ISR Institute for Software Research
University of California, Irvine

# Surveying the Usability of Requirements Approaches using a 3-Dimensional Framework

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

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

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

August 2008

ISR Technical Report # UCI-ISR-08-3

**Institute for Software Research**
ICS2 221
University of California, Irvine
Irvine, CA 92697-3455
**www.isr.uci.edu**

**www.isr.uci.edu/tech-reports.html**

# Surveying the Usability of Requirements Approaches using a 3-Dimensional Framework

Kristina Winbladh, Hadar Ziv, Debra J. Richardson

Department of Informatics
Donald Bren School of Information and Computer Sciences
University of California, Irvine
{awinblad,ziv,djr}@ics.uci.edu

UCI-ISR-08-3

August, 2008

**Abstract**

Many problems in software development originate from requirements problems. Software is often late, does not meet requirements, or is costly because of expensive fault repairing. Software engineering researchers have developed a wide range of requirements approaches to address these challenges [10, 12, 14, 15, 18, 20, 21, 24, 31, 49], but it has been shown that many software development companies do not use these "textbook approaches" [7].

This survey describes a framework for evaluating the usability and usefulness of requirements engineering approaches. The evaluation framework focuses on three essential characteristics: clarity, testability, and ease of artifact manipulation. These three dimensions are decomposed into a set of sub-questions with corresponding metrics that can be used to assign scores for each evaluated requirements engineering approach. The survey evaluates twelve requirements approaches using the framework. The results show that none of the surveyed requirements approaches fulfill all three characteristics well. The survey concludes with a discussion of the results and directions for future research in requirements engineering.

# Contents

# 1 Introduction

> "The primary measure of success of a software system is the degree to which it meets the purpose for which it was intended. Broadly speaking, software systems requirements engineering (RE) is the process of discovering that purpose, by identifying stakeholders and their needs, and documenting these in a form that is amenable to analysis, communication, and subsequent implementation. [NE00]"

The goal of requirements engineering (RE) is to aid the elicitation, specification, analysis, and validation of stakeholder needs and wishes for a software system. RE is a continuous process that permeates every aspect of the software lifecycle due to the evolutionary nature of software and businesses that result in an endless demand to accommodate new needs. The product of a RE process is a *requirements specification* (model, artifact, and document are other common terms). In this paper, the term *requirements* refers to the product of the RE process and the term *RE approach* refers to a process or modeling scheme that aids RE activities such as elicitation, specification, analysis, and validation.

Software requirements are rapidly becoming the "final frontier" for development success and a persistent obstacle to significant improvements in project success rates [45]. RE is recognized as an important and difficult undertaking, and as such, the software community has produced a wide variety of RE approaches, e.g., [10, 12, 14, 15, 18, 20, 21, 24, 31, 49]. A persistent problem with most approaches advocated for RE is that they seem useful until one tries to apply them to examples that are different or more complex than ones provided in the literature. A recent study shows that many software developers refrain from using the myriad of requirements engineering (RE) approaches available [7]. We further hypothesize that RE stakeholders refrain from using existing approaches because the level of usability and usefulness of these approaches is not up to par with the needs of end users and other stakeholders of large-scale software development. On one hand, formal notations are notoriously difficult, cumbersome, and time-consuming, are not well-suited to the specification of modern user interfaces and user interaction modalities, do not scale well to industrial-strength systems, and are ineffective as a communication medium among non-technical stakeholders. Informal modeling schemes, on the other hand, produce requirements that are ineffective beyond the initial outline of the project and are therefore unlikely to be maintained throughout the project. A strictly informal modelling notation is often perceived ineffective because it does not directly support other software development activities such as design, implementation, and testing. Consequently, software developers, in both academia and industry, consider existing RE approaches neither easy nor cost-effective to use in practice.

In this survey we focus on the *usability* and *usefulness* of RE approaches rather than the muddled notion of formality of the approach. We consider factors other than formality to be the key factors and ingredients of RE, in other words we consider RE characteristics individually regardless of whether they typically would be classified as formal or informal. We believe that viewing RE from this fresh perspective could lead us in a novel and fruitful direction.

The term usability is frequently used in other areas, e.g., human computer interaction, but has not been explored with regard to RE. On the other hand, the RE literature contains a large set of characteristics with regard to the usefulness of RE approaches, but there is no consensus with regard to these characteristics. We therefore see a need and a motivation to define a set of characteristics that are important for the usability and usefulness of RE approaches. We also see a need to investigate whether RE approaches that exhibit these characteristics exist. If they do, it is an indication that the core of the RE problem lays elsewhere. If they do not, however, it is an indication that there is a need and an opportunity to specify an RE approach that better optimizes stakeholder needs with regard to usability and usefulness.

The contribution of this survey is three-fold. First, we present an evaluation framework consisting of a set of measurable criteria by which to evaluate the usability and usefulness of different RE approaches. Second, we describe a set of existing RE approaches and evaluate them against the framework. Third, we present a set of observations and recommendations for future RE research based on the results of the survey.

# 2 Evaluation Framework

In this section we describe the design of the evaluation framework that we use to survey existing RE approaches and the motivation underlying the design. We first describe a set of initial observations regarding some of the difficulties of current approaches. These initial observations are used as a basis for questions in an industry study, whose goal is to better understand the needs and wishes of industry stakeholders with respect to the usability and usefulness of RE approaches. We use the results of the industry study in two ways. First, we derive a set of characteristics that usable and useful RE approaches should exhibit and let these be the basis for our evaluation framework. Second, we derive a set of weights, corresponding to the relative importance of the different characteristics. These weights are used when computing scores for the different RE approaches that are evaluated. Finally, we describe the details of each characteristic in the framework and the metrics used for evaluating the level of fulfillment with regard to the characteristics.

## 2.1 Framework Design Motivation

A first step towards understanding the current practice of not explicitly stating stakeholder needs is to explore what usability and usefulness means for different RE stakeholders. We come up with the following set of initial observations:

- It is difficult for non-techinical stakeholders to understand the requirements specification well enough to be able to validate it.
- Requirements are often too ambiguous and vague for software designers to use as their basis for design.
- Software testers find it difficult to translate requirements into test cases and test oracles.
- Software testers find it difficult to determine what portion of requirements are covered by test cases.
- It is difficult for stakeholders to verify whether or not the final product meets stakeholder needs.
- It is difficult and time consuming for stakeholders to update requirements throughout the software project.
- It is difficult to motivate spending time to write requirements in the first place knowing that their use is limited per the problems described in previous bullet points.

These observations follow three main themes: clarity, testability, and maintainability. We let the observations above and the main themes be the basis of an industry study aimed to further explore the characteristics underlying usability and usefulness of RE approaches.

