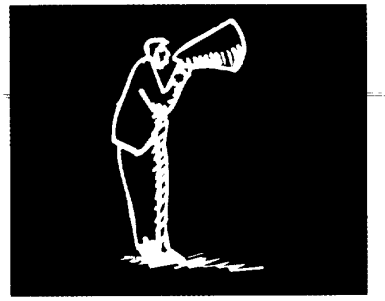


Inquiry-Based Requirements Analysis

COLIN POTTS, *Georgia Institute of Technology*
KENJI TAKAHASHI, *Nippon Telegraph and Telephone Corp.*
ANNIE I. ANTÓN, *Georgia Institute of Technology*

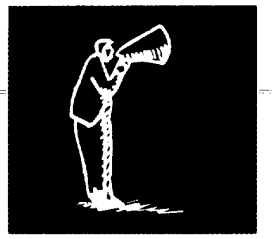
◆ *This approach emphasizes pinpointing where and when information needs occur. At its core is the Inquiry Cycle model, a structure for describing and supporting discussions about system requirements. The authors use a case study to describe the model's conversation metaphor, which follows analysis activities from requirements elicitation and documentation through refinement.*

Requirements analysis remains one of the most challenging areas for software developers, and the literature is filled with proposed methods. Some approaches emphasize linguistic details, transformations, and other formalisms as key to obtaining clear and valid requirements. Others, ourselves included, prefer to view the analysis process as essentially inquiry-based — a series of questions and answers designed to pinpoint where information needs come from and when.

This inquiry emphasis is appropriate for both contractual projects and market-driven product developments. In contractual projects, the customer usually writes a requirements document, but it has so many ambiguities, uncertainties and gaps that the developers must evaluate it carefully, incrementally refining and formalizing the information until they have produced a functional specification. In market-driven projects, there is no easily identifiable customer and no customer-sanctioned requirements document. In this case, the developers (usually in conjunction with marketing staff) must produce a specification from a vague statement of opportunities and goals. This, too, is an incremental inquiry-based process.

To support requirements identification, we have developed the Inquiry Cycle model, a formal structure for describing discussions about requirements.[1] The model uses a conversation metaphor that follows analysis activities from requirements elicitation and documentation through refinement. This metaphor directly addresses what a study of requirements practices in 23 project organizations revealed: The principal problems project teams face are communication, agreement about requirements, and managing change.[2]

0740-7459/94/$04.00 © 1994 IEEE



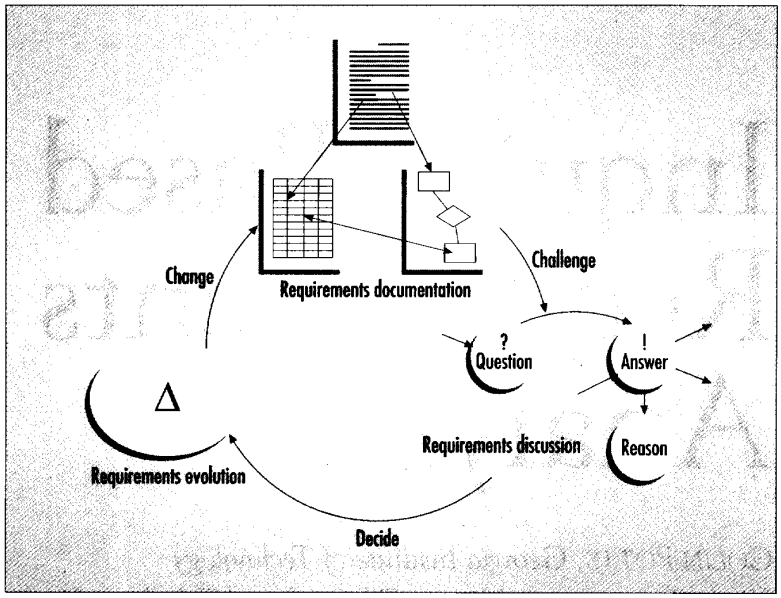


Figure 1. *The Inquiry Cycle model. Requirements documentation, consisting of requirements, scenarios and other information, is discussed through questions, answers, and justifications. Choices may lead to requested changes, which in turn modify the requirements documentation.*

The Inquiry Cycle model is a hypertext model because the pieces of information recorded and exchanged in requirements discussions are primarily textual (and therefore informal) but have well-defined interrelationships. Unlike most hypertext models, however, the Inquiry Cycle model is dynamic, capturing the ongoing elaboration process, not just a snapshot of the final requirements. It also directly supports inquiry and scrutiny, so everyone knows what information is missing and what assumptions are pending.

We have used the Inquiry Cycle model to analyze the requirements of a simple automated teller machine (described elsewhere[1]) and more recently a meeting scheduler. We present the results of the meeting scheduler application here and describe how scenarios can be used to improve requirements analysis. For tool support, we used FrameMaker, a commercial document processor with some hypertext features, although we have recently implemented a hypertext-based prototype tool, which we plan to use in a development project later this year.

The meeting-scheduler case study is a first step toward validating our model in large-scale studies. Although the meeting scheduler is a theoretical problem, not an actual system, much of the what we learned from it can be applied to real development projects. We investigated the types of questions asked about a set of written requirements, how those questions tend to be answered, and what role scenarios (particular cases of how the system is to be used) play in this process.

The meeting scheduler is a standard problem, and it let us examine analysis issues for both contractual and market-driven requirements. It illustrates contractual requirements problems because the starting point is a short requirements document that must be understood, made unambiguous, and refined. It illustrates market-driven project issues because it leaves open many decisions about what developers should implement.

INQUIRY CYCLE MODEL

Figure 1 shows the Inquiry Cycle model. Users of the model are called stakeholders. A stakeholder is anyone who can share information about the system, its implementation constraints, or the problem domain — including end users, indirect users, other customer representatives, and developers. We prefer this neutral term because although the model may be used by analysts working on customer-specific projects with existing requirements documents, the Inquiry Cycle is intended to support market-driven projects for which there may be no clear customer authority. For this reason, we deliberately avoided tying it to conventional roles or job titles.

