ISR Institute for Software Research
University of California, Irvine

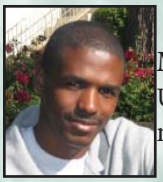
# The Importance of Clarity in Usable Requirements Specification Formats


**Thomas A. Alspaugh**
University of California, Irvine
alspaugh@ics.uci.edu


**Kristina Winbladh**
University of California, Irvine
awinblad@ics.uci.edu


**Susan Elliott Sim**
University of California, Irvine
ses@ics.uci.edu


**Hadar Ziv**
University of California, Irvine
ziv@ics.uci.edu


**Mamadou Diallo**
University of California, Irvine
mdiallo@uci.edu


**Debra J. Richardson**
University of California, Irvine
djr@ics.uci.edu

September 2006

**ISR Technical Report # UCI-ISR-06-14**

**Institute for Software Research**
ICS2 110
University of California, Irvine
Irvine, CA 92697-3455
**www.isr.uci.edu**

# The Importance of Clarity in Usable Requirements Specification Formats

Thomas A. Alspaugh Susan Elliott Sim Kristina Winbladh Mamadou H. Diallo
Leila Naslavsky Hadar Ziv Debra J. Richardson

Institute for Software Research
Department of Informatics
University of California, Irvine
{alspaugh,ses,awinblad,mdiallo,lnaslavs,ziv,djr}@ics.uci.edu

## Abstract

*Clarity is underappreciated as a requirements specification quality attribute. We studied the clarity of requirements forms, operationalized as ease of problem detection, least obstructive to understanding, and understandability by stakeholders. A set of use cases for an industrial system was translated into sequence diagrams and ScenarioML; problems identified during each translation were noted, and system stakeholders were interviewed and given a questionnaire on all three forms. The data showed that ScenarioML best supported requirements clarity, then sequence diagrams but only for stakeholders experienced with them, and finally use cases as the least clear form. Use cases were preferred for non-technical stakeholders to write; sequence diagrams were most effective for details of individual events and for showing interaction with architectural components; with ScenarioML preferred in all other situations.*

# The Importance of Clarity in Usable Requirements Specification Formats

Thomas A. Alspaugh Susan Elliott Sim Kristina Winbladh Mamadou H. Diallo
Leila Naslavsky Hadar Ziv Debra J. Richardson

Institute for Software Research
Department of Informatics
University of California, Irvine
{alspaugh,ses,awinblad,mdiallo,lnaslavs,ziv,djr}@ics.uci.edu

## Abstract

*Clarity is underappreciated as a requirements specification quality attribute. We studied the clarity of requirements forms, operationalized as ease of problem detection, least obstructive to understanding, and understandability by stakeholders. A set of use cases for an industrial system was translated into sequence diagrams and ScenarioML; problems identified during each translation were noted, and system stakeholders were interviewed and given a questionnaire on all three forms. The data showed that ScenarioML best supported requirements clarity, then sequence diagrams but only for stakeholders experienced with them, and finally use cases as the least clear form. Use cases were preferred for non-technical stakeholders to write; sequence diagrams were most effective for details of individual events and for showing interaction with architectural components; with ScenarioML preferred in all other situations.*

*That I require a clearness: and with him to leave no rubs nor botches in the work* – The Tragedy of Macbeth, Act 3 Scene 1, by William Shakespeare

## 1. Introduction

High quality requirements are critical to the success of a software project, because all other development activities depend on them [5, 6, 8]. Consequently, it is of utmost importance that requirements are clear and usable, so that stakeholder needs are readily found and understood, and mistakes and misunderstandings are avoided. An important limiting factor on the clarity of requirements documents is the clarity of requirements specification formats. While properties such as expressibility, analyzability, and completeness are generally accepted as desirable, clarity has been little studied in the context of requirements documents or requirements specification formats.

In this paper, we report on an empirical study that evaluates three requirements specification formats, use cases, sequence diagrams, and our own ScenarioML, for their clarity. Our operational definition of clarity, gleaned from multiple dictionaries and thesauri, is as follows: (1) readily seen, perceived, or understood, (2) distinctness of vision, sound, expression, comprehension, and (3) freedom from anything obstructive. From this definition, we derive the following three research questions: (1) Which format more readily permits the detection of problems in requirements? (2) Which format has a structure that least obstructs understandability? and (3) Which format do stakeholders find more clear? The first phase of our study was concerned with evaluating the capabilities of the formats, the relative frequency of problems, and ease of use. The second phase of our study involved presenting the different representations to stakeholders to ask them about their preferences and also to test them on how well they could read and understand the different formats.

We began by creating requirements documents for a system in three different formats. The system was Mirth, an open source middleware system for integration of healthcare applications, developed by WebReach. The initial requirements elicitation was performed using use cases and these were used as a baseline during the first phase of the study. We then transformed the use cases independently into two other formats: sequence diagrams, and ScenarioML. We performed the transformation step in order to set up a fair comparison for the two formats, with a common starting point that serves as a single source of information. This translation was performed by two graduate students and as they worked, they recorded the problems, issues, and unresolved questions in the use cases that each translation uncovered. Finally, we presented these documents to three Mirth stakeholders. We interviewed them to understand their overall experience with these formats and assessed their ability to answer questions about requirements presented in each of the formats.