## 2.2 Industry Study

### 2.2.1 Study Design

The industry study was designed using the Goal-Question-Metric (GQM) framework (see Figure 1) [8]. The goal of the study is to explore and determine characteristics that concern the usability and usefulness of a RE approach. The overall goal is decomposed into two sets of questions, “Rate & Rank” and “Free text”; and standard statistics are used as metrics to evaluate the results.

The intention behind using both “Rate & Rank” and “Free text” questions is to get feedback on a set of characteristics that we perceive particularly important, i.e., clarity, testability, and manipulability, and also to gain knowledge about characteristics beyond the ones provided by us. The characteristics provided by us are listed below:

1. **Clarity:** Understandable requirements.

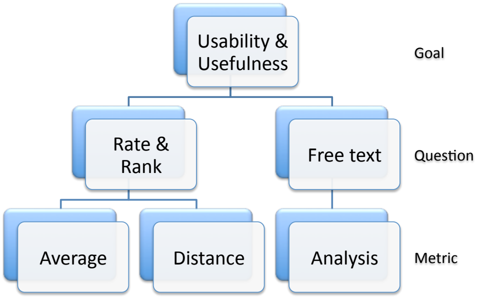


Figure 1: GQM decomposition for industry study.

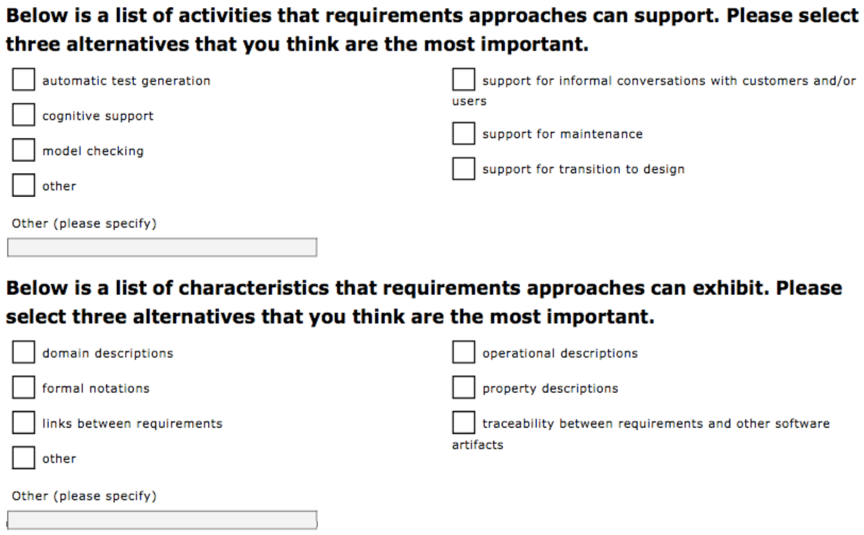
2. **Clarity:** Unambiguous requirements.
3. **Testability:** Using requirements for test case creation.
4. **Testability:** Using requirements for test data selection.
5. **Testability:** Using requirements for test coverage criteria definition.
6. **Testability:** Using requirements for test oracle creation.
7. **Testability:** High quality test artifacts (test cases, test data, test coverage, test oracles).
8. **Testability:** Easy to use the requirements for various testing activities.
9. **Manipulability:** Easy to initially create the requirements.
10. **Manipulability:** Easy to maintain the requirements throughout software development.

The participants are first asked to rate the importance of each characteristic and then to rank the characteristic relative to other characteristics within the same category. The combination of rating and ranking provides a better measurement than either one alone. The use of only rating typically generates over-optimistic results because every characteristic seems important and the result is that nothing is more important than anything else. The use of only ranking is problematic because we cannot assume even spacing between each pair of ranked items. When combining the two approaches we can get a better idea of the actual perceived importance of the characteristics. We evaluate the answers to the “Rate & Rank” questions by calculating the average of the ratings and the frequency count of the relative rankings.

The first question in the “Free text” section asks the participants to select three activities, among a list of activities, and/or enter their own activities, that are the most important for a RE approach to support. The second question of the “Free text” section asks the participants to select three characteristics among a list of characteristics, and/or enter their own characteristics, that are the most important for a RE approach to exhibit. Figure 2 shows the two questions and their alternatives. The metric used for both questions is a count of which alternatives are most frequently selected.

The remainder of the “Free text” section is a set of questions for which the participants provide free text answers (see list below). The metric for these questions is qualitative analysis of the answers.

1. What requirements approaches/notations do you use?
2. What are some weaknesses of the requirements approaches/ notations you use?
3. What characteristics or activities do you wish your requirements approaches/notations would support?
4. What characteristics of a requirements approach/notation do you consider the most important and why?



**Below is a list of activities that requirements approaches can support. Please select three alternatives that you think are the most important.**

□ automatic test generation

□ cognitive support

□ model checking

□ other

□ support for informal conversations with customers and/or users

□ support for maintenance

□ support for transition to design

Other (please specify)

**Below is a list of characteristics that requirements approaches can exhibit. Please select three alternatives that you think are the most important.**

□ domain descriptions

□ formal notations

□ links between requirements

□ other

□ operational descriptions

□ property descriptions

□ traceability between requirements and other software artifacts

Other (please specify)

Figure 2: First two questions of the "Free text" section.

5. What techniques do you use to validate your requirements, and what are the pros and cons of those?
6. Do you use requirements in testing activities? If so, what techniques do you use and what are the pros and cons of your current process?

We compare the results from rating and ranking and combine these with insights from the "Free text" section to establish a set of desirable characteristics for usable RE.

### 2.2.2 Study Results

When comparing the ratings of understandability and unambiguity to their relative, we see a slight difference in the two results. The rating shows that the two categories are equally important, but when forced to choose, participants chose understandability to be more important almost twice as often.

| Characteristic | Average Rating | Standard Deviation |
|---|---|---|
| Understandability | 4.56 | 0.58 |
| Unambiguity | 4.40 | 0.82 |

Table 1: Average rating of understandability and unambiguity

Although understandability and unambiguity are both important, we believe that it is often difficult to achieve both. In general unambiguity can be improved by more formality. More formality, however, generally decreases understandability for many software project stakeholders. One of the survey participants described the situation as follows:

> "The problem is that you can easily have multiple parties read something and say they understand it, yet have different understanding and expectations. This creates problems because you have an artifact that everyone understands, but differently, yet it does not stand out as ambiguous because everyone understands it. These [misunderstandings] then slip under the radar until the project comes to delivery and problems occur."