As the figure shows, the model has three phases:

- *Requirements documentation.* The stakeholders write down proposed requirements. Each requirement is a separate node in the hypertext.
- *Requirements discussion.* The stakeholders challenge proposed requirements by attaching typed annotations.
- *Requirements evolution.* The stakeholders attach change requests to the requirements on the basis of the discussion, then refine the requirements when the change requests are approved.

It is this integration of requirements documentation, discussion, and evolution that distinguishes the Inquiry Cycle model from simpler conversation metaphors such as Ibis[3] or taxonomies of design transformations.[4]

Requirements documentation. Requirements is the label given to all requirements-related information — domain-specific assumptions, scenarios, project-planning commitments, and implementation constraints — as well as the stated requirements themselves.

In some projects, requirements analysis starts with some form of requirements documentation. In others, there may be a one- or two-page statement of goals. However vague the starting point, we assume that the result of using the Inquiry Cycle model will be a refined, agreed-on

specification, essentially a set of requirements that describes the desired system. If there is no requirements document, the model provides a systematic, incremental process for writing one.

There are several ways to analyze requirements in the documentation stage. If you have an existing requirements document, you can begin by reviewing that. If you don't, you must start from scratch to write down requirements on the basis of interviews, technical documentation for similar systems, and so on.

One valuable technique is scenario analysis. In the broad sense, a scenario is simply a proposed specific use of the system. More specifically, a scenario is a description of one or more end-to-end transactions involving the required system and its environment. Scenarios can be documented in different ways, depending on the level of detail needed. The simplest form is a *use case*, which consists merely of a short description with a number attached. More detailed forms are called *scripts*. These are usually represented as tables or diagrams and involve identifying an action and the agent (doer) of the action. For this reason, a script can also be called an *action table*.

Although scenarios are useful in acquiring and validating requirements, they are not themselves requirements, because they describe the system's behavior only in specific situations; a specification, on the other hand, describes what the system should do in general.

Despite the narrower coverage of scenarios, we focus on them, because we have found that scenario analysis and the Inquiry Cycle model complement each other. When challenging tentative requirements, stakeholders often pose what-if questions about the system's interaction with its environment that a scenario analysis can answer. Answering the what-if question by analyzing specific scenarios gives stakeholders insight into general requirements and helps in the refinement process.

Requirements discussion. There are three elements to this phase.

♦ *Questions.* Most discussions start because a stakeholder has a question about a requirement.

♦ *Answers.* These describe solutions to problems, in the form of candidate refinements or revisions that respond to the questions. A question can generate many answers. Answers provide stakeholders with a clearer understanding of the requirements and help them notice ambiguities, missing requirements, and inconsistencies.

♦ *Reasons.* Answers may require justification if they are not immediately obvious.

Requirements discussion can take place gradually and informally or in discrete bursts associated with formal review procedures.

ANSWERING WHAT-IF QUESTIONS BY USING SCENARIO ANALYSIS GIVES USERS INSIGHT AND HELPS REFINEMENT.

Requirements evolution. The ultimate result of a requirements discussion is a commitment to either freeze a requirement or change it. A change request may be traced backward to a discussion, which constitutes its rationale, and forward to the changed requirement once it has been acted on. Like discussion, the evolution phase may occur gradually and informally or in discrete bursts following a formal review and approval procedure.

When applying the Inquiry Cycle model, it is wise to remember several things:

♦ *It is not a rigid process model.* Stakeholders need not follow the model slavishly. Shortcuts are always possible. For example, a requirement may be changed after little or no discussion. An answer may be given even when there is no question, as often happens when stakeholders articulate assumptions that are not explicitly documented. Choices may lack rationale because stakeholders view the reasons as obvious. A change request may be executed directly without a recorded discussion. All these shortcuts may be perfectly reasonable in some circumstances.

♦ *There is no assumption about language or expressive style.* Because most requirements information is in natural language and free-form diagrams, the model embeds no assumptions about the specification language or style of expression. It is quite consistent, however, with the incremental development of formal specifications, object-oriented analyses, or structured-analysis models. For example, in object-oriented analysis, the requirements document evolves from a textual description of system requirements or typical use cases to a collection of object models and dynamic models. The requirements discussion consists of identifying and challenging candidate objects, attributes, associations, and operations. With the Inquiry Cycle model, whatever the representation or method, the analysis progresses toward a more precise specification through the incremental process of challenging and changing requirements.

♦ *Hypertext technology is useful but not mandatory.* Although the Inquiry Cycle model is an active hypertext model, hypertext technology is not needed to apply its underlying inquiry-based method. It does help, however, because the mapping from the model concepts to typical hypertext sys-tem concepts is fairly straightforward. Requirements (usually single paragraphs or sentences from the requirements document or an informal list), discussion elements (questions, answers, and reasons), and change requests can be stored as typed hypertext nodes. Typed links, corresponding to transitions in the model, can be added between nodes. For example, when a stakeholder poses a question about a requirement, a link is created between the requirement and question nodes. We call the process of creating links attaching a node.

TABLE 1
SCENARIOS FOR THE MEETING SCHEDULER

No.	Scenario
1	No conflicts
2	Slow responder
3	Late-coming participant
4	Dropout
5	Substitute active participant
6	Self-appointed active participant
7	Participant changes preferences before meeting is scheduled
8	Date conflict; initiator extends date range
9	Date conflict; participants extend date range
10	Date conflict; participants withdraw
11	Weak date conflict; participants exclude fewer dates
12	Room conflict
13	Scheduled meeting bumped by more important meeting
14	Participant tries to double-book
15	Conflict arises after meeting is scheduled
16	Meeting canceled

TABLE 2
SCENARIOS FOR NO CONFLICTS

No.	Agent	Action
1	Initiator	Request meeting of a specific type, with meeting info. (for example agenda/purpose) and date range
2	Scheduler	Add default (active/important) participants, and so on
3	Initiator	Determine three participants
4	Initiator	Identify one presenter as active participant
5	Initiator	Identify initiator's boss as important participant
6	Initiator	Send request for preferences
7	Scheduler	Send appropriate mail messages to participants (including additional requests to boss and presenter)
8	Ordinary participant	Respond with exclusion and preference sets
9	Active participant	Respond with exclusion and preference sets and equipment requirements
10	Scheduler	Request required equipment
11	Important participant	Respond with exclusion and preference sets and possibly location preference
12	Scheduler	Schedule meeting on the basis of responses, policies, and room availability
13	Scheduler	Send confirmation message to all participants and meeting initiator

CASE STUDY

We chose the meeting scheduler as a case study for the Inquiry Cycle for several reasons: First, the research community has adopted the meeting scheduling problem as a benchmark, and there is an existing two-page requirements document, written by Axel van Lamsweerde and his students of the Catholic University at Louvain. Second, the requirements illustrate problems typical of requirements for real systems. They specify policies that may not work well in every organization, there is ample opportunity to dispute different interpretations, and many important issues are left unresolved.

