to working-memory forgetting and the disruption of ongoing tasks [4, 5]. Thus ScenIC is a strategy for directing attention to requirements issues depending on the current contents of long-term memory.

ScenIC instantiates the Inquiry Cycle [1, 2], the cyclical application of the following steps: *expression* of semantic or episodic ideas, raising and resolution of issues (*criticism*), and consequent *refinement* of long-term memory (see Figure 1).

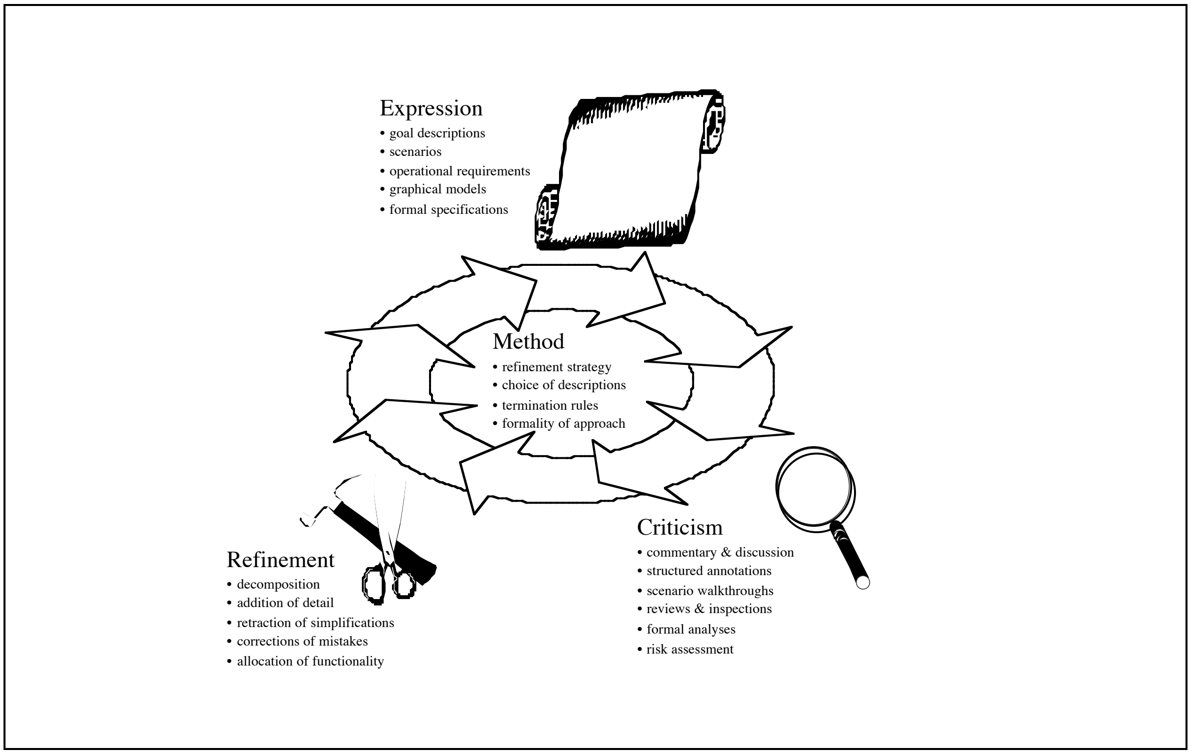

Figure1:The Inquiry Cycle, consisting of the activities expression, criticism and refinement ot software descriptions under the control of a refinement method. The example expression, criticism and refinement types are typical of ScenIC.

ScenIC instantiates this cycle by providing content in all three phases. Expression is supported by adopting semantic and episodic memory schemas. Criticism is supported by issue-raising guidelines that direct attention. Refinement is supported by resolution guidelines that suggest refinements. This model of mediated issue-raising and resolution derives from earlier research into design rationale [6, 7], but ScenIC supports ongoing discussion, not subsequent rationale.

The ScenIC Memory Schemas

Each of the three ScenIC memories has an entity-relationship schema. They are loosely coupled and are described separately. Detailed specification of schema elements is left undefined so that ScenIC may be applied at varying levels of formality.

Semantic Memory Schema

ScenIC semantic memory reflects information about the system. Because semantic memory changes during refinement, the "system" may include both "environment" and "machine" [8]. The starting point is a set of goals the system is to achieve, but semantic memory also includes behaviors and architectural commitments.

Semantic entities in ScenIC include actors, goals (of which there are two subclasses) and obstacles (see Figure 2).

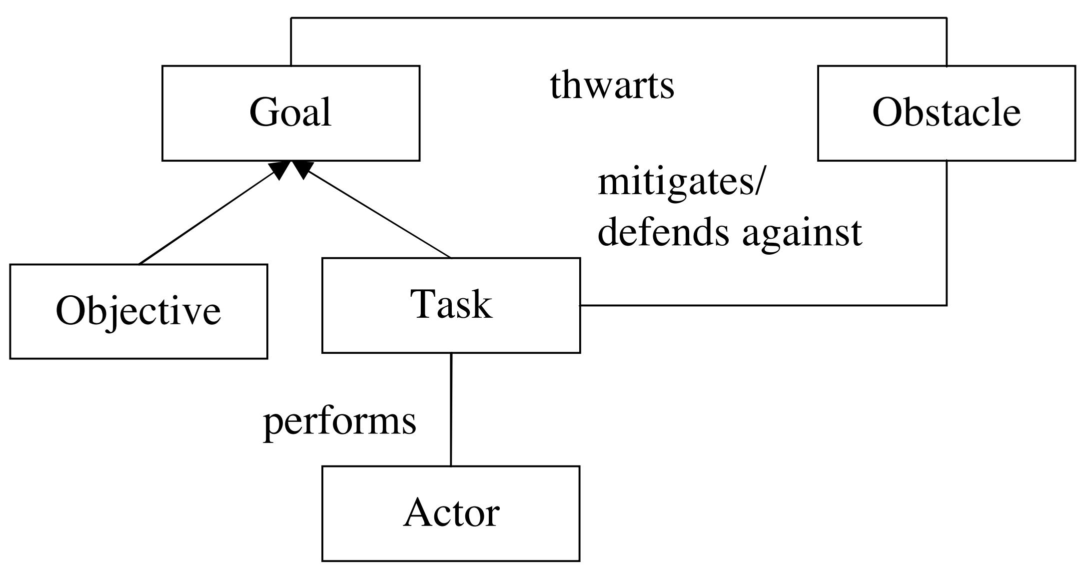

Figure 2: ScenIC semantic model schema in UML notation. (Boxes represent object classes; arrows represent specialization; arcs represent associations).

Actors are entities that participate in changes of state. Actors in the world include user roles, organizations, physical devices, external systems, and even Nature. Actors in the machine are putative architectural components, possibly derived from the system's existing architecture. The entire system may be an actor in the black-box case.

There are two kinds of *goals*: objectives and tasks. *Objectives* are expressed by a trajectory of improvement (e.g. "reduce customer complaints") or the preservation or prevention of states of affairs (e.g. "the elevator should never move with its doors open").

Tasks are goals that are stated in terms of achievement of a state (E.g. once a meeting request is entered into the system, a time and place for the meeting will eventually be scheduled or performance of an action).