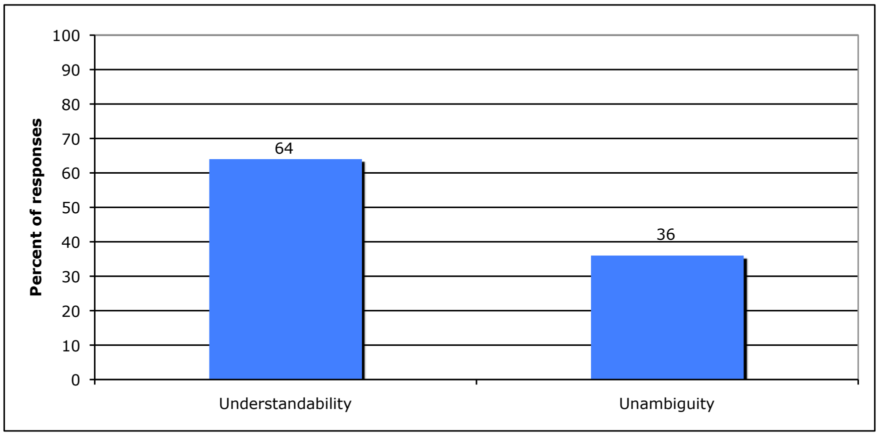


Figure 3: Ranking of understandability and unambiguity.

On the other hand, many participants pointed out that, although not an optimal solution, unambiguity could be identified and dealt with in later stages of development whereas understandability is absolutely essential up front. The responses to this question offer a new insight that there are two types of understandability: perceived understandability and true understandability. Perceived understandability is the ease with which stakeholders interpret the requirements, and true understandability is whether stakeholders actually understand the requirements properly.

The table below shows the average rating of the importance of a RE approach supporting four different testing activities.

| Characteristic | Average rating | Standard deviation |
|---|---|---|
| Test case creation | 4.68 | 0.48 |
| Test data selection | 3.82 | 0.96 |
| Test coverage criteria | 3.55 | 1.14 |
| Test oracle creation | 3.45 | 1.37 |

Table 2: Average rating of different testing characteristics.

Figure 4 shows the response distribution for first to fourth place ranking per testing activity and Figure 5 shows the overall ranking results. The results from rating and ranking correlate in that support for test case creation is perceived the most important, then support for test data selection, then support for test coverage criteria, and test oracle creation is perceived least important. Most survey participants find it highly important that a RE approach supports test case creation and test data selection.

In most companies software testing is mainly a human dependent and human intensive activity. As such, survey participants pointed out that rather than supporting particular testing activities, it is more important that the requirements are clear so that a human tester can use them for testing; this reinforces our previously defined characteristics understandability and unambiguity. One survey participant pointed out that a more important problem, than generating tests from the requirements, is generating tests at all. Software testing typically does not receive enough resources to be done sufficiently. The responses regarding testability reinforce our notion that requirements-based testing is quite important. Direct support in the form of automation for at least one or two of the testing activities is advantageous as it provides an additional direct benefit of using the requirements approach as well as it addresses the need for testing efficiency since testing is often under-budgeted.

When comparing the results of importance of initial manipulability and maintenance manipulability,

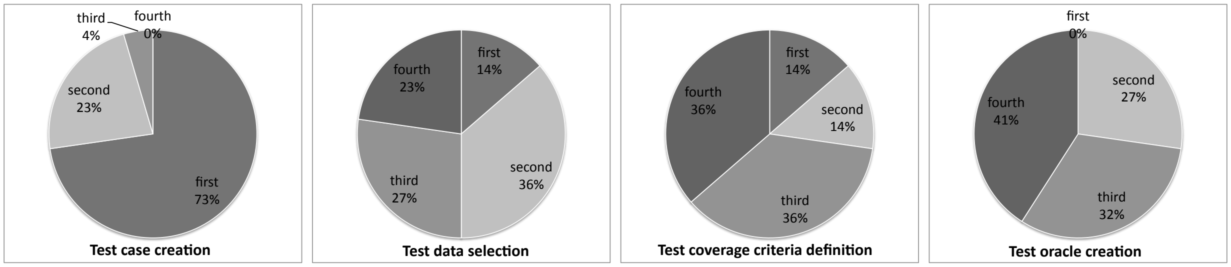


Figure 4: Ranking of test case creation, test data selection, test coverage criteria definition, and test oracle creation. The pie charts show the percentage of each testing characteristic being selected as most important to least important (first, second, third, and fourth).

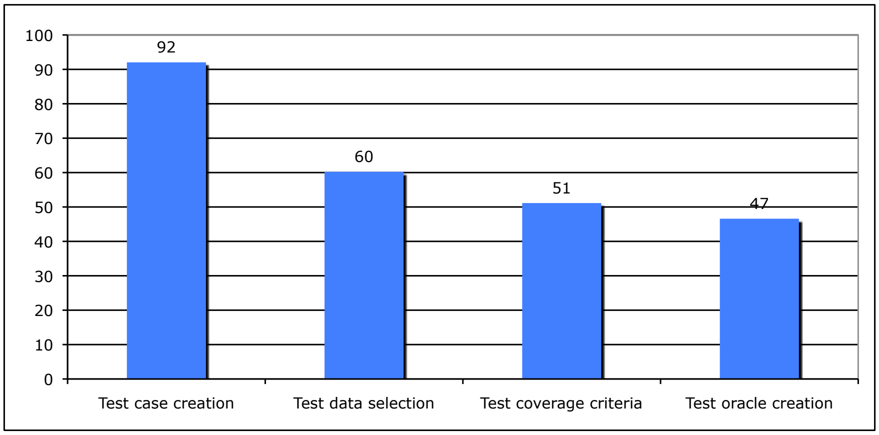


Figure 5: Overall ranking of the four testing characteristics.

both the rating (see Table 3) and ranking results (see Figure 6) show that it is highly important to support maintenance of requirements and that it is more important to support maintainability than initial manipulability.

| Characteristic | Average rating | Standard deviation |
|---|---|---|
| Initial manipulability | 3.45 | 1.10 |
| Maintenance manipulability | 4.41 | 0.59 |

Table 3: Average rating of initial manipulability and maintenance manipulability.

These results are certainly not surprising. Although maintenance is only important once requirements exist, maintenance is often more costly than initial effort and therefore requires better support. One survey participant gave the following motivation (others expressed similar motivations):

> “Reality: requirements are never completely specified up front, and things change midstream. It is more important to be able to easily update them during a project, because its almost a certainty that this will be happening often.”