Finally, we chose this case study because specialized domain knowledge is not necessary to understand it. Most people grasp what it means to attend meetings and juggle a busy schedule, and most have had some experience using personal information managers.

Some would argue that the meeting scheduler is not a real system and in fact is a trivial exercise because the requirements document is only two pages of text. To this we answer that many significant real-world problems should be described as briefly as this initially. The real challenge for requirements analysis at this point is not to make specifications the size of *War and Peace* more readable or formal. Rather, it is to turn a very vague and high-level mission statement into a detailed specification as early as possible.

Problem description and assumptions. The meeting scheduler helps people schedule rooms and equipment for meetings. Each meeting is called by an *initiator* and may have *ordinary*, *active*, and *important* participants.

The requirements do not clearly define these terms. We assume that presenters are active participants and may have special equipment requests. We further assume that an important participant's attendance is more crucial than the attendance of an ordinary participant if there is a scheduling conflict. The initiator proposes some time constraints for the meeting, and the potential attendees respond with their available and preferred times.

Sometimes the scheduler can schedule the meeting; sometimes conflicts arise.

Analysis approach and tool support. We met regularly to review the requirements document and scenarios we constructed. To simplify data gathering and data analysis, we maintained electronic copies of the requirements, a discussion document (consisting of questions, answers and reasons), a change request list, and a collection of use cases and scenario scripts. We produced these using FrameMaker, taking care to cross-reference related text objects together. Some of the documentation and modification was done during meetings. Mostly, however, we resorted to note-taking, updating the documents between meetings.

Requirements documentation. We identified 16 scenarios for the meeting scheduler, which are presented as use cases in Table 1. The requirements were sufficiently detailed to give rise to 14 of the 16 scenarios. Of the remaining two, one scenario (Late-Coming Participant) arose as we analyzed another, Slow Responder, in which a potential participant does not respond in time for the meeting to be scheduled. In Late-Coming Participant, a potential participant becomes a participant after the meeting has been scheduled. The new participant's schedule may require that the meeting be rescheduled. Just as in Slow Responder, an action that affects an agent (person or thing doing the action) is delayed, requiring the system to recover. In Slow Responder, the delayed action is the participant's response; in Late-Coming Participant, it is the initiator's inclusion of the person in the participant set. In both cases, the recovery action is to reschedule.

Looking for analogies of this type is an effective way to identify important scenarios. The driving question each time is "What can go wrong with this action?"

For example, the scenario Self-Appointed Active Participant, in which a new active participant is added to the existing ones, is not covered by the requirements, but Substitute Active Participant, in which a new active participant substitutes for someone else after scheduling has commenced, is included. This scenario is similar enough to suggest the Self-Appointed Active Participant scenario because the only difference between them is that, in the first, the new active participant is substituted; whereas, in the second, he or she is added.

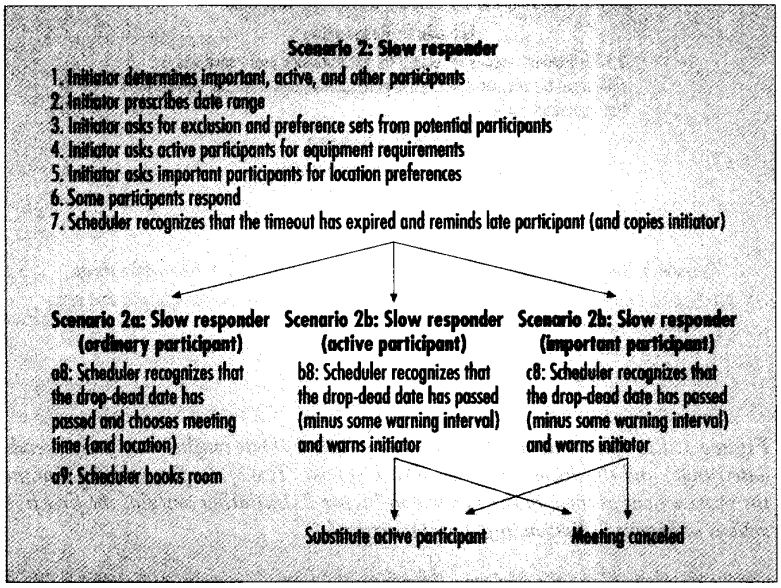


Figure 2. *Scenario family for scenario 2 in Table 1 (Slow Responder). A scenario family has subcases; in this scenario, they are ordinary (2a), active (2b), and important (2c). These subcases share fragments — in this case, the first eight actions, but then branch into different outcomes.*

This discovery strategy seems quite general: whenever a membership-changing action occurs to a collection of entities, stakeholders should consider if the system can handle other, similar changes and, if so, analyze scenarios that explore them. If not, stakeholders should refine the requirement to state precisely what input will be rejected.

Representing scenarios. Natural-language summaries are at the extreme of informality. More formal are tabular representations. Table 2 gives a more formal representation for the No Conflict use case (first row) in Table 1. Tables, unlike natural language, encode temporal sequences unambiguously.

In our case study, we represented scenarios as sequences of actions at two levels of complexity:

- *Complete scenarios or families of scenarios.* A scenario can have subcases, which then makes it a family. Figure 2 shows an example of a scenario family for scenario 2 in Table 1 (Slow Responder) The three subcases, ordinary, active, and important, share an initial set of actions but then branch.
- *Episodes or phases.* The shared actions in different cases are called episodes or phases. Episodes are also sequences of actions but are more fine-grained. For example, the initial actions of scenario 2 in Figure 2 represent the scenario's initiating and responding phases. Figure 3 shows five episodes, two of which (Initiating and Responding) have actions identical to those in Figure 2, even though the five episodes are for a different scenario (scenario 8 from Table 1), in which the initiator resolves a scheduling conflict by suggesting other times. This scenario arose because the requirements do not stipulate how long the scheduler should wait or what actions it should take to remind the tardy participant or the initiator.