Objectives cannot be directly achieved by the machine. Objectives are indirectly realized through combinations of tasks. For example, to prevent elevators leaving with their doors open, the controller checks states and inhibits actions. In contrast, tasks can be realized directly by machine or actors, although many may need to be expanded into subtasks. The specification of the system is therefore the collection of specifications for tasks allocated to the system or its constituent actors.

Goals may be thwarted by *obstacles*. Many obstacles are inherent in the goals themselves (e.g. unsatisfiability of a constraint), but others arise because of failure modes of actors to which tasks are allocated. Obstacles include not only failures, but also any non-normative behavior of actors in the world. In particular, people and organizations may behave in unanticipated ways that violate the assumptions required to achieve a goal.

Frequent or severe obstacles must be resolved either by defensive tasks or by mitigation tasks that the system does just because of the likelihood of obstacles but which are not essential requirements.

ScenIC Episodic Memory Schema

Semantic memory makes general claims (specifications) of system properties. However user and customer representatives cannot always easily generate and envisage consequences from specifications. As a result, the system may seem more desirable when specified than when implemented. These arguments have been voiced to justify prototypes [9], executable specifications and models, mockups and storyboards [10, 11], walkthroughs of work-practice data [12-14], and scenarios [1, 15-22].

ScenIC episodic memory consists of episodes, fragmentary scenarios that are related to the semantic corpus by their mention of actors and tasks. Although scenarios present a more compelling view of system properties than semantic information, they do not necessarily represent the system reliably. Their specificity means that they cannot cover the space of possible behaviors and so should be chosen to maximize their salience [23], the marginal increase in confidence about the system that they engender.

The episodic memory schema is shown in Figure 3.

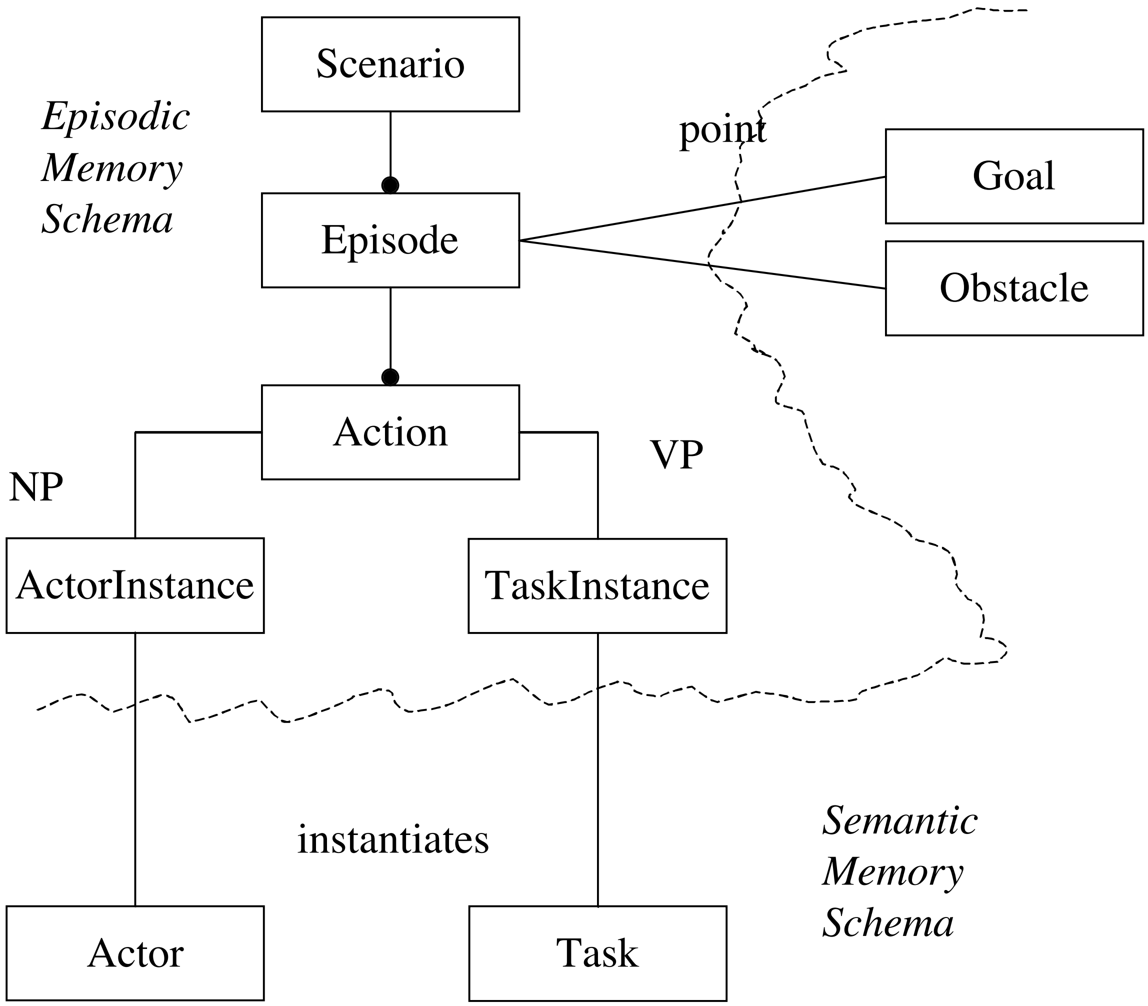

Figure 3: ScenIC episodic memory schema in simplified UML notation. (Filled circles represent aggregation. Associations are shown between episodic schema classes and semantic schema classes.

A ScenIC episodic memory consists of goal-directed and (possibly) obstacle-illustrating episodes bundled into scenarios. An episode is a sequence of purposeful task instances. The narrative structure of scenarios is a simplified story grammar or schema [24-26]. Scenarios may occur at varying levels of detail, and the actor and tasks may be instantiated with varying levels of concreteness. Scenarios are used to explore possibilities such as alternative allocations of tasks to actors, alternative obstacle resolution methods, and alternative future missions.

ScenIC Working Memory Schema

A project working memory is a record of pending issues or resolutions not yet acted on. In ScenIC, it consists of issues and decisions, although it could be extended to IBIS or more complex schemas [6, 7, 27-29]. Issues can be about anything, but methodological guidance is provided by posing standard issues about long-term information.

Working memory is action-oriented. Its contents remind and direct inquiry. Once addressed, though, reminders may clutter memory, making it difficult to find other information. Working memory could be captured or archived as a record of requirements rationale [27, 30], but there is little evidence that such reminders are easily understood later. Just as with human working memory, work must be done to re-encode and reorganize working memories if they are to be accessible and useful. For this reason, ScenIC working memories exists to support ongoing refinement only.

To support refinement, the issues and decisions must be systematically related to the refinement strategy. ScenIC's guidelines can be encoded as reminder templates and their activation contexts.

The working memory schema is shown in Figure 4.