The data from both phases of the study showed that the ScenarioML format was the most clear, followed by use cases, followed by sequence diagrams. In general, the students were able to detect more problems, such as inconsistency, when transforming use cases into ScenarioML than when transforming them into sequence diagrams. While it was easier to detect high-level problems with ScenarioML, it was easier to detect problems with low-level details in sequence diagrams. Stakeholders also preferred ScenarioML over use cases, and over sequence diagrams. They found ScenarioML more clear and would choose to use this format, except in specific situations, such as showing communication between different levels of an architecture.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 introduces the three formats used in our study. Section 4 presents the method used in our empirical study and Section 5 presents the results. The discussion and analysis of the results is presented in Section 6 followed by lessons learned in section 7. The paper ends with conclusions and future work in section 8.

## 2. Related Work

Many textbooks and papers discuss desirable quality attributes of requirements documents and formats. Most notably, IEEE Standard 830 for Software Requirements Specification (SRS) states that a "good" SRS should be correct, unambiguous, complete, consistent, ranked for importance, verifiable, modifiable, and traceable [1]. Interestingly, the IEEE Standard does not include clarity as desirable characteristic of a good SRS. However, a standard textbook on software engineering gives lack of clarity as one of the top problems that arise when requirements are written in natural language sentences [10, page 127]. There has been little investigation of clarity as a quality attribute of requirements. We know of only one contribution that focuses on clarity, the CLEAR (Cognitive Linguistic Elicitation and Representation) method for writing requirements and a case study where CLEAR was applied to the requirements for a medical device [11]. While CLEAR focuses specifically on the language used in requirements, we are examining the importance of clarity in the specification formats.

In recent years there has been a growing interest in evaluating the various research contributions in requirements engineering, as is evident is the series of CERE (Comparative Evaluation in Requirements Engineering) workshops, now in its fourth year. This field is in its infancy in requirements engineering, and empirical methods are still being established. Our study will use both a comparative approach as well as an industrial subject systems and professional software engineers.

Our paper follows the GQM (Goal Quality Metrics) approach to measurement and evaluation as described by Basili *et al.* [3]. In GQM, an empirical study or comparative evaluation must first specify the goals for itself, then it must trace those goals to the data that identifies those goals operationally, and finally provide a framework for interpreting the data with respect to stated goals. Thus, GQM employs a measurement model with three levels, namely a conceptual level (goals), an operational level (questions), and a quantitative level (metrics).

## 3. Requirements Specification Formats

### 3.1. Use Cases

A use case is a description of a sequence of interactions between the system and one or more external actors that results in an outcome that provide value to at least one actor. According to Alistair Cockburn [7], a use case "describes the system's behaviors and interactions under various conditions as a response to a request on behalf of one of the stakeholders – the primary actor – showing how the primary actor's goal gets delivered or fails."

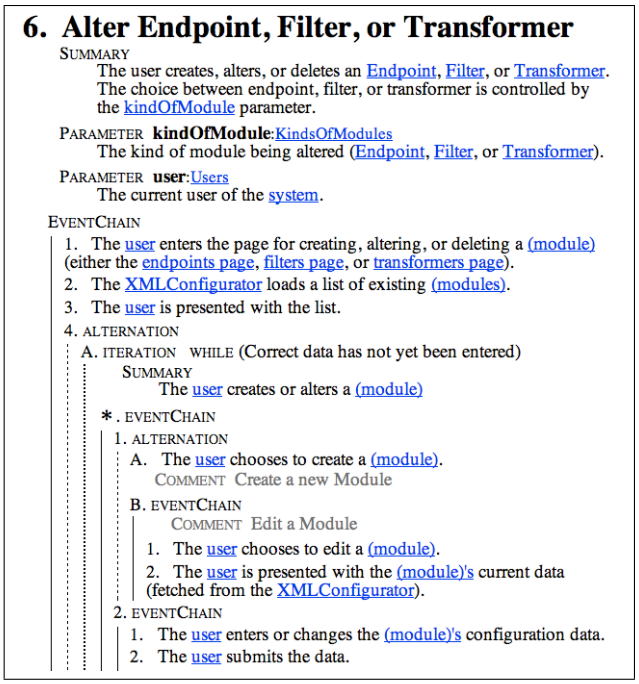
Use cases are fundamentally a structured text form, typically written using a use case template. The use-case author fills in text fields such as use case name, goal, brief description, pre- and post-conditions, and normal, alternative, and exceptional flows. Thus, a use case typically contains multiple scenarios, each representing a specialization of the actors stated goal, or alternative paths by which the actor could reach that goal.

### 3.2. Sequence Diagrams

Sequence Diagrams are dynamic UML diagrams that document, for each use case, the sequence of object interaction that take place. They are typically used during analysis and design activities but, much like use cases, have been used at many different levels of detail and abstraction, even as replacement for and in lieu of use cases. They illustrate the interplay between multiple actors and the system during execution of a use case, by showing the interaction and message passing between those actors and software objects in the system. The objects involved in the interaction are arranged horizontally, while time progresses vertically. Therefore, the messages or operations can be followed in sequence by reading the diagram from top to bottom.

### 3.3. ScenarioML

ScenarioML is a language for scenarios, designed to be read and written by people, while also accomodating machine processing. ScenarioML uses a combination of *recursively-defined events*, *ontologies*, *references*, and *scenario parameters* to make scenarios that are more clear,



**6. Alter Endpoint, Filter, or Transformer**

SUMMARY
The user creates, alters, or deletes an Endpoint, Filter, or Transformer. The choice between endpoint, filter, or transformer is controlled by the kindOfModule parameter.