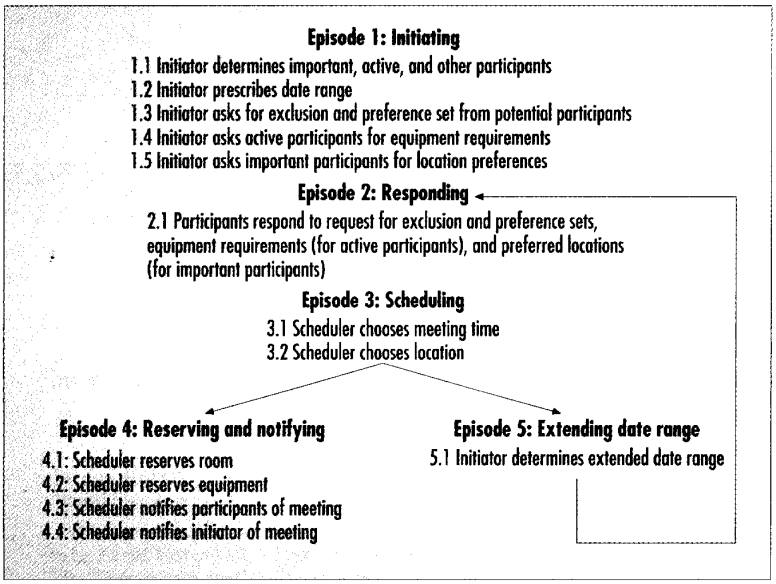


Figure 3. *Episodic structure of scenario 8 in Table 1 (Date conflict; initiator extends date range) and the action structure of each episode. Two of the five episodes contain the same actions as those in the scenario in Figure 2. Initiating contains the first five actions in Figure 2; Responding contains action 6.*

TABLE 3
ACTION TABLE FOR THE SCHEDULING EPISODE — NO CONFLICT, SEVERAL POSSIBLE MEETING TIMES

Agent	Action
Scheduler	Find meeting times and locations that are in preference set and not in exclusion set
Scheduler	Notify initiator of available times
Initiator	Select time and location

TABLE 4
ACTION TABLE FOR SCHEDULING EPISODE — NO CONFLICT ONE POSSIBLE MEETING TIME

Agent	Action
Scheduler	Find meeting times and locations that are in preference set and not in exclusion set
Scheduler	Notify initiator of scheduled meeting time and location time

TABLE 5
ACTION TABLE FOR SCHEDULING EPISODE — CONFLICT NO POSSIBLE MEETING TIME

Agent	Action
Scheduler	Find meeting times and locations that are in preference set and not in exclusion set
Scheduler	Notify initiator of conflict

Level of detail. Scenarios may be more or less detailed. For example, the episode Scheduling, during which the scheduler determines the time and location for the meeting, must cover three cases. Tables 3 through 5 show three scripts for those cases. Tables 3 and 4 are successful cases (no conflict): multiple meeting times and one meeting time, respectively. In Table 5, there is no feasible schedule, so the case is unsuccessful.

Related to the level of detail is the degree to which the terms used to describe a scenario represent instances. All the scenarios presented so far are generic because their agents are not agent instances, like specific names, but agent types, like participant and initiator. When agents of the same type interact, or when several agents of one type interact with one agent of another type, it is more useful to have agent instances to avoid confusion. Each agent instance must then be given a name and properties.

The same argument applies to data objects. The term "meeting" would hardly be satisfactory if six or seven meetings of the same type are being scheduled. Stakeholders should replace such generic terms with instances, like weekly progress meeting.

Consider the unsuccessful use case in which no meeting is feasible. A more specific case is Annie Out Of Town, in which an initiator, Esther, has scheduled a particular meeting that requires the attendance of Annie (active), Kenji (important), and Colin (ordinary). The meeting must be held next week, but Kenji and Colin can attend only on days when Annie is out of town. This is a much more concrete scenario than the scenarios considered so far.

However, introducing instances into scripts has the drawback of doubling their size and possibly introducing irrelevant details. Single actions in the generic scenarios translate into multiple action instances in the instantiated scenario. (We found 51 actions in the script for Annie Out Of Town, for example, as compared with 21 actions in the scheduling episode.) Compare the initiating episode of Figure 3 with the detailed script in Table 6, for example.

But size is not the only factor to consider in weighing the effects of using instances. Instances make scenarios more concrete, which may make requirements discussion easier and help resolve conflicts more quickly. We are currently investigating this possibility.

Requirements discussion. Figure 4 shows how a requirements discussion evolves. Here the requirements information being challenged is not a requirement, but a fragment of a scenario (Sc8.3) that explores the planning of a meeting when one of the potential participants does not respond to the meeting initiator (Esther) with his or her meeting-time preferences. This is the replanning phase of the Slow Responder scenario described earlier.

The scenario fragment is Esther's action of determining that the drop-dead date for scheduling the meeting should be Friday noon. (We introduced the concept of a drop-dead date to avoid the situation in which the scheduler has not received all the participants' preferences by the time the meeting should have occurred. At the drop-dead date, some decision must be made about scheduling the meeting.)