The free text questions show that approximately 90% of the survey participants use informal textual requirements. Below is a typical response from one of the participants:

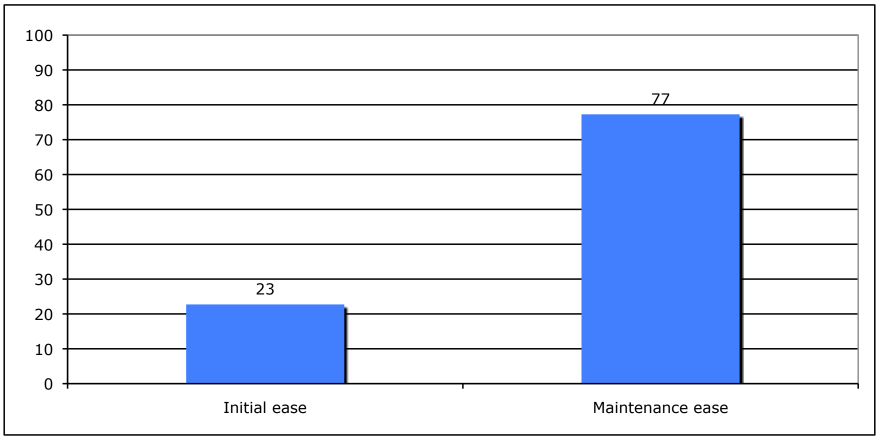


Figure 6: Ranking of initial manipulability and maintenance manipulability.

> "The requirements are text in various formats such as documents, spreadsheets, emails, meeting minutes. Sometimes they are worked out during software development (*shudder*). Yeah, this is the stone age, no doubt about it."

When asked about weaknesses of current approaches, most survey participants mentioned lack of formality or guidance when creating requirements; too much ambiguity; that clarity for one group of users seems to be at the expense of another group (developers, designers, customers); and that it is difficult to maintain the requirements as they change. When asked which characteristics they wished their RE approaches exhibit, the majority of survey participants answered that it has to be quick and easy to create and maintain the requirements. Several also mentioned that the requirements should be clear and unambiguous.

The top chart in Figure 7 shows activities the survey participants wish their RE approach would support. The four most popular alternatives are: support for maintenance, support for transition to design, automatic test generation, and support for informal conversations with customers. The bottom chart in Figure 7 shows the results for the question of what characteristics the survey participants wish their RE approach would exhibit. The most frequently selected alternative is operational descriptions. This alternative is followed by traceability between requirements and other software engineering artifacts, domain descriptions, and links between requirements. Most of the answers from the "Free text" questions reinforced the usability characteristics on our list above, but a few brought up new desirable characteristics such as designability and traceability.

## 2.3 Framework Structure

We use our initial list of characteristics together with the results of the study to formulate the evaluation framework. The structure of the framework is derived using the GQM method [8]. The goal is to evaluate the usability and usefulness of RE approaches. The goal is then decomposed into a set of questions whose answers will provide the foundation for achieving the goal. The decomposition of the goal into questions is based on our initial list of characteristics divided into the three categories clarity, testability, and manipulability. Designability and traceability, two new characteristics derived from the study, are not directly added to the evaluation framework. Instead designability and traceability will be discussed separately. This is because we are not yet certain of what features make a RE approach particularly useful for design and tracing activities. However, whenever the literature of a RE approach explicitly mentions support for design and/or traceability we will make note of that. The specific questions in each of the three categories are described in detail in subsequent subsections. Each specific question is decomposed into a set of metrics that are used to answer the question. The metrics used to answer the specific questions employ Likert items [35] ranging from zero

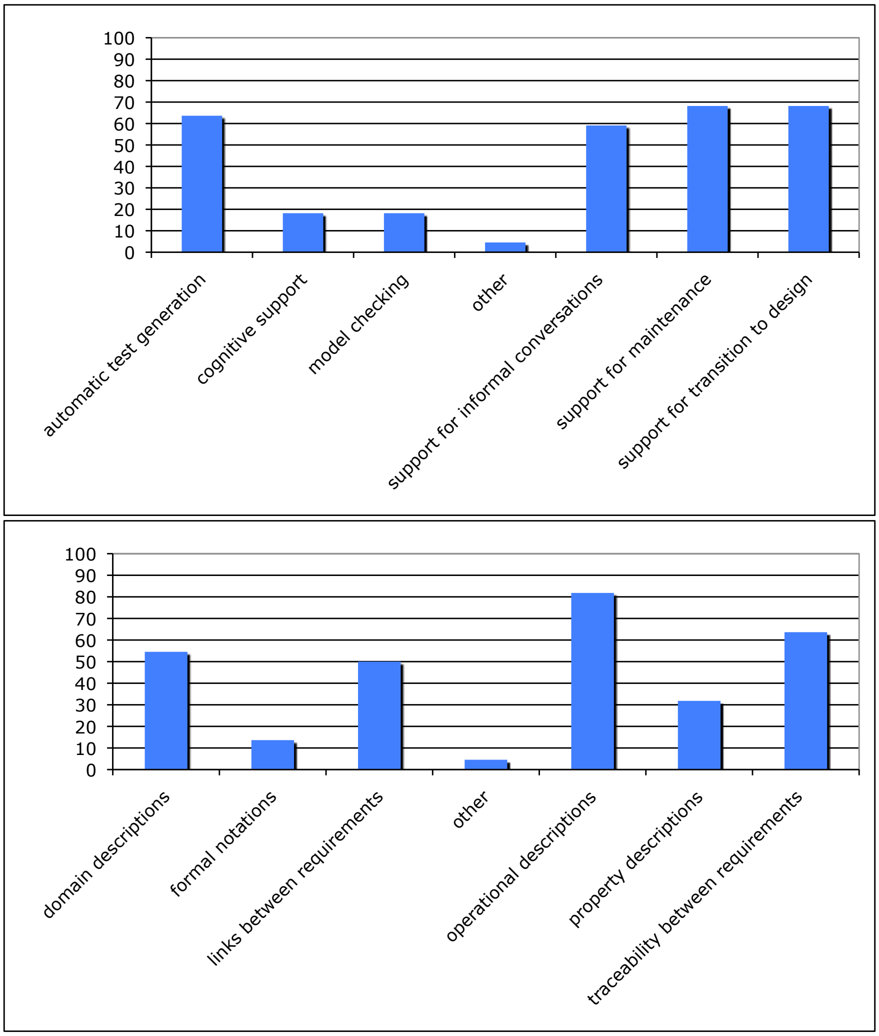


Figure 7: Activities and characteristics chosen as important for RE approaches to support.

to four points. Each requirements approach that is evaluated using the evaluation framework receives a total score in each dimension. A requirements approach's total score in a dimension is a normalized sum of the scores of the dimensions sub-dimensions. Details of the metrics are provided in subsequent subsections.