PARAMETER **kindOfModule**:KindsOfModules
The kind of module being altered (Endpoint, Filter, or Transformer).

PARAMETER **user**:Users
The current user of the system.

EVENTCHAIN
1. The user enters the page for creating, altering, or deleting a (module) (either the endpoints page, filters page, or transformers page).
2. The XMLConfigurator loads a list of existing (modules).
3. The user is presented with the list.
4. ALTERNATION
   A. ITERATION WHILE (Correct data has not yet been entered)
      SUMMARY
      The user creates or alters a (module)
      *. EVENTCHAIN
      1. ALTERNATION
         A. The user chooses to create a (module).
            COMMENT Create a new Module
         B. EVENTCHAIN
            COMMENT Edit a Module
            1. The user chooses to edit a (module).
            2. The user is presented with the (module)'s current data (fetched from the XMLConfigurator).
      2. EVENTCHAIN
         1. The user enters or changes the (module)'s configuration data.
         2. The user submits the data.

**Figure 1. Scenario in human-oriented HTML form**

```
<scenario name="CreateModifyModule">
 <title>Alter Endpoint, Filter, or Transformer</title>
 <summary>The user creates, alters, or deletes an
  <ref to="#Endpoints" which="any"><text>Endpoint</text></ref>,
  <ref to="#Filters" which="any"><text>Filter</text></ref>, or
  <ref to="#Transformers" which="any"><text>Transformer</text><
  The choice between endpoint, filter, or transformer
  is controlled by the <ref to="#CreateModifyModule.kindOfModul
  </summary>
 <parameter name="kindOfModule"><type ref="#KindsOfModules",
  <text>The kind of module being altered
   (<ref to="#Endpoint"><text>Endpoint</text></ref>,
  <ref to="#Filter"><text>Filter</text></ref>, or
  <ref to="#Transformer"><text>Transformer</text></ref>).</text>
 <parameter name="user"><type ref="#Users"/>
  <text>The current user of the
   <ref to="#system"><text>system</text></ref>.</text></parameter

 <eventChain>
  <simpleEvent>
   <text>The <ref to="#CreateModifyModule.user"><text>user</te
    enters the page for creating, altering, or deleting a
    <ref to="#CreateModifyModule.kindOfModule"><text>(modul
    (either the
    <ref to="#endpoints_page"/>,
    <ref to="#filters_page"/>,
    or
    <ref to="#transformers_page"/>).</text>
```

**Figure 2. ScenarioML scenario (excerpt)**

more useful, and more effective. The basic element of a ScenarioML scenario is an *simple event* that describes in words one thing happening in the world. A *compound event*, groups several events to describe several things happening in the world in a particular sequence (or other more complex temporal relation). Allen's interval algebra relations express the temporal relationships among the subevents. An *event schema* compactly expresses a group of possible event combinations, for example as an iteration representing any of several chains of an event repeated some number of times, and an alternation representing any one of a group of alternative events. Finally, an *episode* uses an entire scenario as one event of another scenario. Figure 1 shows a ScenarioML scenario incorporating several of these kinds of events.

A ScenarioML *ontology* defines technical or specialized terms of concepts, a set of types of entities in the world, named instances of such types, and relationships among types. Links from words and phrases in the scenario to definitions in an ontology adds greater clarity to the text. The ScenarioML language currently defines its own ontology definition constructs rather than using an established ontology language because we are still investigating the extent to which automatic processing of scenarios can make use of ontological information.

ScenarioML also supports references to new entities created or identified during the course of a scenario's events. This allows a scenario author to be clear about entities, and is particularly useful for scenarios in which two of the same kind of entity are discussed. An example in point from the current study is the scenario "Add/Edit/Delete Users", in which the actor (who is a user) can add a new user, edit a user (including himself), and delete a user (but not himself). There are a number of references to a 'user' in this scenario, but not all to the same user. ScenarioML references allow this to be done unambiguously.

ScenarioML provides *scenario parameters* for adapting a scenario to different contexts. Each episode that reuses a scenario binds its parameters to arguments appropriate for that context and use. The type of each parameter, i.e. the set of its possible arguments, is specified as a type defined in an ontology.

ScenarioML is defined in an XML representation of which an example is seen in Figure 2. The definitions of the ScenarioML constructs allow software tools to read and operate on them, performing tasks such as checking that references are grounded in definitions and transforming a scenario with alternations into one with all choices made (both implemented at this writing); inlining an episode by replacing it with the corresponding scenario events with arguments substituted for parameters, or the reverse process of abstracting a new scenario from a compound event given a desired list of scenario parameters; unrolling a fixed iteration, or rolling up a chain of repetitions of an event; merging two or more scenarios using alternations for their differences; and other scenario refactorings and specializations. Automated transformations such as these help scenario au-

thors and readers verify their understanding of a scenario and edit scenarios more efficiently and with confidence that the intended meaning will be expressed.

ScenarioML has been the basis of work with autonomous animated social agents, [4], and in a novel visualization of software scenarios as social interactions between autonomous animated agents generated automatically from the scenario text [2]. The principles behind ScenarioML have been applied in an approach for goal-driven specification-based testing, in which the lower-level goals for a system are organized into plans for achieving higher-level goals and tested against them[12].