This scenario fragment is annotated with two notes: Q33, a question and A9, an answer to an unstated question (the answer is actually an assumption). Q33 asks when the drop date can be relative to the date range for meeting that Esther originally proposed. In other words, when exactly is the latest time that the decision about scheduling the meeting must be made? This question is resolved by answer A26. A9 answers the unstated question about when meetings can be

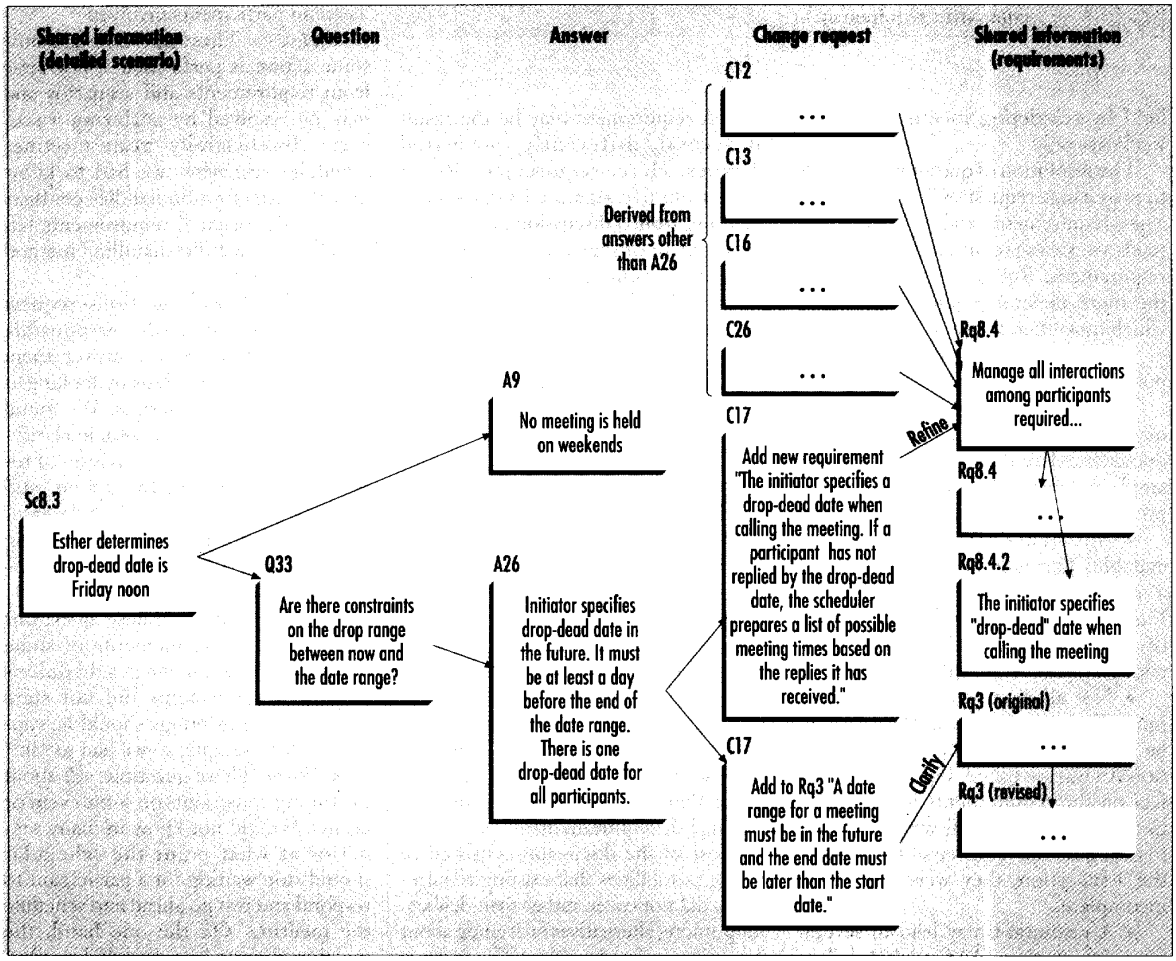


Figure 4. *Sample requirements discussion for the meeting scheduler. The information being challenged is a scenario fragment (Sc8.3), not a requirement. Sc stands for scenario, Rq for requirement, Q for question, A for answer, and C for change request.*

TABLE 6
DETAILED SCRIPT FOR INSTANTIATED SCENARIO ANNIE OUT OF TOWN

Agent	Action
Esther	Creates new meeting
Esther	Determines that Kenji is an important participant
Esther	Determines that Annie will be presenting
Esther	Determines that Colin is an ordinary participant
Esther	Types meeting description
Esther	Sets data range to be Monday - Friday next week (It's Wednesday p.m. now)
Esther	Determines drop-dead date is Friday noon
Scheduler	Sets timeout to be Fri 9am
Scheduler	Sends boilerplate message to Colin requesting constraints
Scheduler	Sends boilerplate message to Kenji requesting constraints and preferred location
Scheduler	Sends boilerplate message to Annie requesting constraints and equipment requirements

held by restricting meeting times to weekdays only.

The resolution of question Q33 leads to two change requests: C17 (which adds a new requirement), and C29 (which extends an existing requirement). The new requirement, Rq8.4.2, is subsumed by the more general requirement Rq8.4, which must therefore also be changed.

This example illustrates several points about the Inquiry Cycle model.

- You do not have to initiate requirements discussions by analyzing documented requirements. In this example, we analyzed a scenario, yet it still produced changes to the requirements.
- Shortcuts are useful and inevitable. The assumption A9 answered a question that was not worth stating. Furthermore both it and the resolution of another question, A26, were obvious enough to skip any explicit rationale.
- Not all discussion elements lead immediately to changes but they should be addressed eventually. The assumption A9 was recorded, but it had no effect on the requirements in this iteration of the cycle. Until we got around to changing the requirements to reflect the assumption, they were essentially incomplete.
- A resolution may lead to several nonlocal changes. Answer A26 led to changes to two separate requirements (Rq 8.4 and Rq3).
- A requirement may be the result of several, differently motivated changes. Here, requirement Rq8.4 stems from five separate changes, each resulting from a discussion about some requirement or scenario fragment.
- Requirements are themselves structured. The requirements in this example are numbered in sequence (Rq3 comes after Rq8) and hierarchically (Rq8.4 subsumes the more specific Rq8.4.2).

Effectiveness of scenarios. As the previous example supports, scenarios are at least as effective as the requirements document in prompting questions about requirements. About half the questions (55 percent) about the meeting scheduler were raised while analyzing or constructing scenarios. Recording questions definitely helps keep track of open issues. All the questions were answered, one of them directly by a scenario; the others by answers. Most answers (94 percent) explicitly answered questions, rather than being assumptions or facts attached to requirements.