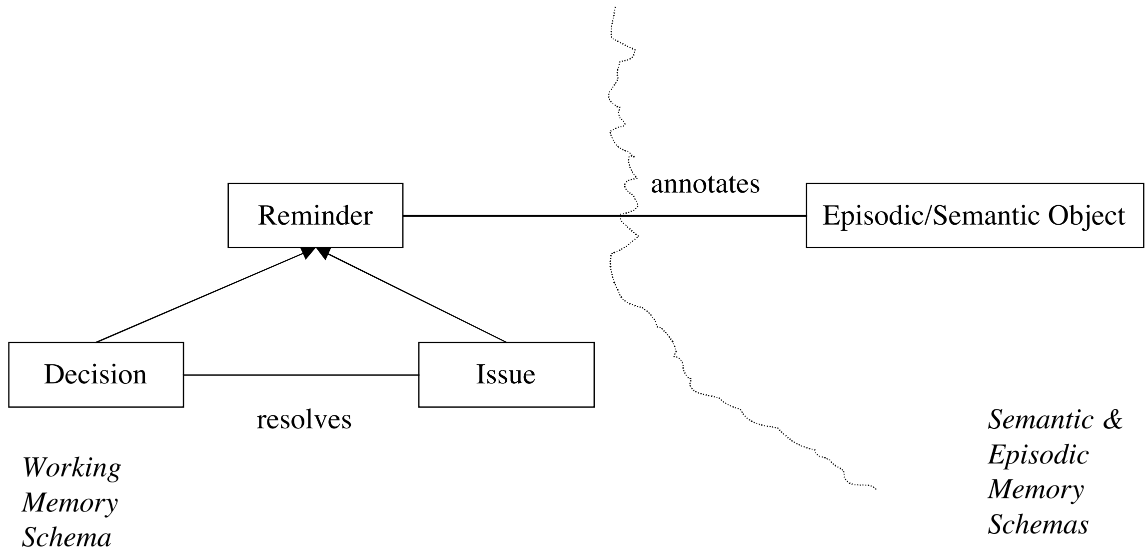

Figure 4: Working memory schema. Association "annotates" is shown between working memory schema classes and episodic and semantic memory schema classes, where "Episodic/Semantic Object" stands for any episodic or semantic schema class.

ScenIC Inquiry Strategy and Guidelines

ScenIC instantiates the expression phase of the Inquiry Cycle by providing episodic and semantic memory schemas for information about the proposed system. Criticism is supported by standard issues that are triggered by the state of the long-term memories. Refinement is supported by resolution guidelines that codify plausible decisions to resolve open issues. The framework is outlined in Table 1, the rows of which illustrate example applications of the framework for specific requirements situations.

Table 1: ScenIC strategy and guidelines framework. Issues are triggered by the state of the episodic and semantic memories. The decision reached may be suggested by a resolution guideline and is realized through changes to the semantic and episodic memories.

Triggering condition	Issue	Resolution guideline	Resulting change
Condition of the semantic memory or episodic memory.	Issue to raise concerning the system requirements.	ScenIC suggestions.	Condition of semantic memory or episodic memory.
E.g. content of a scenario.	E.g. is scenario correct?	E.g. ways to amend scenario or augment it with another scenario.	E.g. revised content of scenario or union of old and added scenarios.
E.g. an actor performing a task in an episode.	E.g. can actor fail, causing task incompletion?	E.g. risk analysis on actor failure likelihood and consequences.	E.g. Addition of actor failure obstacle and one or more scenarios containing an episode with the actor's failure as their point.

Scoping the System and Identifying Actors

Actors consist of external actors and internal actors. External actors are those that exist outside the designers' control, including human, organizational, natural and designed entities. The main heuristic used to judge whether to include an external actor is whether it performs significant actions within the scope of the system (see Table 2).

Table 2: Action identification guidelines

Triggering condition	Issue	Resolution guideline	Resulting change
No identified external actors	What are the external actors?	(1) Either (a) the actor should perform actions or have responsibilities relevant to the purpose of the system, or (b) The actor should interact directly with it (2) Look for people, teams, and organizations, devices and systems, external software systems, and elements of the environment considered as actors.	Extended context diagram.
No identified internal actors.	What are the internal actors?	If the internal architecture is unknown or unconstrained, do not decompose further.	Black-box, single-actor system.
Existing architecture	What are the internal actors?	(1) Include only components that correspond to physical (i.e. build-time) components. (2) Components should interact directly with the environment or with other components that do. (3) Components should either currently exist in the architecture, be additional components in the new version, or be components that result from a reorganization of the current architecture to which the design team is committed.	Gray-box, multi-actor system.

Example: the external and internal actors in a meeting scheduling system may be shown in the following context diagram (Figure 5), extended beyond the rules of Structured Analysis [31-33] so that

1. Interactions among external actors are noted when the system/environment boundary is not fixed and an understanding of the functioning of the system requires an understanding of them.
2. Interactions are in the form of dependencies, not data flows.
3. Actors may be specialized or sub-classed to simplify interaction descriptions.
4. Actors may be classified (person, system, flight vehicle, natural environment, etc.).

5. The black box is made “translucent” to show components responsible for system functions.

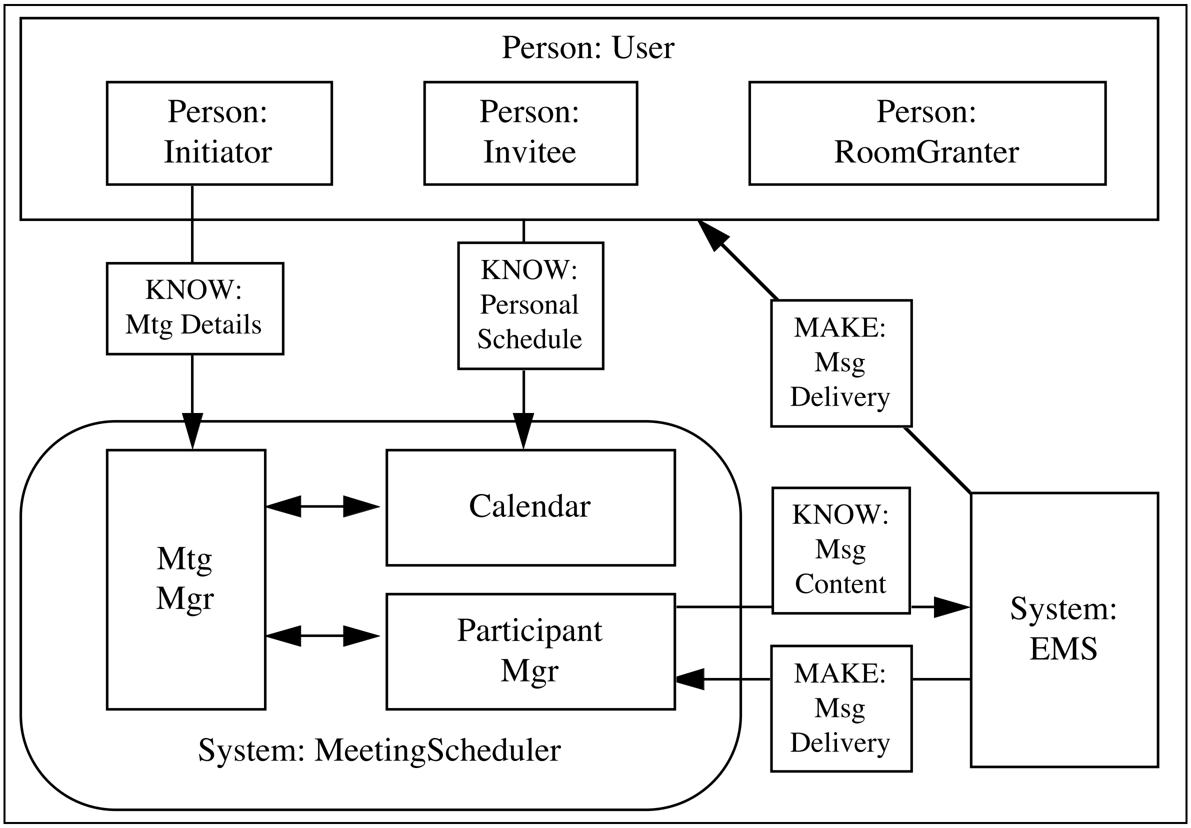

Figure 5:"Grey-box" architecture, showing external and internal actors and goal dependencies between them.