# 4. Method

Our goal was to investigate the clarity of three requirements formats and the documents written using these formats. Using a GQM approach, we began with a the definition of clarity given in the introduction and derived three research questions.

1. Which requirements format most readily permits the detection of problems?
2. Which requirements format has a structure that least obstructs understanding?
3. Which requirements format do stakeholders find more clear?

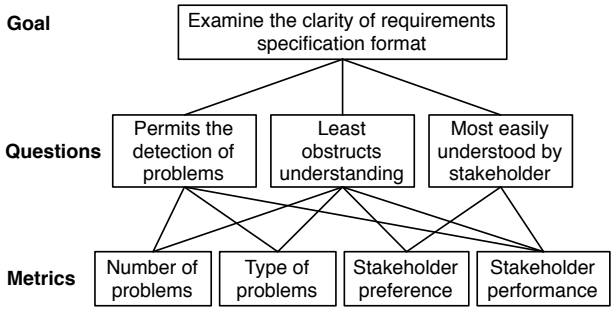



**Figure 3. GQM diagram**

To answer these questions, we conducted a two-part empirical study. In this study, we created and studied the requirements for a single system, Mirth, expressed in three different formats, use cases, sequence diagrams, and ScenarioML. In the first part, we examined the clarity of the formats when creating requirements documents. The metrics that we collected were the effort required to produce the documents, and the number and types of problems encountered when creating the documents. To this end, the participants kept minute-by-minute logs while they worked. In the second part, we examined the clarity of the formats when being read by stakeholders. The metrics that we collected were their subjective statements about their experiences with the documents and their objective performance in answering questions that tested their understanding of the documents presented in different formats. See Figure 3 for relationships between goal, questions, and metrics.

In the remainder of this section, we will describe in detail the procedure that we used in our empirical study.

## 4.1. Subject System

The study uses requirements for the Mirth system from WebReach. The use cases in our study were developed for WebReach as part of a student project. Mirth is an open source cross-platform HL7 interface engine that enables bi-directional transfer between systems and applications over multiple transports. HL7 is the primary software language of Health Information Systems. Mirth has a channel-based architecture and allows messages to be filtered, transformed, and routed based on user-defined rules.

## 4.2. Requirements Creation

In this part of the study, we investigated the clarity of the requirements format during creation of documents. In order to eliminate differences in the elicitation of requirements, we used a set of use cases as a starting point and translated them into two other formats, sequence diagrams and ScenarioML. By limiting our study to a translation activity, we were able to focus on the effort of writing a requirements document in a particular format, thereby enabling a fair comparison of the creation effort. Also, by restricting the initial information source, the translation would reveal problems and the clarity of the formats.

We chose sequence diagrams, because they are well known and widely used. While the other two formats are textual, we deliberately included a graphical format to draw out their relative merits. We chose to use ScenarioML, because we wanted to conduct a formative evaluation and it is similar in structure to use cases, but with additional features to support analysis. In this sense, ScenarioML is more complex than use cases, but produces documents that are more straightforward, because event chains are linear.

### 4.2.1 Procedure

In this part of the study, we translated requirements from a set of use cases to sequence diagrams and to ScenarioML. We collected data on both the process and the outputs of the translation. The logic of using a transformation step is that we could have a pair of comparable operations. Figure 4 depicts the procedure that we used to conduct the transformation and analysis.

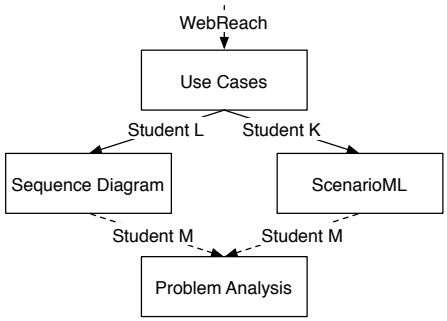



**Figure 4. Transformation & analysis process**

Student L translated the use cases to sequence diagrams. A second student, K, translated the use cases to ScenarioML. They both kept minute-by-minute logs while they worked that contain time-stamped problems discovered by the students and general descriptions of how they accomplished the translation. A third student, M, analyzed the logs and the artifacts produced to identify the number and types of problems encountered. He identifed the types of problems by inducing categories from the data, as is done in a Grounded Theory approach [9].

#### 4.2.2 Participants

The two students that participated in the translation task were selected because they were familiar, but not expert, with both the source and target specification formats, so that their levels of expertise would not be a factor in comparing their results. Each had a small amount of high-level familiarity with Mirth but had not read the use cases. Student M had experience in the analyzing process and product data, as well as conducting comparative evaluations.

### 4.3 Stakeholder Experience

In the second part of the study, we presented the three formats for evaluation by three stakeholders. We were able to use all three formats in this part of the empirical study, because the stakeholders were only presented with the final documents. They were asked about their impressions of the formats in an interview and then tested on their ability to understand them.

#### 4.3.1 Procedure

Each interview was conducted individually and lasted for about one hour. We began by asking them about their impressions and experiences with the formats. We then gave each participant a small test to assess the clarity of the three formats. Each participant was given three different sections of the requirements, presented in one of three formats, and asked questions to probe their understanding. We asked questions to see if they fully grasped complex interactions, understood potentially confusing material, and detected errors. The three problems were carefully selected, because each contained a potential problem in clarity.

**Problem 1: Add/Edit/Clone/Delete a Channel** This use case was complex and involved inter-relations between the main success scenario and alternate scenarios. A participant who completely understood these requirements would be able to paraphrase the steps and identify missing information.