Most of the discussion consisted of raising possibilities that existing requirements did not cover, rather than deliberating among alternatives or arguing about the rationale for decisions. The average question gave rise to only 1.3 answers. Only about one-third of the recorded answers represented alternatives that had been rejected.

Reasons were given for about half the answers (43 percent of the selected answers, 50 percent of the rejected ones).

Kinds of questions. Requirements discussions were triggered by several distinct types of questions.

- *What-is.* These questions request more information about a requirement (not a scenario) and are usually resolved by a definition. One of the meeting scheduler's original requirements was that it keep the participants involved in the scheduling process. We asked the what-is question, "What is meant by 'keeping participants involved'?"
- *How-to.* These questions ask how some action is performed. They arise from requirements and scenarios and may be resolved by analyzing a scenario. To effectively create meeting-scheduler scenarios, we had to know how the organization handles conflicts because the original requirements left the choice of conflict-handling method open.
- *Who.* These questions request confirmation about the responsible agent. Stakeholders can answer them through scenario analysis by looking at the agents listed in scripts. We found scenarios particularly helpful in clarifying policy issues and the division of responsibility between the system and users. In looking at meeting-scheduler scenarios, we found ourselves asking who determines if the type of participant is ordinary, active, or important?
- *What-kinds-of.* These questions request further refinements of some concepts. The meeting scheduler's original requirements did not state what kinds of meetings should be supported, for example, so we had to ask.
- *When.* These questions ask about the timing constraints on some event or events. We did not know in many scenarios at what point the scheduler should stop waiting for a participant to respond and just go ahead and schedule the meeting. On the one hand, the meeting cannot be scheduled until at least *some* potential participants have responded; on the other hand, schedul-

ing can certainly not be delayed until after the proposed meeting time. By asking this "when" question, we discovered that the original requirements had failed to account for the concept of a drop-dead date — the time the scheduler makes its best schedule on the information available. This concept was essential to a more detailed specification of the scheduling requirements. Stakeholders may disagree about how the system determines the drop-dead date or if it is fixed, but they cannot escape the conclusion that a drop-dead date is needed.

♦ *Relationship.* These questions ask how one requirement is related to another. The drop-dead date just mentioned has implications for the interpretation of other requirements. For example, are there constraints on the drop-dead date, given the current date and the initiator's proposed date range for the meeting? Obviously a fixed drop-dead date of seven days is going to be unworkable in a system that must call meetings at shorter notice.

Although most of the questions raised were about the term or idea being analyzed at the moment, there were also many instances in which analyzing one requirement or scenario suddenly prompted a question about something else.

It is not practical to untangle the train of thought in most such cases, and we do not try. The results of the train of thought, however, are very useful and should not be lost. We call these serendipitous discussion elements parenthetical insights.

Parenthetical insights can be recorded in the same way as more logical discussion elements. For example, if a stakeholder asks a question about drop-dead dates while inspecting a scenario that does not feature the setting or expiration of a drop-dead date, that question should still be attached to the current scenario because it is the current focus of inquiry. This method of attaching questions to the current focus is less burdensome than having to explicitly switch context. Of course, if a stakeholder does not act on the discussion quickly by revising the requirements to include the concept of a drop-dead date, another stakeholder may encounter the same need in another scenario and recommend an alternative, conflicting change.

Questions that lead to parenthetical insights are raised in two ways, both of which occurred equally often in the experiment when analyzing the requirements directly and when analyzing scenarios:

♦ *What-if questions.* It is especially informative to ask about cases in which an action could go wrong or its preconditions could be unsatisfied. Pursuing this type of question often leads to insights about apparently unrelated system features. An example is, what would happen if the latecomer responds with preferences after the meeting has already been scheduled and the preferences conflict with the schedule? This question can prompt the team to investigate the relative status of meeting participants when scheduling conflicts arise and if this has any bearing on whether or not a participant is important.

♦ *Follow-on questions.* These stem from other pending questions. An important category of follow-on question is where one question generalizes another. The answer to the new, broader question, may lead to changes in the requirements in places other than the one that led to the question. Consider the question, is an important participant's attendance vital? In other words, can a meeting go ahead without an important participant? This suggests the follow-on question: Is an active participant's attendance vital? You can generalize both these questions into a second follow-on question: What are the preconditions for holding a meeting?

GIVEN THEIR INSIDIOUS NATURE, YOU SHOULD FLAG ASSUMPTIONS AND CONSIDER ALTERNATIVES.

Assumptions. Stakeholders often make assumptions — that is, they answer a tacit question about the requirements without articulating the question. For example, an assumption about the meeting scheduler's communication medium could be construed as an answer to the question, what is the nature of the communication medium?

Stakeholders seldom justify their assumptions or consider alternatives. Assumptions, like answers to explicit questions, may be retracted. Given the insidious nature of unrecognized but false assumptions and their tentative nature, you should flag all assumptions carefully and make an extra effort to justify and authorize assumptions, consider alternatives, and reconsider them occasionally. Because A9 in Figure 4 annotates a requirement directly and does not answer an explicit question, you can clearly see that it records an assumption. Our prototype hypertext system provides standard queries for stakeholders to list unchallenged assumptions.

Assumptions seem commonest when discussing implementation constraints or the boundaries between the proposed system and its environment — especially in market-driven projects, where boundaries (particularly the degree of automation) are often fuzzier than in contractual projects. For example, the original meeting scheduler requirements did not state if the system is mediated by electronic mail or uses a shared calendar database. They also did not state if the scheduler should communicate with the participants or just display the schedule information to the initiator.

We found scenarios useful in challenging assumptions about the system boundaries and in helping us make a commitment to a range of options. In this case, we decided to make the scheduler itself an e-mail participant. An equally valid interpretation would have been to have a single user invoke the scheduler and have that user be responsible for sending messages and interpreting replies.