Goal Identification and Refinement

Goals are identified by reflecting on the system’s purpose, interviewing stakeholders, or inferring goals from background documentation. Anton’s dissertation [34] contains suggestions on identifying enterprise goals. Goals may also be identified from long-term memory content. Scenarios, for example, may illuminate previously ignored or underemphasized goals. Obstacles may suggest subsidiary tasks to reduce their likelihood or consequences. Table 3 lists ScenIC guidelines for identifying goals.

Table 3: Goal identification guidelines

Triggering condition	Issue	Resolution Guideline	Resulting change
No goals	What are the goals for the system?	Obtain from mission statements, direct questions to stakeholders, etc.	Semantic memory with goals.
No goals	What are the goals for the system?	Categorize goals using basic verb vocabulary	Goal list, with goals classifed as objectives (improvement or maintenance) or tasks, and further classified according to verb lexicon
Episode with behavior but no or partial point.	Why is this actor doing that?	(Default) Actions often define tasks one-to-one. Ask “why?” to identify neglected objectives.	(1) Episode with point. (2) Task ascribed to actor. (3) (possibly) higher-level objectives.
An obstacle that must be defended against or mitigated but which is not.	How should the obstacle be defended against? How should it be mitigated if it occurs?	Standard preventive maintenance and risk analysis methods.	Maintenance objective: “Avoid <obstacle>“ and its expansion into tasks. Achievement goal that encapsulates mitigation strategy.

A basic lexicon of verbs can be useful in identifying objectives and tasks (Table 4). Example objectives for the meeting scheduler that are consistent with the lexicon include "Maximize utilization of meeting rooms" and "Avoid boring meetings." Example tasks include "Satisfy meeting requests" and "Know availability of rooms."

Table 4: Types of goal and suggested verbs for labeling them

Object of goal achievement	Plausible verbs
Improving some condition, measurable or otherwise in the environment.	Improve, reduce, accelerate, maximize, minimize, optimize, satisfice, increase, decrease, eradicate
Maintaining some desirable condition in the environment	Keep, maintain, preserve
Avoiding some undesirable condition in the environment	Avoid, keep from, prevent, guard against, defend against
Enabling/preventing a specific actor to achieve a goal	Enable, let, empower, give (rights or control), disable, prevent, remove (rights or control)
Satisfying some condition	Satisfy, achieve
Bringing about a desirable state of affairs	Make (happen), create, do, achieve, fulfill, achieve, bring about
Bringing about a state of knowledge	Know, find out, ascertain, compute, work out
Bringing about a state of commitment	Decide, select, agree, converge (on)
Bringing about a state of knowledge in another actor	Tell, inform, notify, warn, alert, remind
Soliciting information from another actor to bring about a state of knowledge	Ask, request
Bringing about a state of knowledge concerning individual objects	Identify, distinguish, detect

Goals are expanded by means-end analysis. There are a number of guidelines for expanding maintenance objectives and tasks, but improvement objectives tend to be very domain-specific.

Table 5: Refinement of goals into expanded subgoals.

Triggering condition	Issue	Resolution Guideline	Resulting change
MAINTAIN p.	What are the sub-conditions of p?	Think of all the properties ($p_1..p_n$) that contribute to p.	MAINTAIN p_1 & ... MAINTAIN p_n
AVOID p (or MAINTAIN ~p)	What are the indicators of p?	Identify early-warning signs or operational definitions (q) of p and any tasks (anti-p) that counter-act those conditions.	DETECT q & MAKE anti-p
MAKE p	How to satisfy p?	Think of the knowledge [q] and action [r] sub-goals separately and combine. E.g. q may be a condition for satisfying r that has to be detected or known.	KNOW q & MAKE r
KNOW k	How to find out k?	(1) Ask another actor in a send/ack. cycle (2) Perform background computation	MAKE (a, asked[k]) & MAKE (a, ansd[d, k']) KNOW k_1 & ... KNOW k_n

Task Dependencies

Tasks often must be done in order, because they depend on initiating conditions. These temporal orderings are implicit in dependencies among tasks. In most situations, high-level tasks gradually consume information or depend on partial completion of dependent tasks. For this reason, it is best to defer analysis of task dependencies until tasks are detailed and could plausibly be allocated to actors. Sometimes, it becomes apparent that a task depends on the availability of some information without it being obvious how that information is to be provided. Generally, an additional knowledge goal will be needed.

Table 6: Task dependency guidelines

Triggering condition	Issue	Resolution Guideline	Resulting change
Task t = "MAKE p"	What are the preconditions for t (sometimes)?	What must be true for p to be true?	t <u>requires</u> r (where a task t_2="MAKE r")
Task t	What information is needed to achieve t?	Look for information that is used directly by t or, if that is not obvious, decision-making sub-tasks of t	(KNOW k & t_i...) <u>expand</u> t
Task t = "MAKE p"	What are preconditions for t in general?	Collate all the preconditions so identified	WHEN $q_1...q_n$ MAKE p

For example, in the meeting scheduling domain, the task dependencies shown in Figure 6 occur when a meeting is scheduled.