**Problem 2: Deleting a User** This requirement was included in the study, because it contains an inconsistency. There are only steps for adding a user and an alternative scenario forbidding a user from deleting himself. Participants were tested on their understanding when asked to describe the sequence of steps necessary to complete this task.

**Problem 3: Change Channel Status** This requirement was used in the test, because the steps on how to disable a channel are missing. In addition, the alternate scenario does not share any steps with the main scenario. Again, users were asked to identify steps required to perform a task.

Given that we had three formats, three test problems, and only three participants, we used a Latin square design. Each participant saw each format and each problem but in different combinations, so that we could perform between- and within- participant comparisons of performance. Figure 5 depicts the order of presentation to each participant of the different problems and three formats. The exception is Participant A who mistakenly looked at a sequence diagram instead of the ScenarioML document for one of the problems. The order of presentation of problems was fixed, although the order of the formats was randomized.

| | Problem 1 | Problem 2 | Problem 3 |
|---|---|---|---|
| Participant A | Use Case | Sequence Diagram | Sequence Diagram |
| Participant B | ScenarioML | Use Case | Sequence Diagram |
| Participant C | Sequence Diagram | ScenarioML | Use Case |

**Figure 5. Presentation of formats**

The interviews were transcribed verbatim, without imposing sentence structure or clarification. We tagged and categorized utterances in the transcripts. During analysis, we identified statements that pertained to the clarity of each of the formats. We were careful to consider both positive and negative statements when summarizing the interviews and selecting examples.

#### 4.3.2 Participants

Three stakeholders at WebReach were participants in this study. The first stakeholder has over 20 years of experi-

ence as an enterprise software architect. He has designed and implemented solutions for clients in a wide variety of industries and technologies. The second stakeholder has 20 years of software engineering and management experience. His primary focus is business development and key-client account management. The third stakeholder is software engineering and helped lead the architecture and development of the Mirth Project. He is also currently a PhD student.

### 4.4. Validity

In this subsection, we discuss the internal and external validity of the study, as well as threats to validity.

**Internal Validity.** Internal validity is the soundness of the relationships within a study. In the requirements translation portion of the study, we linked the clarity of a format with the clarity of a document produced using that format. These in turn, were measured indirectly in terms of the problems that the format could reveal. A requirements specification format demands inclusion of certain kinds of information, and different formats have different demands. Therefore, translating requirements from one format to another will reveal shortcomings of the source document and the source format.

**External Validity.** External validity is the degree to which the results from the study can be generalized. In this study, we used only one subject system, three students, and three stakeholders. This is a small sample size on all counts. However, we are not trying to create statistically significant results that can be generalized to a population of subject systems, students, nor stakeholders. Rather, we are seeking to build theories and deepen understanding of requirements specification formats. In the requirements creation part of the study, we seek to reason analytically about properties and qualities of the formats. These properties will hold across subject systems and documents. In terms of stakeholders, the data are not as strong, so we use these only as corroborating evidence. However, we feel this evidence is valid, because these are actual stakeholders.

If the four metrics converge, the findings will be persuasive, because the weight of their support will triangulate a phenomenon, the importance of requirements clarity. Because the three kinds of data were produced in different ways and show different viewpoints on the issues, it is unlikely that all three will converge due to chance.

**Threats to Validity.** The students who participated in this study are authors on this paper and involved in some way in the design of ScenarioML. This may result in a bias in favor of ScenarioML. We attempted to minimize this threat by having the students work independently and having different students perform different parts of the study. In addition, we consulted stakeholders, rather than students, regarding their preferences.

## 5. Results

In this section, we present our findings regarding our four metrics. These metrics will be used to answer the questions from the GQM method in the next section.

### 5.1. Number and Type of Problems

Following the translation exercise, Student M analyzed the logs. A catalog of the problems found was created. These problems were then grouped according to similarities. Based on the classification, a taxonomy was created so that each problem encountered fit into only one category. The result was a comparison table (see Figure 6) between the problems found during the two translation activities. This table was verified by Students K and L to ensure that the taxonomy was accurate.

The data shows that ScenarioML revealed more problems in the original use cases than the sequence diagrams. However, sequence diagrams were better at showing problems with the details of events. In contrast, ScenarioML more effectively show problems at a high level of detail and interstitial problems, i.e., problems between use cases or between different parts of use cases.

Category 1 in the table in Figure 6 addresses inconsistencies between the use case diagram and the textual use cases. Some use cases seemed to be represented in both, but under different names such as “CreateModifyModule” and “CreateModifyViewModule”. Others were visible in the diagram and simply missing in the textual representation. Another problem that surfaced during the translation was that use cases do not clearly express iterations, this type of problem is shown in category 2. Category 3 shows structural problems, where clarity could have been gained by combining similar use cases. UC-03, UC-04, and UC-05, for example, perform exactly the same steps with different actors. Another problem was the confusing numbering of events that resulted in ambiguous alternatives and exception scenarios. Category 4 shows such problems, which were found in three of the ten use cases. Category 5 shows problems of missing alternatives. The option of deleting a user, for example, was listed as an option but there were no events that described this process. Category 6 shows mistakes, such as mixing up the order of events in the use cases. Category 7 addresses the problem of incompleteness, such as referencing non existing use cases, while category 8 deals with inconsistencies such using different names for the same actors. The last row combines problems in individual events, such as where data can be stored.