We will consider the three overall dimensions equally important, but we realize, however, the possibility of their sub-questions contributing unevenly with regard to their category. We therefore use the "Rate & Rank" results from the study described above to determine weights of sub-questions within each dimension. The weights are calculated by normalized geometric sum.

$$Understandability = \frac{\sqrt{\frac{4.56}{5}^2 + 0.64^2}}{\sqrt{\frac{4.56}{5}^2 + 0.64^2} + \sqrt{\frac{4.4}{5}^2 + 0.36^2}} = 0.54$$

$$Unambiguity = \frac{\sqrt{\frac{4.4}{5}^2 + 0.36^2}}{\sqrt{\frac{4.56}{5}^2 + 0.64^2} + \sqrt{\frac{4.4}{5}^2 + 0.36^2}} = 0.46$$

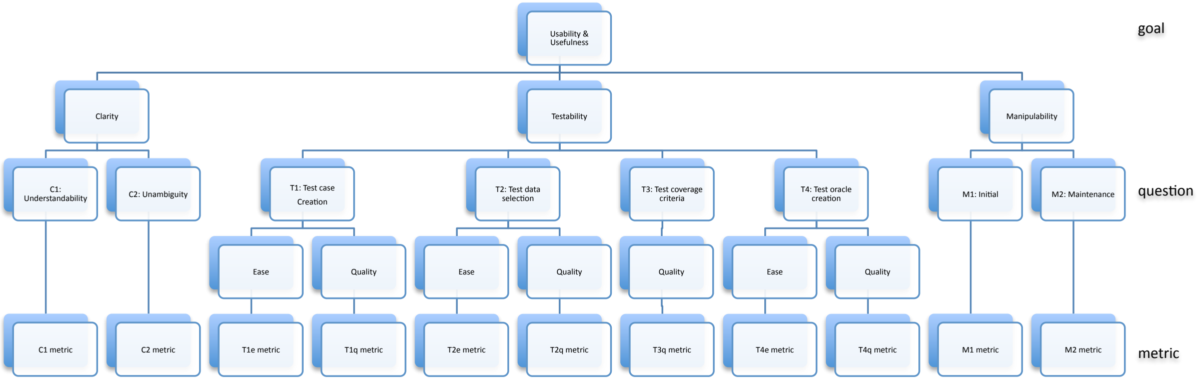


Figure 8: GQM decomposition of evaluation framework.

$$Test-case-creation = \frac{\sqrt{\frac{4.68}{5}^2 + 0.92^2}}{\sqrt{\frac{4.68}{5}^2 + 0.92^2} + \sqrt{\frac{3.82}{5}^2 + 0.6^2}\sqrt{\frac{3.55}{5}^2 + 0.51^2}\sqrt{\frac{3.45}{5}^2 + 0.47^2}} = 0.33$$

$$Test-data-selection = \frac{\sqrt{\frac{3.82}{5}^2 + 0.6^2}}{\sqrt{\frac{4.68}{5}^2 + 0.92^2} + \sqrt{\frac{3.82}{5}^2 + 0.6^2}\sqrt{\frac{3.55}{5}^2 + 0.51^2}\sqrt{\frac{3.45}{5}^2 + 0.47^2}} = 0.24$$

$$Test-coverage-definition = \frac{\sqrt{\frac{3.55}{5}^2 + 0.51^2}}{\sqrt{\frac{4.68}{5}^2 + 0.92^2} + \sqrt{\frac{3.82}{5}^2 + 0.6^2}\sqrt{\frac{3.55}{5}^2 + 0.51^2}\sqrt{\frac{3.45}{5}^2 + 0.47^2}} = 0.22$$

$$Test-oracle-creation = \frac{\sqrt{\frac{3.45}{5}^2 + 0.47^2}}{\sqrt{\frac{4.68}{5}^2 + 0.92^2} + \sqrt{\frac{3.82}{5}^2 + 0.6^2}\sqrt{\frac{3.55}{5}^2 + 0.51^2}\sqrt{\frac{3.45}{5}^2 + 0.47^2}} = 0.21$$

$$Initial-manipulability = \frac{\sqrt{\frac{3.45}{5}^2 + 0.23^2}}{\sqrt{\frac{3.45}{5}^2 + 0.23^2} + \sqrt{\frac{4.41}{5}^2 + 0.77^2}} = 0.38$$

$$Maintenance-manipulability = \frac{\sqrt{\frac{4.41}{5}^2 + 0.77^2}}{\sqrt{\frac{3.45}{5}^2 + 0.23^2} + \sqrt{\frac{4.41}{5}^2 + 0.77^2}} = 0.62$$

## 2.4 Questions and Metrics

### 2.4.1 Clarity Questions and Metrics

The following sections describe the decomposition of clarity into two more precise questions with regard to understandability and unambiguity (C1 and C2) and the metrics to use for answering each question.

C1. What is the level of **understandability** of requirements created using this requirements method?
**Evaluation metrics:** $\oslash, \star, \star\star, \star\star\star, \star\star\star\star$
$\oslash$ – Requirements produced with this approach have been reported to be difficult to understand, require significant training in the modeling scheme, the modeling scheme seems difficult to learn, or for some other reason the requirements are not intuitively understandable by both technical and non-technical stakeholders.
$\star$ – Requirements produced by this approach seem difficult to understand, require training in the modeling scheme, or for some other reason appear to not be intuitively understandable by both technical and non-technical stakeholders.
$\star\star$ – Requirements produced by this approach have neither been reported as understandable or not understandable, could possibly require some training in a modeling scheme, or for some other reason appear somewhat intuitively understandable by both technical and non-technical stakeholders.
$\star\star\star$ – Requirements produced by this approach have been reported to be somewhat understandable, the modeling scheme requires very little training, or for some other reason the requirements seem somewhat intuitively understandable by both technical and non-technical stakeholders.
$\star\star\star\star$ – Requirements produced by this approach have been reported as easy to understand or clear, do not require any or very little training in a language/notation, or for some other reason appear to be highly intuitively understandable by both technical and non-technical stakeholders.

C2. What is the level of **unambiguity** of requirements created using this requirements method?
**Evaluation metrics:** $\oslash, \star, \star\star, \star\star\star, \star\star\star\star$
$\oslash$ – Requirements produced by this approach have been reported to be highly ambiguous.
$\star$ – Requirements produced by this approach have been reported to be notably ambiguous, or it is obvious that the language/notation leaves a lot of room for different interpretations.
$\star\star$ – Requirements produced by this approach have been reported to be somewhat ambiguous, or it seems that the language/notation leaves some room for different interpretations.
$\star\star\star$ – Requirements produced by this approach seem to have a low level ambiguity, or seem to leave little room for different interpretations.
$\star\star\star\star$ – Requirements produced by this approach have been reported to have a very low level of ambiguity, or it seems that there is very little room for different interpretations.

| Clarity | Question | Evaluation Criteria |
|---|---|---|
| | Understandability | $\oslash, \star, \star\star, \star\star\star, \star\star\star\star$ |
| | Unambiguity | $\oslash, \star, \star\star, \star\star\star, \star\star\star\star$ |

Table 4: Summary of questions and evaluation criteria for clarity.