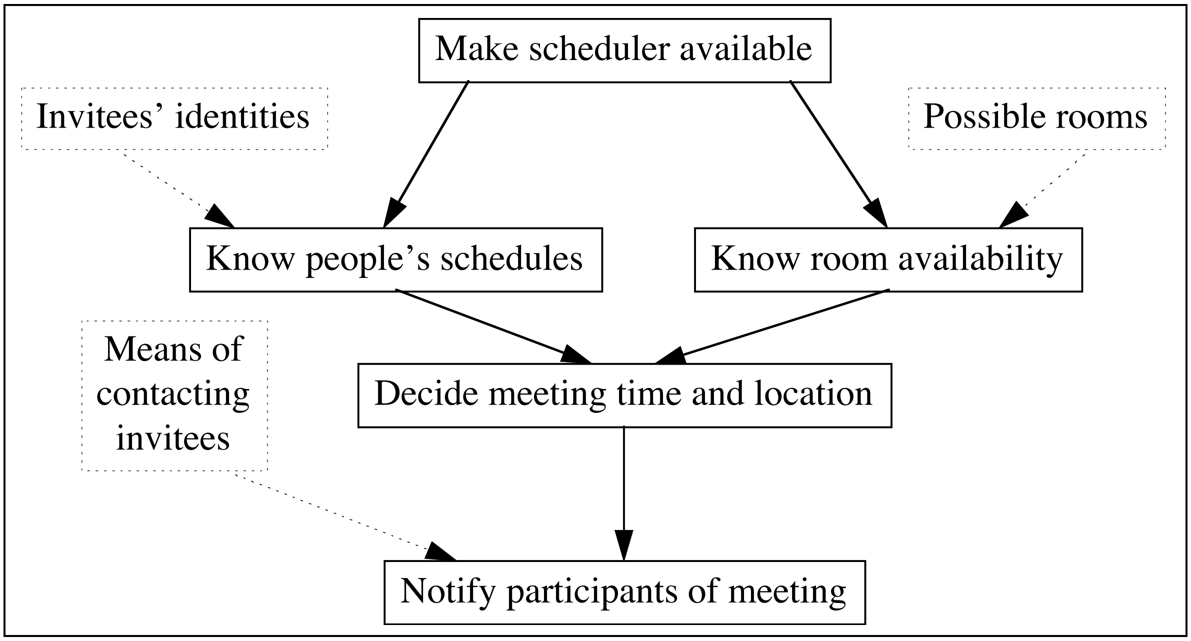

Figure 6: Task and knowledge dependencies in the meeting scheduler. Tasks and task dependencies are shown by solid boxes and arrows. Knowledge and knowledge dependencies are shown by broken boxes and arrows.

Allocation of Tasks to Actors

Which tasks should be done by the system and which by external actors is seldom pre-ordained. For example, the goal of deciding the best time and place for a meeting could be accomplished by a system automatically choosing them based on constraints imposed by the initiator and on calendar data, but an alternative would be for the information to be collated and presented to the initiator to make the final decision. Both systems could reasonably be described as supporting meeting scheduling.

Task allocation is aided by scenario analysis. Alternative allocations can be discussed in alternative scenarios, and the issues raised used in evaluating them. Possible allocations are partly determined by the current architecture and partly by current work practices. Choosing among alternative allocations reduces to three types of issues (Table 7).

Table 7: Task allocation guidelines

Triggering conditions	Issue	Resolution guideline	Resulting change
Actors and unallocated tasks	What task allocation is possible?	Constraints afforded by current architecture and work practices Creative invention	Candidate task allocations
Actors and unallocated tasks	What task allocation is feasible and economic?	Application-specific feasibility guidelines. Architectural evaluation using scenarios (e.g. SAAM)	Candidate task allocations
Candidate task allocation	What are the induced obstacles task allocation?	Consider failure modes specific to type of actor	Induced obstacles
Objectives and candidate task allocation	What are the effects of this allocation on the objectives?	Episode-by-episode scenario analysis	(Possibly) weakened objectives or revised candidate allocation

Obstacle Identification

Obstacles can be identified by a number of processes that range from the most informal to systematic engineering practices. Generally, obstacles are approached top-down and bottom-up (Table 8).

Table 8: Obstacle identification guidelines

Triggering conditions	Issue	Resolution guideline	Resulting change
Objectives and possibly allocations	What obstacles can block the satisfaction of this objective?	Generate episodes and allocations and test against objective	(Possibly) weakened objective and/or revised episodes and allocations
		Consider anti-scenarios or known negative cases	(Possibly) obstacles
Goals and possibly allocations	What could cause this goal to be blocked?	Consider inherent obstacle types (e.g. resource unavailability)	Obstacles
		Consider failure modes specific to actor type	

1. *Top-down identification of obstacle combinations that block objectives* Single obstacles are seldom responsible for an objective not being achieved. For example, "Avoid boring meetings" can be thwarted by a combination of factors. If this is a legitimate goal for the system, it becomes necessary to ask how people characterize meetings as boring, how they avoid attending them, and how a system might inadvertently make it difficult to avoid them. A minimal top-down strategy is generate-and-test: systematically question each allocation decision and episode illustrating it as to whether the decision undermines an objective. Another strategy is the construction of current-case anti-scenarios (e.g. why I ended up attending a boring meeting last Tuesday) or critical incident anti-scenarios (e.g. the interception by the USS Vincennes of an Iranian Airbus) that illustrate the non-achievement of an objective.

2. *Bottom-up identification of obstacles that block tasks*. Some obstacles are inherent in the problem domain, but many arise from failure modes of actors to which tasks have been allocated. For example, a best place for a meeting cannot be chosen if there are no rooms available. This is an example of an inherent resource-unavailability obstacle that is always a possibility when considering a

“decide” or “allocate” task. On the other hand, the obstacles that block achievement of the goal to find out when people are available to meet arise from human failings or system failure modes: people may not update their online calendars, a calendar may become corrupted, etc.

Table 9 categorizes some frequently occurring types of obstacles.

Table 9: Categories of obstacle with examples.

Categories of Obstacle	Example obstacles
Actor failure	Calendar database unavailable.
	Lens won't focus.
Medium failure	Email message does not arrive.
	Film speed is not encoded on cartridge.
Resource contention	There are no rooms available at that time.
	Call cannot be completed because there are no channels available on the satellite switch.
Replicate confusion	Wrong John Smith was invited to meeting.
Inherent goal conflicts	Taxpayer convenience is thwarted by need to audit returns.
	You can't have maximum depth of field and freeze action.

Elaboration of Objectives and Tasks

There are three ways to deal with an obstacle: (1) ignore it if it does not present a significant risk; (2) defend against it, ideally preventing it altogether; and (3) mitigate its effects, ideally recovering altogether. RPM [35] translates directly into inquiry-based questions, as shown in Table 10.

Table 10: Guidelines for identifying defensive and mitigation strategies

Triggering conditions	Issue	Resolution guideline	Resulting change
A hidden obstacle	Ca n the hidden obstacle be defended against?	To be worth defending against, the likelihood of a simultaneous obstacle to any defensive strategy should be acceptably low.	Defensive strategy consisting of early warning and remediation
Any obstacle	Are the consequences hazardous?	To be worth defending against, the defensive strategy must reduce the likelihood of the obstacle to an acceptable level.	Defensive strategy, consisting of early warning, remediation, and possibly auditable recording
Any non-hazardous obstacle	Are the consequences worth defending against?	To be worth defending against, the cost of the consequences should outweigh the cost of the defensive strategy.	A periodic or early-warning-based defensive strategy

In Rational Management [36], tables like Table 11 are constructed for listing defensive and mitigation strategies (called preventive and contingency actions, respectively).

Table 11: Identification of defensive and mitigation strategies using the Rational Management preventive and contingency action identification approach.

Obstacle	Probability	Defensive strategy	Residual probability	Mitigation strategy
Unavailability of room at satisfactory time	High	Tentative reservation of room based on quorum responses before all schedules are known.	Medium	Relax invitees' schedules by considering only their hard constraints, not their preferences.

In summary, mitigation is only worth considering when defense is likely to fail and the risk of the residual obstacle is still worth worrying about. In many cases, defense is more effective and less expensive than mitigation, but when the obstacle's consequences are mild and the cost of defense is high (perhaps in inconvenience to users), it makes sense to rely just on mitigation.