| Problems | # found in SDs | # found in scenarios |
|---|---|---|
| 1. Inconsistency between UC diagram and textual UC | 0 | 4 |
| 2. Iteration was not specified | 0 | 2 |
| 3. Redundancy in the UCs | 0 | 1 |
| 4. UC numbering scheme does not show where alternative flows start/stop | 3 | 4 |
| 5. All possible alternatives were not specified | 0 | 2 |
| 6. Mistakes in the UC | 1 | 4 |
| 7. Incomplete UCs | 3 | 3 |
| 8. Two names referring to the same term | 5 | 5 |
| 9. Problems with details of events | 6 | 3 |

**Figure 6. Problems found during translation**

## 5.2. Stakeholder Preference

Each participant was interviewed individually and asked about their preferences regarding the three formats. In general, the participants preferred ScenarioML over the other two formats. While they pointed out a few things that were unclear and not intuitive in the scenarios, such as the star that represents a number of iterations, they did not perceive these as off-putting or thought that other stakeholders would. However, they also thought that use cases might be superior for initial requirements gathering and for users to write themselves. Sequence diagrams were disliked by the two participants who were not already familiar with the notation. Sequence diagrams were seen as useful or advantageous in more restricted contexts and audiences. The participants liked ScenarioML although, they had same or less experience with this notation. The participants liked use cases but found ScenarioML scenarios to be an improvement in a number of ways. Below is a summary of the results per format.

**Use Cases.** The participants perceived use cases as "fairly easy to understand". In particular they found it easy to understand the steps described in the use cases, the summaries of what the use cases do, and the main and alternative scenarios that express alternative flows. Things that the users found difficult to understand included the numbering scheme for alternative scenarios, and the terms used in events. The participants liked that the use cases were easy to read and that the notation was familiar. Some things the participants disliked were the lack of detail, the numbering scheme, the inconsistency in terminology, and the lack of hyperlinks to supporting documents.

**Sequence Diagrams.** Two of the participants perceived the notation confusing, particularly confusing was how to follow events. The third participant found the notation "pretty easy", and understood the calls between components and the separation of layers. The subject that understood the sequence diagrams liked them because of their graphical nature. Some things that the other two participants did not like were the lack of readability, the visual overflow, and the lack of hyperlinks to supporting documents.

**ScenarioML.** The participants found the notation easy to understand. The participants understood the concepts of parameters, ontology definitions, event chains, alternations, references, and instances. There were some things that were not apparent to the participants at first, such as the star notation for iterations. The participants liked that the notation was simple and textual with indentations, the declarative statements with parameters, that there were better and more consistent definitions of terms, that it was more rigid, and had hyperlinks to supporting documents and terminology.

The overall observation is that the participants think ScenarioML specifies scenarios with greater clarity, except in showing interactions with various layers of a system with a layered architecture.

## 5.3. Stakeholder Performance

This section will shows the participants' performance on the questionnaire (see Section 4 for the particular questions).

**Use Cases.** Overall the participants performed poorly when trying to identify problems in the use cases. In Problem 1, the participant with the use case managed to find a problem regarding how the alternate extension fit into the main use case. The participant failed to see any other problems, such as the event sequence for the "Clone" option not making sense. The participant still thought he understood what the use case was "trying" to do. The participant did not think that this notation was particularly useful for finding problems.

In Problem 2, the participant with the use case listed a sequence of steps that did not make sense, since the option was missing in the use case. The participant concluded that the use case was "kind of ugly". The participant had not noticed this problem before, and did not think that the notation did anything to point out this problem.

In Problem 3, the participant with the use case thought it was difficult to determine the starting point of the two scenarios (main and alternate), because they were different. The participant did not notice this problem when looking through the use cases during the interview.

**Sequence Diagrams.** In Problem 1, the participant with the sequence diagram understood the notation pretty well, and was positive toward the notation although no problems were identified.

In Problem 2, the participant with the sequence diagram listed that the user can add, modify, or delete a user, when in

fact only the add option is supported. The participant realized that the option to modify a user was not well articulated before starting the questionnaire, but did not think that the notation is helpful for detecting these kinds of problems.

In Problem 3, there were two participants with sequence diagrams. Neither of the participants could list the steps to disable a channel. One participant noticed this problem before the questionnaire, and the other participant thought the notion of a step was difficult in general. One participant thought that "This specification is really not a good way to specify this problem and, as a result, it makes relatively simple problem[s] difficult to spot." and the other participant found sequence diagrams in general to be "incomprehensible".

**ScenarioML.** In Problem 1, the participant with the ScenarioML scenario understood the scenario well.

In Problem 2, the participant with the ScenarioML scenario understood the scenario well, and pointed out that the nested `EventChain` highlights the fact that something is occurring in an otherwise flat sequence of events and draws attention to it, and that "This causes me to further inspect these nested events more closely as they often seem to be problematic areas."

In Problem 3, the participant with the ScenarioML scenario mistakenly answered the question using a sequence diagram instead of a scenario, which impairs the Latin square. The participant confirmed this afterwards.

It is interesting to note that the subjects exhibited low performance to find problems in the use cases and sequence diagrams despite their familiarity with the subject system. Another interesting observation is the low performance vs. the high preference expressed in the interview. The interview showed that all three participants felt positive toward use cases and thought they were clear. Participants showed more negativity toward use cases, when asked to use them. It also shows that they were difficult to use because they were unclear, contradicting the initial reaction from the stakeholders.