### 2.4.2 Testability Questions and Metrics

Testability is decomposed into the four testing activities: test case creation, test data selection, test coverage criteria definition, and test oracle creation. This sub-division is selected because it is important that the requirements support each major testing activity explicitly. The decomposition into activities is not specific enough for evaluating the level of support a requirements approach provides. Each activity is therefore further decomposed into a question regarding the level of ease of using the requirements for a particular testing activity and a question regarding the level of quality of the product of the activity. The following sections describe the sub-framework by defining a question that represents each testing activity (see T1-T4) and two questions (ease and quality) that further decompose the evaluation criteria for each activity. The subsections also describe the evaluation metrics used to answer each question.

T1. How well does the requirements approach support **test case creation**?
**Ease:** How easy is it to use the requirements for test case creation?

**Background:** This question is concerned with the level of manual and automatic support for the process of creating test cases from the requirements. The level of support for the activity is directly related to the amount of work required by a tester. The metrics for the question are concerned with amount of labor and specific support or possibility for support if none exist.

**Evaluation metrics:** ⊘, ⋆, ⋆⋆, ⋆⋆⋆, ⋆⋆⋆⋆

⊘ – The requirements approach does not support test case creation manually or automatically, or it does not seem viable that the requirements approach could be used for manual or automatic test case creation.

⋆ – The requirements approach offers weak support for manual test case creation or it seems that the requirements approach could offer weak support for manual test case creation.

⋆⋆ – The requirements approach offers moderate support for manual test case creation or it seems possible that the requirements approach could be used for manual test case creation.

⋆⋆⋆ – The requirements approach offers satisfactory support for manual test care creation, or some support for automatic test case creation, or it seems possible that the requirements approach could be used for manual or automatic test case creation.

⋆⋆⋆⋆ – The requirements approach offers strong support for manual test case creation, or satisfactory support for automatic test creation, or it is highly likely that the requirements approach can be successfully used for manual or automatic test case creation.

**Quality:** What level of quality of test cases can be created from the requirements?

**Background:** This question is concerned with the quality of the support for test case creation as well as the quality of the resulting test cases. The metrics for the question are concerned with whether the process uses the requirements alone or relies on supplemental documentation, whether the test cases cover the most important behavior described by the requirements, whether the test cases describe a flow of actions/events/results, the size of the resulting test suite, and the connection between test cases and high-level requirements.

**Evaluation metrics:** ⊘, ⋆, ⋆⋆, ⋆⋆⋆, ⋆⋆⋆⋆

⊘ – Test case creation relies heavily on supplemental documentation, particularly low-level design specifications. The test suite is large and there is no support for selecting important test cases. The test cases do not have flow. There is no connection between test cases and high-level requirements.

⋆ – Test case creation relies somewhat on supplemental documentation. There is no support for selecting important test cases. The test cases do not have flow. There is very little connection between test cases and high-level requirements.

⋆⋆ – Test case creation relies only somewhat on supplemental documentation, but no low-level design specifications. There could be some support for selecting important test cases. The test cases have flow. There could be a weak connection between test cases and high-level requirements.

⋆⋆⋆ – Test case creation does not rely on any supplemental documentation. There is some support for selecting important test cases. The test cases have flow. There is some connection between test cases and high-level requirements.

⋆⋆⋆⋆ – Test case creation does not rely on any supplemental documentation. There is satisfactory support for selecting important test cases. The test cases have flow. There is good connection between test cases and high-level requirements.

T2. How well does the requirements approach support **test data selection**?

**Ease:** How easy is it to use the requirements for test data selection?

**Background:** This question is concerned with the level of manual and automatic support for the process of selecting test data from the requirements. The level of support for the activity is directly related to the amount of work required by a tester. The metrics for the question are concerned with amount of labor and specific support or possibility for support if none exist.

**Evaluation metrics:** ⊘, ⋆, ⋆⋆, ⋆⋆⋆, ⋆⋆⋆⋆

⊘ – The requirements approach does not support manual or automatic test data selection, or it does not seem viable that the requirements approach could be used for manual or automatic test data selection.

⋆ – The requirements approach offers weak support for manual test data selection or it seems that the

requirements approach could offer weak support for manual test data selection.
$\star\star$ – The requirements approach offers moderate support for manual test data selection or it seems possible that the requirements approach could be used for manual test data selection.
$\star\star\star$ – The requirements approach offers satisfactory support for manual test data selection, or some support for automatic test data selection, or it seems possible that the requirements approach could be used for manual or automatic test data selection.
$\star\star\star\star$ – The requirements approach offers strong support for manual test data selection, or satisfactory support for automatic test data selection, or it is highly likely that the requirements approach can be successfully used for manual or automatic test data selection.

**Quality:** What level of quality of test data can be selected from the requirements?
**Background:** This question is concerned with the quality of the support for test data selection as well as the quality of the resulting test data selected. The metrics for the question are concerned with whether the process uses the requirements alone or relies on supplemental documentation, whether the test data covers different categories of input, e.g., normal, exceptional, and boundary, described by the requirements, and the connection between test data and high-level requirements concepts or actors and other domain concepts.
**Evaluation metrics:** $\oslash, \star, \star\star, \star\star\star, \star\star\star\star$
$\oslash$ – Test data selection relies heavily on supplemental documentation, particularly low-level design specifications. There is no support for selecting data from different categories of test data. There is no connection between test data and high-level requirements.
$\star$ – Test data selection relies somewhat on supplemental documentation. There is no support for selecting test data from different data categories. There is very little connection between test data and high-level requirements.
$\star\star$ – Test data selection relies only somewhat on supplemental documentation, but no low-level design specifications. There could be some support for selecting test data from different categories. There could be a weak connection between test data and high-level requirements.
$\star\star\star$ – Test data selection does not rely on any supplemental documentation. There is some support for selecting test data from different data categories. There is some connection between test data and high-level requirements.
$\star\star\star\star$ – Test data selection does not rely on any supplemental documentation. There is satisfactory support for selecting test data from different data categories. There is good connection between test data and high-level requirements.

T3. How well does the requirements approach support determining **test coverage criteria** that estimate how well the test cases cover the requirements?
**Background:** It is desirable to measure how much of the requirements are covered by test cases in order to determine the progress and quality of the software under development. Below is a list of common code coverage criteria definitions modified to be applicable to requirements specifications. Coverage is defined by the structure of the specification, and the task of defining a coverage criterion is therefore not associated with work for the tester. However, there is work associated with measuring the actual coverage of executing a test suite. Explicit relations between test cases and requirements could reduce that work, but the topic of work associated with test case execution is out of scope for this evaluation. This testing activity only has one specific sub-question, which is with regards to the quality of the coverage possible.