The Inquiry Cycle and Scenario Exploration

The point of exploring scenarios is to raise and resolve issues about system requirements. Many types of issues may be relevant in different situations: What else can happen at this point? Is this obstacle really worth worrying about? How frequent/severe is it? How else could we prevent this from happening / mitigate its effects? However, scenarios are not in working memory, because they are likely to be refined throughout development to support architecture evaluation, detailed design and test planning. Scenarios can be annotated by working-memory reminders just like semantic memory elements.

One benefit of scenario analysis is the serendipitous insights that a scenario may yield. When an issue that was raised from a scenario is resolved, the resulting changes may occur anywhere in episodic or semantic memory.

Identifying and Elaborating Episodes

Episodes follow directly from task structure. An episode may the point that several goals are satisfied and one or more obstacles occurs, but single tasks are a good starting point for identifying episodes.

In naming episodes, it is a good idea to be specific and domain-specific in naming and either to use the present participle form of a verb or a derived abstract noun. The guidelines are summarized in Table 12. Notice that episode identification is almost mechanical.

Table 12: Episode identification guidelines

Triggering condition	Issue	Resolution Guideline	Resulting change
MAKE p.	What episodes have making p as their point?	Syntactic rewriting ("making p") where MAKE p is not a lowest-level task "action") and preferably requires the collaboration of several actors in its satisfaction	EPISODE: making p POINT a makes p
EPISODE: making p POINT a makes p	How is p accomplished?	Goal expansion as above.	[TASK (b, p_1) & ... TASK(c, p_n)] <u>expand</u> TASK(a, p) EPISODE: making p BEHAVIOR: b p_1; ... c p_n
(EPISODE: making p) & other episodes	How are the preconditions established?	Think about all the conditions that must be true (tacit preconditions) for a task to be done or an episode to be started and then create or elaborate episodes that establish them.	TASK p requires q & MAKE q & EPISODE making q: POINT: make q

Scenario Identification and Composition

Normal-course scenarios are scenarios in which no episode illustrates the occurrence of an obstacle. An "anti-scenario" is a scenario that illustrates the non-achievement of a goal. They arise from considering not only obstacles but also the undesirable combination of episodes that detract from objectives. In the meeting scheduler, for example, what seemed to be a normal-course scenario could be recorded as an anti-scenario if it clearly undermined the goal of keeping participants' control over their schedules. Guidelines for scenario identification are shown in Table 13.

Table 13: Guidelines for identifying scenarios.

Triggering conditions	Issue	Resolution guideline	Resulting change
Existing episodes and goals	What is main-line use case for system?	Find highest-level tasks (not objectives) and orient scenario around task-satisfying episodes.	New scenarios and episodes.
Existing episodes and goals	What are initiating or system maintenance scenarios?	(1) Find administrative actors & construct scenarios from episodes that illustrate their tasks (2) Identify set-up tasks that satisfy preconditions of normal-course tasks. (E.g. look for registration actions).	New scenarios and episodes (and possibly actors)
Existing episodes, goals and obstacles	What are salient single-case exception scenarios?	Find significant obstacles for each task in normative case.	New scenarios and episodes (and possibly obstacles)
Existing episodes, goals and obstacles	What are salient obstacle-combination exception scenarios?	Find significant obstacle co-occurrences and perform risk analysis. If risk is above threshold,	Obstacle expansions (cf. fault trees).
Objectives	What are the tasks and obstacles that contribute to or detract from achievement of this objective?	Elaborate a scenario that illustrates the achievement or otherwise of an objective	Objective-demonstrating scenario or anti-scenario

Having identified obstacles in the manner described above, it is possible to compose any number of scenarios that contain normative (goal satisfaction) and exceptional (obstacle occurrence) episodes. But this leads to an explosion of possibilities, most of which are redundant or of marginal interest. We would like to put our early effort into the most *salient* scenarios, the ones will tell us the most [23]. Scenario composition is therefore a form of sampling like test-case generation. But scenarios differ from test cases in two important respects. First, to test a program, there has to be a program to test. But scenarios are used to discover and elaborate requirements, not just to validate the ones that are known, so any measure of scenario salience or coverage is a moving target. Secondly, we can assess the adequacy of test cases against an objective baseline, whereas any requirements baseline is subjective. The ScenIC solution is pragmatic: The set of goals and obstacles is taken as the baseline for assessing scenario coverage. As the analysis of scenarios helps in the identification of further goals and objectives, so the heuristics shown in Figure 14 should be reapplied, leading to the revision of existing scenarios, and possibly the composition of new ones.

Figure 14: Guidelines for identifying salient scenarios.

Triggering conditions	Issue	Resolution guideline	Resulting change
Goals, obstacles, episodes	How many and what scenarios are needed?	Normal-case coverage heuristic Single obstacle-coverage heuristic Check for satisfaction of objectives (at least absolute ones) Salient obstacle-combination scenarios (identified in turn using FTA or FMEA, etc.)	Salient scenarios
Critical incidents	How many and what scenarios are needed?	Model or sketch critical incidents as anti-scenarios	Salient anti-scenarios

1. *Normal-case coverage*. There should be at least one scenario for each normal case. Because of task dependencies there are usually alternative normal cases, some of which cover initiation or system-maintenance goals. In the case of the meeting scheduler, normal-case scenarios include what happens when a time and place can be chosen that meets everyone's preferences, what happens when a time and place can be chosen that meets everyone's constraints but not all their preferences, and what happens when a room is partitioned into two or a new employee is hired.

2. *Single-obstacle coverage*. Each obstacle should occur in at least one scenario.

3. *Coverage of objectives*. Ideally, every scenario should be assessed against every objective. For greater efficiency, it is advisable to divide the objectives into two: those that absolutely must be achieved, and those that are strong preferences but

could be violated. Objectives of the first type should be checked when elaborating every scenario, whereas those of the second type may be sampled randomly.

Discussion

Experience

Previous studies have explored and evaluated the application of the Inquiry Cycle [2] and GBRAM [37] to industrial projects. ScenIC has been or is currently being applied to the domains of web browsing and telephone call processing management.

Related and Previous Work

ScenIC is a component of MORALE (Mission-Oriented Requirements Analysis and Legacy Evolution) [38], which also includes methods for reverse engineering [39] and architectural evaluation [40].

ScenIC emphasizes scenarios, in which respect it resembles other efforts in requirements engineering [15, 16, 18, 19, 21, 23, 40, 41], object-oriented design [20, 22, 42], usability engineering [10, 11, 17, 23], and strategic planning [43-46]. Previous work has addressed the integration of scenarios into the Inquiry Cycle [1, 2].

ScenIC resembles other requirements-engineering approaches that start with goals [1, 16, 34, 37, 47-51]. It also incorporates insights and techniques from artificial intelligence [52], business planning [53], decision theory [54], human-computer interaction [55], management science [36], maintenance engineering [35], soft-systems methodology [56], and software metrics [57].