## 6. Discussion

### 6.1. Which requirements format most readily permits the detection of problems?

In our study, the translation process is used to compare the ease of problem detection in each of our target formats. This comparison is based on the reasoning that information required by a target format, but missing from the source format is indicative of a shortcoming in the source format. We decide the strengths and weaknesses of a format by comparing numbers and types of problems identified in the translation to that format. A drawback of this approach is it does not permit a side-by-side comparison of use cases to either sequence diagrams or ScenarioML scenarios, as use cases were the source format in both translations. However, data from the interviews and comprehension test show that stakeholders detected few of the problems present in either the use cases or sequence diagrams they examined, which is consistent with the inference that use cases are not markedly better than sequence diagrams for uncovering problems.

The list of problems uncovered (see Figure 6) support the following inferences: that sequence diagrams more readily detect problems in the details of events; and that ScenarioML scenarios more readily permit detection of larger-scale problems related to clarity of requirements. Overall, 3 of the problems found in the translations were uncovered only using sequence diagrams, 15 problems were uncovered by both formats, and 13 additional problems were uncovered solely by ScenarioML scenarios. The data from the interviews provided evidence converging with that from comparing the lists of problems uncovered during translation, indicating stakeholders thought sequence diagrams more effective for design tasks in which layers and architectural components are important, while ScenarioML scenarios would be more effective for requirements, as a contract between stakeholder and developer, and for high-level communication. This triangulation across two quite different sources of data strengthen the inference drawn from each.

Overall, we infer that ScenarioML scenarios more readily permit the detection of errors than sequence diagrams to a substantial degree, but that sequence diagrams appear to more readily permit detection of detailed problems within individual events. There is some evidence that use cases are equivalent to sequence diagrams in permitting detection of errors, but this evidence is not strong.

### 6.2. Which requirements format has a structure that least obstructs understanding?

The degree to which the three formats obstruct understanding is addressed by the notes from the translation process, the comprehension tests, and the interviews. The student who translated use cases to ScenarioML scenarios commented that she frequently had to consult other documents for missing information. The comprehension test showed that all three stakeholders found the use case form obstructed understanding, with comments such as "it's not too helpful for me in understanding these types of issues at all", "doesn't do anything to point it [the problem] out", and "more difficult to spot what is a problematic step". The stakeholder who was familiar with sequence diagrams did not find any of the problems they contained (although he felt they were easy to understand). The two stakeholders who examined ScenarioML scenarios indicated the form did not obstruct their understanding, and noted ways in

which they felt the form may have helped. The interview data revealed essentially the same pattern: all stakeholders identified ways in which use cases obstructed their understanding; stakeholders' responses for sequence diagrams were divided by whether they understood the form or not; and all stakeholders commented on ways in which ScenarioML enhanced their understanding,while identifying two minor ways in which it obstructed their understanding. (The stakeholders did not understand that an iterated event numbered with $*$ indicated it may occur many times; and they wished they could optionally collapse compound events for clarity when looking at the overall picture.)

All three sources of data converged on the inferences that use cases most obstructed understanding, sequence diagrams did not obstruct understanding if one already understood the format, and ScenarioML tended to aid understanding rather than obstructing it.

### 6.3. Which requirements format do stakeholders find more clear?

The questionnaires and interviews were the sources of data for this question. All three stakeholders indicated they felt they understood the original use cases, although one also said of use cases "this form of specification is confusing to me". A second was less certain, saying "perhaps this is a problem with the user creating the use cases versus the format of the use case itself", while the third did not comment specifically. The responses for sequence diagrams corresponded to whether the stakeholder was already familiar with them. The two stakeholders who examined ScenarioML scenarios said they were "clear" and "do I understand what the event says? I think so ...", respectively.

In the interviews, all three stakeholders were neutral to moderately dissatisfied with use cases' ease of understanding, saying for example "it's hard to tell", "You don't necessarily get an idea of what the possible values are here", and "That's not that clear" (but also "Fairly easy" to understand, "pretty straightforward", and "they're easy to read"). The comments for sequence diagrams correlated with whether the stakeholder was familiar with them. All three stakeholders found ScenarioML scenarios easy to understand, with comments such as "Very easy", "unambiguous", "this is a very good form for a developer to get something in", "a richer, more defined way of doing use cases", "wow, absolutely, I think that's a definite improvement" (over use cases), "That's good, I like that", and "It's kind of like a UseCase++". Two stakeholders indicated contexts in which they felt ScenarioML scenarios would not be appropriate: "another guy in the company that's just an über-genius and something like this for him, just doesn't need it, wouldn't help him, would just get in the way", and "... if I was going to have ... users author use cases, they may not be familiar with this".

The data from the questionnaires and interviews converge, supporting the inference that the use case form is least well understood by stakeholders despite their comfort with the format, the sequence diagram form only understood by those familiar with it but apparently well understood in these cases, and that ScenarioML is easily understood by stakeholders despite their inexperience with it. The interview data on ScenarioML is strikingly positive on this point.

## 7. Lessons Learned

Our study has lessons that can be applied to selecting requirements formats, to designing requirements formats, and to empirical evaluation of requirements formats.