1. Function coverage
   (a) Code coverage: Has each function in the program been executed?
   (b) Specification coverage: Has each requirement in the specification been tested?
2. Statement coverage
   (a) Code coverage: Has each line of the source code been executed?

   (b) Specification coverage: Has each ”line” (could be event, or state, logic expression, etc) been tested?

3. Condition coverage
   (a) Code coverage: Has each evaluation point (such as a true/false decision) been executed?
   (b) Specification coverage: Has every evaluation point in the specification been tested?

4. Path coverage
   (a) Code coverage: Has every possible route through a given part of the code been executed?
   (b) Specification coverage: Has every possible route through the specification been tested?

**Evaluation metrics:** ⊘, ⋆, ⋆⋆, ⋆ ⋆ ⋆, ⋆ ⋆ ⋆⋆
⊘ – The requirements approach does not support defining test coverage criteria at all, or it does not seem viable that the requirements approach could be used for defining test coverage criteria.
⋆ – The quality of test coverage criteria defined over the requirements is low, i.e., function coverage or equivalent.
⋆⋆ – The quality of test coverage over the requirements is reasonable, i.e., statement coverage or equivalent.
⋆ ⋆ ⋆ – The quality of test coverage over the requirements is fairly satisfactory, i.e., path coverage or equivalent.
⋆ ⋆ ⋆⋆ – The quality of test coverage over the requirements is excellent, i.e., path coverage and condition coverage, or equivalent.

T4. How well does the requirements approach support **test oracle creation**?
**Ease:** How easy is it to use the requirements for test oracle creation?
**Background:** This question is concerned with the level of manual and automatic support for the process of creating test oracles from the requirements. The level of support for the activity is directly related to the amount of work required by a tester. The metrics for the question are concerned with amount of labor and specific support or possibility for support if none exist.
**Evaluation metrics:** ⊘, ⋆, ⋆⋆, ⋆ ⋆ ⋆, ⋆ ⋆ ⋆⋆
⊘ – The requirements approach does not support manual or automatic test oracle creation, or it does not seem viable that the requirements approach could be used for manual or automatic test oracle creation.
⋆ – The requirements approach offers weak support for manual test oracle creation or it seems that the requirements approach could offer weak support for manual test oracle creation.
⋆⋆ – The requirements approach offers moderate support for manual test oracle creation or it seems possible that the requirements approach could be used for manual test oracle creation.
⋆⋆⋆ – The requirements approach offers satisfactory support for manual test oracle creation, or some support for automatic test oracle creation, or it seems possible that the requirements approach could be used for manual or automatic test oracle creation.
⋆ ⋆ ⋆⋆ – The requirements approach offers strong support for manual test oracle creation, or satisfactory support for automatic test oracle creation, or it is highly likely that the requirements approach can be successfully used for manual or automatic test oracle creation.
**Quality:** What level of quality of test oracles can be created from the requirements?
**Background:** This question is concerned with the quality of the support for test oracle creation as well as the quality of the resulting test oracles created. The metrics for the question are concerned with whether the process uses the requirements alone or relies on supplemental documentation, the level of precision of the oracles created, e.g., machine executable, human executable, and the connection between test oracles and high-level requirements concepts.
**Evaluation metrics:** ⊘, ⋆, ⋆⋆, ⋆ ⋆ ⋆, ⋆ ⋆ ⋆⋆
⊘ – Test oracle creation relies heavily on supplemental documentation, particularly low-level design specifications. The oracles created are imprecise. There is no connection between test oracles and high-level

requirements.
⋆ – Test oracle creation relies somewhat on supplemental documentation. The oracles created are fairly imprecise. There is very little connection between test oracles and high-level requirements.
⋆⋆ – Test oracle creation relies only somewhat on supplemental documentation, but no low-level design specifications. The oracles created are moderately precise. There could be a weak connection between test oracles and high-level requirements.
⋆⋆⋆ – Test oracle creation does not rely on any supplemental documentation. The oracles created are fairly precise. There is some connection between test oracles and high-level requirements.
⋆⋆⋆⋆ – Test oracle creation does not rely on any supplemental documentation. The oracles created are precise. There is good connection between test oracles and high-level requirements.

| Testability | Ease | Quality |
|---|---|---|
| Test case | ⊘, ⋆, ⋆⋆, ⋆⋆⋆, ⋆⋆⋆⋆ | ⊘, ⋆, ⋆⋆, ⋆⋆⋆, ⋆⋆⋆⋆ |
| Test data | ⊘, ⋆, ⋆⋆, ⋆⋆⋆, ⋆⋆⋆⋆ | ⊘, ⋆, ⋆⋆, ⋆⋆⋆, ⋆⋆⋆⋆ |
| Test coverage | NA | ⊘, ⋆, ⋆⋆, ⋆⋆⋆, ⋆⋆⋆⋆ |
| Test oracle | ⊘, ⋆, ⋆⋆, ⋆⋆⋆, ⋆⋆⋆⋆ | ⊘, ⋆, ⋆⋆, ⋆⋆⋆, ⋆⋆⋆⋆ |

Table 5: Summary of the questions and evaluation criteria for the testability dimension.

### 2.4.3 Manipulability Questions and Metrics

Manipulability refers to the ease by which creation, modification, and deletion of requirements within a requirements collection can be carried out, and is an essential characteristic of requirements. The category is decomposed into the manipulability during initial creation and the manipulability during maintenance. A certainty in software development is that the requirements are going to change over time. It is irrelevant whether the changes are because of initial misunderstandings or because of a change of heart, but it is important that the requirements can be modified with minimal effort. If the notation does not support the task of manipulation, it is unlikely that the requirements are updated and usable throughout the software lifecycle. The following sections will describe the two sub-dimensions by defining a question for each (see M1 and M2) and define a set of evaluation metrics for answering each question.

M1. How easy is it to create requirements using this requirements method?
**Evaluation metrics:** ⊘, ⋆, ⋆⋆, ⋆⋆⋆, ⋆⋆⋆⋆
⊘ – The requirements approach requires major effort to produce requirements.
⋆ – The requirements approach requires a considerable amount of effort to produce requirements.
⋆⋆ – The requirements approach requires a moderate amount of effort to produce requirements.
⋆⋆⋆ – The requirements approach requires a small amount of effort to produce requirements.
⋆⋆⋆⋆ – It is trivial to use the requirements approach to produce requirements.