Obstacle identification, defense and mitigation owe more to project planning, industrial engineering and manufacturing than to software engineering. The methods of fault-tree analysis (FTA) and failure-modes effects analysis (FMEA) are described in numerous books on safety and reliability engineering [58]. There is a good tutorial summary of contingency planning in Rational Management [36].

Thinking of human short-term memory as a working memory stems from work by Baddeley [59], and the distinction between semantic and episodic memories from Tulving [3]. The ScenIC scenario schema derives from the story grammars of Thorndyke [26] and Rumelhart [25].

Design rationale, or the retention of organizational memories about why design decisions were made has been discussed for many years (see [60] for a collection of influential articles). There have been some promising accounts of the value of recording design rationale in large projects [30], but the notion of rationale "capture" fails to recognize important differences between the requirements on long-term and short-term memories that account for why not all short-term memories enter long-term memory: Short-term memories are superficially coded and organized, but are easy to find because there are few of them and they are the focus of attention; long-term memories, in contrast, are richly organized. Early investigators ignored the cost of reorganizing and integrating rationale so that it could be retrieved meaningfully later, a cost usually not borne by those who would benefit [61].

Levels of Formality

All three ScenIC memories could be implemented at varying levels of formality. For example, goals could be represented as text statements, graphical goal networks [51] or logical formulae [48]. Scenarios could be represented as narrative texts, structured or tabular scripts [20] or formal traces [41]. And reminders could consist of text annotations, semi-structured hypertext networks [6, 27, 29], or formal change dependencies.

Future Work and Tool Support

The Inquiry Cycle is a general approach to inquiry-based improvement of documented ideas. In ScenIC, these ideas are about a planned system or enhancements to an existing system, but the principle of using directed inquiry as a working memory applies to codifying any method for producing, organizing, refining or evaluating knowledge work.

We are currently developing tool support for the expression and criticism phases of the ScenIC Inquiry Cycle ("ScenIC View") and for ScenIC-specific issue-raising and and resolution guidelines ("ScenIC Ride"). These are based on the use of hypermedia annotations demonstrated in Ecolabor [62].

ScenIC itself does not suggest methods for coming up with goals or scenarios other than to ask experts, but sometimes official accounts conflict with how work happens. We are developing an Inquiry-Cycle formulation ("EthnIC") of fieldwork-based or contextual data analysis techniques, techniques that demonstrate promise in design and requirements analysis, but which do not provide a design vision. The idea behind EthnIC is to make field-based data gathering more procedural and design-focused. We have been heavily influenced by Contextual Inquiry [12, 14, 63].

The Inquiry Cycle can also be used to direct analysis of existing systems. MORPH [39] addresses the identification of terminal-based user-interface code and its re-targeting onto GUI abstractions. SAAM [40] is a method for walking through architectural scenarios. Such methods raise questions that are answered either by analysis or by practitioners scrutinizing summary data. We have not yet integrated these approaches into a coherent reverse-engineering strategy, and again the Inquiry Cycle (in the form of "ArchaIC") may provide the necessary glue.

Acknowledgments

Effort sponsored by the Defense Advanced Research Projects Agency and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-96-2-0229. The US Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, Rome Laboratory, or the US Government.

References

1. Potts, C., K. Takahashi, and A.I. Antón, *Inquiry-Based Requirements Analysis*. IEEE Software, 1994. **11**(2): p. 21-32.

2. Potts, C., *et al*. *An Evaluation of Inquiry-Based Requirements Analysis for an Internet Service*. in *IEEE Symposium on Requirements Engineering (RE'95)*. 1995. York, UK.

3. Tulving, E., *Episodic and Semantic Memory*, in *Organization of Memory*, E. Tulving, Editor. 1972, Academic Press: New York.

4. Catledge, L. and C. Potts. *Collaboration During Conceptual Design*. in *International Conference on Requirements Engineering (ICRE '96)*. 1996. Colorado Springs, Colorado: IEEE Computer Science Press.

5. Potts, C. and L. Catledge, *Collaborative Conceptual Design: A Large Software Project Case Study*. Computer-Supported Cooperative Work: The Journal of Collaborative Computing, 1996. **5**: p. 414-445.

6. Potts, C. and G. Bruns. *Recording the Reasons for Design Decisions*. in *10th International Conference Software Engineering*. 1988. Singapore: IEEE Computer Society Press.

7. Potts, C. *A Generic Model for Representing Design Methods*. in *11th International Conference Software Engineering*. 1989. Pittsburgh, Pennsylvania: IEEE Computer Society Press.

8. Zave, P. and M. Jackson, *Four dark corners of requirements engineering*. ACM Transactions on Software Engineering, 1997. **6**(1): p. 1-30.

9. Vonk, R., *Prototyping: The effective use of CASE technology*. 1990, New York, NY: Prentice Hall. 157.

10. Muller, M., *et al.*, *Bifocal tools for scenarios and representations in participatory activities with users*, in *Scenario-Based Design: Envisioning Work and Technology in System Development*, J.M. Carroll, Editor. 1995, Wiley: New York.

11. Ehn, P., *The Art and Science of Designing Computer Artifacts*, in *Scandinavian Journal of Information Systems*. 1989. p. 21-42.

12. Beyer, H.R. and K. Holtzblatt, *Apprenticing with the customer*. Communications of the ACM, 1995. **38**(5): p. 45-54.

13. Brun-Cottan, F. and P. Wall, *Using video to re-present the user*. Communications of the ACM, 1995. **38**(5): p. 61-70.

14. Holzblatt, K. and H. Beyer, *Making customer-centered design work for teams*. Communications of the ACM, 1993. **36**(10): p. 92-103.

15. Anderson, J.S. and B. Durney. *Using Scenarios in Deficiency-driven Requirements Engineering*. in *IEEE International Symposium on Requirements Engineering (RE'93)*. 1993. San Diego, CA: IEEE Computer Society Press.

16. Antón, A.I., W.M. McCracken, and C. Potts. *Goal Decomposition and Scenarios Analysis in Business Process Reengineering*. in *Advanced Information Systems Engineering: 6th International Conference, CAiSE '94*. 1994. Utrecht, The Netherlands: Springer-Verlag (Lecture Notes in Computer Science).

17. Carroll, J.M. and M. Rosson, *Getting Around the Task-Artifact Cycle: How to Make Claims and Design by Scenario*. ACM Transactions on Information Systems, 1992. **10**(2): p. 181-212.

18. Filippidou, D., *Designing with Scenarios: A Critical Review of Current Research and Practice*. Requirements Engineering, 1998. **3**(1): p. 1-22.

19. Rolland, C., *et al.*, *A Proposal for a Scenario Classification Framework*. Requirements Engineering, 1998. **3**(1): p. 23-47.

20. Rubin, K.S. and A. Goldberg, *Object behavior analysis*. Communications of the ACM, 1992. **35**(9): p. 48-62.

21. Sutcliffe, A., *Scenario-Based Requirements Analysis*. Requirements Engineering, 1998. **3**(1): p. 48-65.

22. Jacobson, I., *Object-Oriented Software Engineering: A Use-Case Driven Approach*. 1992: Addison-Wesley.

23. Potts, C. *Using Schematic Scenarios to Understand User Needs*. in *Symposium on Designing Interactive Systems: Processes, Practices, Methods and Techniques*. 1995. University of Michigan, Ann Arbor, Michigan: ACM.