When selecting a requirements specification format, stakeholders and other software development participants should be aware that some requirements specification formats are inherently clearer than others. Although we do not have definitive metrics, we have cataloged ways in which formats can be unclear. For example, the numbering scheme for event sequences needs to be easy to follow and understand. Another problem can be found in graphical notations that obscure readability by visual overflow and confusing notations that lack clear definitions.

Clarity has been often overlooked in designing requirements specification formats, because of its seemingly simple character. Even though undergraduate textbooks in software engineering mention it, the IEEE standard 830 for Software Requirements Specification (SRS) [1] does not include clarity as one of its requirements quality attributes. This research presents both an operational definition for clarity and empirical methods to evaluate it. It shows the importance of clarity and suggests clarity to be taken more seriously. This study is a basis for further investigations on the clarity of requirements in regard to current and future formats.

## 8. Conclusion

In this paper, we reported on an empirical study that revealed the importance of clarity in requirements. We compared three requirements specification formats, use cases, sequence diagrams, and ScenarioML. Our empirical study consisted of creating documents in each of the formats, collecting data on their creation, and evaluating the usability of these documents from the point of view of stakeholders. We compared the formats themselves, our experience working with the formats, and stakeholder preference and understanding of information in the formats. The data from the transformation and user study showed that ScenarioML

was the format with the greatest clarity, followed by use cases, followed by sequence charts. Despite stakeholders' greater familiarity with use cases, they both preferred ScenarioML and found the format more clear. Our analytic results, stakeholders' subject opinions, and stakeholders' objective performance were are resounding endorsement of ScenarioML.

Despite some modest initial goals for this study, we were able to obtain a number of research contributions. The main contribution of this research is an understanding of the importance of clarity in requirements documents, and in turn requirements formats. While many quality attributes for requirements have been much more widely studied, little attention has been paid to clarity. Yet clarity is critical to the usability of a requirements format. Stakeholders and developers need to be able to find the information they need easily, understand that information, and know when they don't understand the information. We suspect that clarity has been little studied for three reasons: (1) clarity is the responsibility of the software engineer creating the requirements; (2) clarity is not an attribute that can be controlled at the level of a requirements format; and (3) clarity of requirements is assumed by other research and improvements beyond this are sought. Our research has brought all three reasons into doubt. It is possible for a requirements specification format to increase or decrease clarity, thereby helping or hindering a software engineer. And finally, it is possible and necessary to make research contributions that improve a fundamental characteristic such as clarity.

To this end, we have made a number of theoretical and methodological contributions to the study of clarity in requirements. We have formulated an operational definition of clarity. To reiterate, this definition is (1) clearly defined, (2) readily seen, perceived, or understood, (3) distinctness of vision, sound, expression, comprehension, etc., and (4) freedom from anything obstructive. The most valuable aspect of this definition is it can be made measurable and used in an empirical study. We have illustrated metrics for measuring clarity of a requirements format and a technique for applying the metrics. In addition, we have contributed a method for conducting comparative evaluation of requirements formats. Our approach includes a method for creating comparable requirements documents and a method for assessing the comprehension of stakeholders. These contributions can serve as a template for others who wish to conduct similar evaluations.

We have performed a careful analysis and thoughful empirical study of clarity in requirements. This work is only the beginning, but it paves the way for subsequent research into clarity as a requirements quality attribute, requirements specification formats that promote clarity, such as ScenarioML, and methods for studying requirements format quality. The benefits of clear requirements formats and in turn clear requirements documents abound. While Macbeth required clearness, to "leave no rubs nor botches in the work," he did not achieve it and this led to tragedy. We hope to avoid such sad endings for practitioners and stakeholders alike.

## References


[1] IEEE recommended practice for software requirements 830 specications, 1998.

[2] T. A. Alspaugh, B. Tomlinson, and E. Baumer. Using social agents to visualize software scenarios. In *ACM Symposium on Software Visualization (SoftVis'06)*, 2006.

[3] V. R. Basili, G. Caldiera, and H. D. Rombach. The goal question metric approach. In *Encyclopedia of Software Engineering*, pages 528–532. John Wiley and Sons, 1994.

[4] E. Baumer, B. Tomlinson, M. L. Yau, and T. A. Alspaugh. Normative echoes: use and manipulation of player generated content by communities of NPCs. In *Artificial Intelligence and Interactive Digital Entertainment (AIIDE-06)*, 2006.

[5] B. W. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.

[6] F. P. Brooks, Jr. No silver bullet: Essence and accidents of software engineering. *IEEE Computer*, 20(4):10–19, 1987. Reprinted from *Proceedings of the IFIP Congress,* Dublin, Ireland, 1986.

[7] A. Cockburn. *Writing Effective Use Cases*. Addison-Wesley Longman Publishing Co., Inc., 2000.

[8] A. M. Davis. *Software Requirements: Analysis and Specification*. Prentice-Hall, 1990.

[9] B. G. Glaser and A. L. Strauss. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine Publishing Company, 1967.

[10] I. Sommerville. *Software Engineering*. Addison-Wesley, third edition, 1989.

[11] K. Wasson. A case study in systematic improvement of langauge for requirements. In *Proceedings of The Fourth International Workshop on Comparative Evaluation in Requirements Engineering*, Minneapolis/St. Paul, Minnesota, USA, September 2006.

[12] K. Winbladh, T. A. Alspaugh, H. Ziv, and D. J. Richardson. An automated approach for goal-driven, specification-based testing. In *International Conference on Automated Software Engineering (ASE'06)*, 2006. To appear.