Stakeholders must draw the proposed system's boundary by surfacing assumptions such as this, because progress becomes impossible unless an assumption or decision is made. Without a real customer (for example, when end users or customer representatives are temporarily unavail-

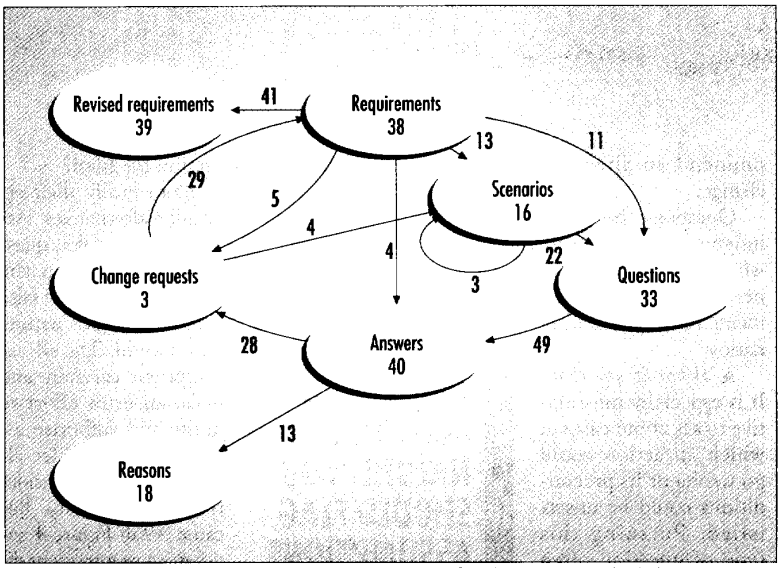


Figure 5. *Results of analyzing the requirements for the meeting scheduler. The ovals depict instances of a particular element of Inquiry Cycle model; the numbers on the arrows are the instances of that particular transition from one element to another. Some of the arrows represent shortcuts — they cut across the circular flow depicted in Figure 1. The numbers on these arrows indicate how often a particular shortcut occurred.*

able, or when the developers are designing a product for a market), such assumptions may lead to the questions being answered tentatively in several different ways. The stakeholders can then pursue the consequence of each plausible system boundary.

Requirements evolution. We found that there are three types of change requests: mutation, restriction, and editorial. A *mutation* request calls for a change or addition to the requirements themselves. Thus, when such a change is enacted, the system being described changes.

A *restriction* request, on the other hand, calls for a change to the requirements document, expressed as the addition of a clarification or definition. The system is further constrained and potential ambiguities are removed, but the original intent is unchanged.

An *editorial* request is a request to reword or rewrite some requirement. It is not intended to remove ambiguity, but to correct grammatical or spelling errors or to introduce consistent terminology, such as changing "potential attendee" to "potential participant."

In our experiment, about two-thirds of these changes (65 percent) were part of a full inquiry cycle consisting of questions, answers, reasons and a change. Of the shortcuts to the inquiry, about half the discussions (54 percent) omitted questions, the changes in these cases resulting from discussion about assumptions (which in turn clarified requirements) that were attached directly to the requirements document or scenario. The rest (56 percent) omitted answers, too. The changes in these cases arose directly from requirements analysis.

Also, more than half the changes to the meeting-scheduler requirements (58 percent) stemmed from analyzing scenarios rather than reviewing the requirements document itself. About a quarter of the questions (28 percent) that arose while analyzing the requirements document itself could not be answered except by constructing and analyzing scenarios.

Despite the number of changes, the number of requirements increased by only one, from 38 to 39. Most change requests applied to requirements, not scenarios (by a margin of 28 to three). This is to be expected: the purpose of the Inquiry Cycle is to improve requirements; the purpose of scenarios is to make them clearer.

ANALYSIS OF RESULTS

Figure 5 shows the number of times each type of model element and transition occurred for the meeting scheduler. The ovals show the number of instances of that element type; the arcs show the number of instances of that transition type. We assigned numbers to all paragraphs and subparagraphs of the requirements document to get 38 distinct requirements in the original version. Later, we revised the requirements to 39.

The figure shows that stakeholders raised 33 questions about 11 requirements and 22 scenario fragments. Of the 38 change requests, only 29 were acted on, resulting in 41 separate modifications to the original requirements. These modifications had the net effect of adding a requirement.

Usefulness of scenarios. There is little doubt that scenarios can be useful for elaborating requirements. In our experience, some questions about requirements are not easily answered except by resorting to scenario analysis. About half the improvements to a set of requirements came from analyzing scenarios, not from analyzing requirements documents themselves.

We have only begun to investigate different forms of scenario analysis and their effectiveness in clarifying and improving requirements. We are especially interested in comparing the value of general, thematic descriptions of scenarios with detailed, instantiated scenarios. Intuition suggests that increased detail is more appropriate once you know more about the system. Paradoxically, however, our preliminary work suggests that fully instantiated scenarios are equally useful early in requirements analysis. Perhaps the effort required to construct them forces stakeholders to surface and discuss assumptions that would otherwise be hidden for longer. A team of stakeholders may even go as far as role-playing a concrete scenario (as we did in the case of Annie Out Of Town).

Suitability of model. Our experience convinces us that the Inquiry Cycle vo-

cabulary is expressively adequate for the types of discussions held during requirements analysis. Intelligent tool support would require a more explicit and formal model of the domain (meeting scheduling) and a richer theory of speech acts and transformations. However, increased formality would defeat the object of using the Inquiry Cycle model as a foundation for directing and systematizing exploratory thought during the requirements phase.

Because our goal is to devise guidelines and principles that help analyze requirements for real systems — with a balance between regimenting an inherently informal and situated activity and providing no guidance at all — we believe that the Inquiry Cycle model's level of abstraction is an appropriate one.

The Inquiry Cycle model makes it easier to record discussion information in two ways:

♦ *It is artifact-based.* Unlike most idea-generation and issue-based methods, the Inquiry Cycle is directed at reviewing existing artifacts. Therefore, discussion always centers around the current requirements.

♦ *Shortcuts are always possible.* For example, a stakeholder may record an assumption about implementation constraints by annotating the appropriate requirement without having to raise a spurious question first.

Because discussions are attached to requirements or scenario components and may be consolidated into rationale objects,[1] we assume that the rationale for requirements should be fairly easy to retrieve. But because our emphasis has been on keeping track of ephemeral reasoning to improve requirements analysis, not to justify reasoning for subsequent implementers, we have not investigated how requirements discussions are used in later stages of development. We do believe, however, that stakeholders (especially maintenance programmers) might find it very useful to know the reasons for a requirement.