24. Bartlett, F.C., *Remembering*. 1932, London: Cambridge University Press.

25. Rumelhart, D., *Notes on a Schema for Stories*, in *Representation and Understanding*, D. Bobrow and A. Collins, Editors. 1975, Academic Press.

26. Thorndyke, P.W., *Cognitive Structures in Comprehension and Memory for Narrative Discourse*. Cognitive Psychology, 1977. **9**: p. 77-110.

27. Conklin, J. and M. Begeman, *gIBIS: A Tool for all Reasons*. Journal of the American Society of Information Science, 1989(May): p. 200-213.

28. Kunz, W. and H. Rittel, *Issues as Elements of Information Systems*, . 1970, Institute of Urban and Regional Development, University of California at Berkeley: Berkeley, California.

29. Lee, J.-t. *Extending the Potts and Bruns Model for Recording Design Rationale*. in *15th International Conference on Software Engineering*. 1991: IEEE Computer Society Press.

30. Burgess-Yakemovic, K.C. and J. Conklin. *Report on a Development Project Use of an Issue-Based Information System*. in *CSCW'90: Computer-Supported Cooperative Work*. 1990.

31. DeMarco, T., *Structured Analysis and System Specification*. 1979, Englewood Cliffs, New Jersey: Prentice-Hall.

32. Ross, D.T., *Structured Analysis: A Language for Communicating Ideas*. IEEE Transactions on Software Engineering, 1977. **3**(1): p. 16-34.

33. Ross, D.T. and K.E. Schoman, *Structured Analysis for Requirements Definition*. IEEE Transactions on Software Engineering, 1977. **3**(1): p. 6-15.

34. Antón, A.I., *Goal Identification and Refinement in the Specification of Software-Based Information Systems*, in *College of Computing*. 1997, Georgia Institute of Technology: Atlanta, GA.

35. Moubray, J., *Reliability-Centered Maintenance*. 1992, New York: Industrial Press.

36. Kepner, C.H. and B.B. Tregoe, *The Rational Manager: A Systematic Approach to Problem Solving and Decision Making*. Second ed. 1976, Princeton, NJ: Kepner-Tregoe, Inc.

37. Antón, A.I. and C. Potts. *The Use of Goals to Surface Requirements for Evolving Systems*. in *20th International Conference on Software Engineering (ICSE '98)*. 1998. Kyoto, Japan: IEEE Computer Society Press.

38. Goel, A., *et al*. *MORALE: Mission Oriented Architectural Legacy Evolution*. in *International Conference on Software Maintenance'97*. 1997. Bari, Italy.

39. Moore, M. *Rule-Based Detection for Reengineering User Interfaces*. in *Proceedings of the Third Working Conference on Reverse Engineering*. 1996. Monterey, California: IEEE Computer Society.

40. Kazman, R., *et al*., *Scenario-Based Analysis of Software Architecture*. IEEE Software, 1996(November): p. 47-55.

41. Hsia, P., *et al*., *Formal Approach to Scenario Analysis*. IEEE Software, 1994. **11**(2): p. 33-41.

42. Texel, P.P. and C.B. Williams, *Use Cases Combined with Booch, OMT, UML: Process and Products*. 1997: Prentice-Hall.

43. Heijden, K.v.d., *Scenarios: The Art of Strategic Conversation*. 1996, Chichester, UK: Wiley.

44. Ringland, G., *Scenario Planning: Managing for the Future*. 1998, Chichester: Wiley.

45. Schwartz, P., *The Art of the Long View*. 1991: Doubleday.

46. Waltre, M., *Scenario Analysis: An Approach to Organisational Learning*, in *Department of Computer and Systems Sciences*. 1996, Stockholm University/Royal Institute of Technology: Stockholm, Sweden. p. 86.

47. Antón, A.I. *Goal-Based Requirements Analysis*. in *International Conference on Requirements Engineering (ICRE '96)*. 1996. Colorado Springs, Colorado, USA: IEEE Computer Society Press.

48. Dardenne, A., A.V. Lamsweerde, and S. Fickas, *Goal-directed requirements acquisition*. *Science of Computer Programming*, 1993. **20**(1-2): p. 3-50.

49. Green, S., *Goal-Driven Approaches to Requirements Engineering*, . 1994, Imperial College of Science, Technology and Medicine, Department of Computing: London.

50. McDermid, J., *Requirements Analysis: Orthodoxy, Fundamentalism and Heresy*, in *Requirements Engineering: Social and Technical Issues*, M. Jirotka and J. Goguen, Editors. 1994, Academic Press.

51. Yu, E. *Towards modeling and reasoning support for early phase requirements engineering*. in *Proceedings RE97: Third International Symposium on Requirements Engineering*. 1997. Annapolis, Maryland: IEEE Computer Society Press.

52. Schank, R.C. and R.P. Abelson, *Scripts, Plans, Goals and Understanding: An Inquiry into Human Knowledge Structures*. The Artificial Intelligence Series, ed. R.C. Schank. 1977, Hillsdale, NJ: Lawrence Erlbaum Associates.

53. Rouse, W.B., *Best Laid Plans*. 1995, Englewood Cliffs, NJ: PTR Prentice Hall.

54. Keeney, R.L., *Value-Focused Thinking: A Path to Creative Decisionmaking*. 1992, Cambridge, MA: Harvard University Press.

55. Card, S.K., T.P. Moran, and A. Newell, *The Psychology of Human-Computer Interaction*. 1983, Hillsdale, NJ: Lawrence Erlbaum Associates.

56. Checkland, P. and S. Holwell, *Information, Systems and Information Systems: Making Sense of the Field*. 1998, Chichester, UK: John Wiley and Sons.

57. Basili, V. and D. Rombach. *Tailoring The Software Process to Project Goals and Environments*. in *Ninth International Conference on Software Engineering*. 1987. Monterey, CA: IEEE Computer Society Press.

58. Storey, N., *Safety-Critical Computer Systems*. 1996, Harlow, UK: Addison-Wesley Longman.

59. Baddeley, A.D. and G.J. Hitch, *Working Memory*, in *Advances in Learning and Motivation*, G. Bower, Editor. 1974, Academic Press: New York. p. 47-90.

60. Moran, T.P. and J.M. Carroll, eds. *Design Rationale: Concepts, Techniques and Use*. Computers, Cognition and Work, ed. G.M. Olson, J.S. Olson, and B. Curtis. 1996, Lawrence Erlbaum Associates: Mahwah, New Jersey.

61. Grudin, J., *Evaluating Opportunities for Design Capture*, in *Design Rationale: Concepts, Techniques and Use*, T.P. Moran and J.M. Carroll, Editors. 1996, Lawrence Erlbaum Associates: Mahwah, New Jersey.

62. Takahashi, K., *et al*. *Hypermedia Support for Collaboration in Requirements Analysis*. in *Second International Conference on Requirements Analysis (ICRE'96)*. 1996. Colorado Springs, Colorado: IEEE Computer Society.

63. Beyer, H. and K. Holzblatt, *Contextual Design: Defining Customer-Centered Systems*. 1998, San Francisco, CA: Morgan-Kaufmann.