Level of effort. Extrapolating from our experiment, we estimate that a full elaboration of all 16 scenarios for the meeting scheduler, several iterations of the inquiry cycle, and a formal revision process would take approximately 500 to 1,000 person-hours.

If we had developed the resulting product in what Barry Boehm calls a semidetached development mode and delivered about 32,000 lines of code, his Basic Cocomo model estimates a project effort of 146 person-months.[5] Since a semidetached project typically spends seven percent of its total development effort in requirements and planning, about half of which is actually spent doing requirements analysis, the requirements effort would require 4.7 person-months, or about 750 person-hours.

THE INQUIRY MODEL MAY NOT REQUIRE MORE EFFORT, JUST A REFOCUSING OF EFFORT.

Another sanity check is to compare our projected effort with that required to perform a Joint Application Design session.[6] If the session lasts for one week and involves 10 stakeholders full-time, with two person-weeks of preparation and two of follow-up, the total effort would be 800 person-hours.

Obviously, these estimates are only very rough. However, that our numbers are in agreement with both other development modes shows that the Inquiry Cycle model may not require more effort, just refocused effort.

The Inquiry Cycle model has been a satisfactory framework for investigating requirements-analysis issues. Since this case study, we have been working to refine it and pursue larger scale applications. Topics we have selected for additional effort include

♦ *Scenario types.* We are investigating the representations and values of different types of scenarios. We are also investigating goal-based heuristics that suggest what scenarios to analyze and which of those to elaborate further. The relationship between goals and scenarios seems to be particularly direct and fruitful in exploring business-process reengineering. Our approach is to analyze different stakeholders' goal structures (obtained through interviews, reading, and observation), resolve structural conflicts among the structures, assign role responsibilities to the goals, and propose the introduction of information technology to support specific responsibilities.

Different types of scenarios occur in several parts of this strategy. We are currently analyzing the financial services organization of an academic institution to elaborate the strategy.

♦ *Transition to design.* Following work in the object-oriented analysis/object-oriented design communities, we are looking at the transition from scenario-based requirements analysis to object identification and responsibility-driven design.

♦ *Tool support.* Although our emphasis is on process, not tools, our experience shows that tool support is an important factor in the success of inquiry-based requirements analysis. We chose a commercial document processor for our case study over a prototype tool that we had developed ourselves.[1] That tool, which was based on Emacs, was not very usable. To follow an inquiry-based approach, we believe that the user must have access to an effortless annotation environment.

We have recently implemented Tuiqiao, a hypertext support tool for the Inquiry Cycle model, to provide a more transparent annotation environment.[7] Although the inquiry cycle is best described from a bird's eye view (as in Figure 4), in which the types of information and their relationships are viewed from above, our experience suggests that the view provided by a support tool should center on requirements documentation. Tuiqiao accordingly presents requirements discussion, rationale, and change requests as cascading annotations to the requirement or scenario. This means that while inspecting a requirement (or fragment of a scenario), a user can easily call up the

open questions or assumptions that have been posted about it, any change requests referring to it, and its rationale (what earlier decisions led to it).

Tuiqiao supports the use of link reversal and querying to help users find requirements information. To support rationale management, it lets users find reasons for current rationale by navigating backward through a version history of requirements, scenario-element, and discussion-element nodes. It also lets users plan and monitor the analysis process by checking for invalidly linked nodes that may represent unchallenged assumptions or unimplemented changes.

♦ *Case study research.* Although the meeting scheduler is a rich and realistic problem, the "stakeholders" really don't have a stake in the solution to the degree that real stakeholders in a commercial or industrial system do. To broaden our practical experience with the model, we plan to apply it and prototype tools to real system development projects in the telephone industry starting immediately. We plan to apply Tuiqiao to the requirements-analysis phases of several projects including LAN/WAN design, telecommunications system development, and the development of CASE environments for telecommunication systems. In these case studies, we will seek to answer how scenarios are used to challenge the requirements of more detailed systems, and how the model's components are used during an extended project involving many people. ◆

Colin Potts is an associate professor in the College of Computing at Georgia Institute of Technology, where he is a member of the Software Research Center and the Graphics, Visualization and Usability Center. His research interests include the definition and design of interactive systems and computer-supported cooperative work.

Potts received a PhD in cognitive psychology from Sheffield University. He is a member of the IEEE Computer Society and ACM.

Annie I. Antón is a PhD candidate in computer science at the Georgia Institute of Technology, where she is a member of the Software Research Center and the Center for Information Management Research. Her research interests include the definition of information-system requirements, computer-supported cooperative work, and software processes.

Antón received an MS in computer science from the Georgia Institute of Technology. She is a member of the IEEE Computer Society and ACM.

Kenji Takahasi is a senior researcher at the Software Laboratories of Nippon Telegraph and Telepone Corp. His reasearch interests include requirements engineering and multimedia support for software development.

Takahashi received a BS and an MS in computer science from Toyko Institute of Technology.

Address questions about this article to Potts at Georgia Institute of Technology, College of Computing, Atlanta, GA 30332-0280; potts@cc.gatech.edu.

REFERENCES

1. C. Potts and K. Takahashi, "An Active Hypertext Model for System Requirements," *Proc. Workshop Software Specification and Design*, IEEE CS Press, Los Alamitos, Calif., 1993, to appear.
2. M. Lubars, C. Potts, and C. Richter, "Developing Initial OOA Models," *Proc. Int'l Conf. Software Eng.*, IEEE CS Press, Los Alamitos, Calif., 1993.
3. W. Kunz and H. Rittel, "Issues as Elements of Information Systems," Working Paper 131, Inst. Urban and Regional Development., Univ. Calif. at Berkeley, Berkeley, Calif., 1970.
4. N. Goldman, "Three Dimensions of Design Development" *Proc. AAAI*, Amer. Assoc. for Artificial Intelligence, 1983.
5. B. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, N.J., 1981.
6. J. Wood and D. Silver, *Joint Application Design*, John Wiley & Sons, New York, 1989.
7. K. Takahaski and C. Potts, "Tuiqiao: A Hypertext Tool for Requirements Analysis" Tech. Report GIT-CC-94107, Georgia Inst. of Technology, Atlanta, 1994.