M2. How easy is it to modify requirements using this requirements method?
**Evaluation metrics:** ⊘, ⋆, ⋆⋆, ⋆⋆⋆, ⋆⋆⋆⋆
⊘ – The requirements approach requires major effort to maintain requirements.
⋆ – The requirements approach requires a considerable amount of effort to maintain the requirements.
⋆⋆ – The requirements approach requires a moderate amount of effort to maintain the requirements.
⋆⋆⋆ – The requirements approach requires a small amount of effort to maintain the requirements.
⋆⋆⋆⋆ – It is trivial to use the requirements approach to maintain the requirements.

| Manipulability | Questions | Evaluation Criteria |
|---|---|---|
| | Initial | $\oslash, \star, \star\star, \star\star\star, \star\star\star\star$ |
| | Maintenance | $\oslash, \star, \star\star, \star\star\star, \star\star\star\star$ |

Table 6: Summary of the questions and evaluation criteria for manipulability.

## 2.5 Data Analysis

A total score for each of the three dimensions is calculated by propagating the bottom-level scores up through the GQM structure. The numerical scores are determined by the rating received (number of $\star$s) and then normalized over the possible rating in each sub-category to get a result between 0 and 1. The constants c1, c2, t1, t2, t3, t4, m1, m2, are weights representing the relative importance of each sub-question within its dimension. These weights are determined through the industry study and provided above.

The following equations are defined to determine the scores:

$$\begin{aligned}
\text{Clarity} \quad &= \quad c1 \times \frac{understandability\ score}{understandability\ total} + c2 \times \frac{unambiguity\ score}{unambiguity\ total} \\
\text{Testability} \quad &= \quad t1 \times \frac{test\ case\ ease\ score}{test\ case\ ease\ total} + \frac{test\ case\ quality\ score}{test\ case\ quality\ total} + \\
&\qquad t2 \times \frac{test\ data\ ease\ score}{test\ data\ ease\ total} + \frac{test\ data\ quality\ score}{test\ data\ quality\ total} + \\
&\qquad t3 \times \frac{test\ coverage\ quality\ score}{test\ coverage\ quality\ total} + \\
&\qquad t4 \times \frac{test\ oracle\ ease\ score}{test\ oracle\ ease\ total} + \frac{test\ oracle\ quality\ score}{test\ oracle\ quality\ total} \\
\text{Manipulability} \quad &= \quad m1 \times \frac{initial\ manipulability\ score}{initial\ manipulability\ total} + m2 \times \frac{maintenance\ manipulability\ score}{maintenance\ manipulability\ total}
\end{aligned}$$

The overall scores and per category scores will help give insights regarding the current state of the art as well as point out areas of potential improvements.

## 2.6 Framework Limitations

This survey framework is incomplete in the sense that it does not address every important property that a requirements approach can address. The attempt is to select a set of properties that are valuable and viable to evaluate with regard to the usability and usefulness of requirements approaches. The three dimensions of the framework were selected as three essential traits that seem non-conflicting, i.e., there is no obvious reason that a requirements approach cannot score high on all three dimensions.

There are also obvious risks with obtaining weights from the questionnaire data, e.g., misunderstandings, lack of background knowledge of requirements engineering, and other biases. However, the use of questionnaire data is a less biased approach than relying completely on the first author's perception. Questionnaire questions were designed to limit biases. For example, each question had a paragraph explaining terms in order to reduce misunderstandings, the order of sub-categories were randomized in order to reduce order-biases, and each question had a comment box in which participants could motivate their answers and/or explain their viewpoint. It is also noteworthy that there are many other ways to obtain weights and the current set of weights is in no way considered a "true" set of weights.

There are natural risks with assigning numbers to qualitative questions such as the ones in this survey. In order to mitigate these risks, the survey uses Likert items, which in general is an accepted evaluation-scale for qualitative and social science studies [35]. Furthermore, the Likert items stay consistent over the set of surveyed approaches so that the potential flaws in quantifying qualitative items are the same in each evaluation. Another potential problem is the range of five values in the Likert items and the translation of these into numerical values used in the scoring mechanism. For example, two requirements approaches could receive the same score within a Likert item although one is on the lower bound and one is on the upper bound. During data analysis, the score is used in weighted summations in which the imprecision could grow larger. We mitigated some of the risks in rating by also using forced ranking. This allows us to better compute a score that matches the intention of the participant. This survey addresses imprecision by not basing conclusions on individual data points, but instead on larger trends and when appropriate standard deviations are reported.

The evaluations of the surveyed approaches pose a threat to the validity of the results reported, because the evaluations are solely based on the authors understanding of each surveyed approach. The risks were addressed by using publicly available material such as research papers and research sites and by using more than one source for each requirements approach.

# 3 Survey of RE Approaches

The selection of requirements approaches for the survey is based on a requirements approach's capacity of modeling the composite system. A broad definition of requirements states that requirements should focus on what the proposed system should do. A more refined description, supported by many RE researchers [7, 18, 31, 34], states that requirements need to model the composite system, which means that the requirements should be expressed in terms of the application domain and model the relations between the application domain and the proposed system. Some approaches commonly denoted as requirements approaches, and initially considered for this survey are: Message Sequence Charts [53], Modal Transition Systems [19], Specification Description Language [1], and Software Cost Reduction [23]. These approaches were excluded from the survey because they typically focus on modeling requirements in terms of system properties and because of the inherent difficulty for non-technical stakeholders to understand these notations. The requirements approaches selected for the survey are:

- User stories
- Informal use cases
- Semiformal use cases
- Formal use cases
- Shall statements
- RETNA (natural language using natural language processing)
- Scenarios
- Functional documents
- Requirements Modeling Language
- KAOS
- I*
- Problem frames

## 3.1 Agile User Stories

User stories are short stories describing some feature that the proposed system should exhibit. User stories are typically used as one form of requirements in agile development processes. They are primarily intended for project planning and not to fulfill the same variety of purposes that requirements do in conventional software development. In Agile development, the requirements are typically not documented but are internal models, i.e., someones unrecorded understanding. The internal requirements models are often made tangible in the form of test cases. Although Agile employs user stories, test cases, and internal models as requirements, only user stories are selected to represent Agile requirements in this survey. The other two forms are not appropriate because a persons internal requirements models do not generalize and cannot be evaluated, and because testability is one part of the evaluation framework and it does not make sense to evaluate the testability of test cases.

User stories are typically short, one-sentence, descriptions of a feature that the software system should exhibit [17]. For example, “As a User, I want to cancel my hotel reservation.” is a user story from a travel booking software system. The main benefits of using user stories is that they focus on user needs, they are short and concise, and easy to create and maintain throughout development. The lack of detail in user stories is to be clarified through interactions with onsite customers.

A major disadvantage of using user stories is that the terse documentation can severely hinder software maintenance, which is often described as one of the most extensive and important development activities [46, 20]. Maintenance can become particularly difficult when developers and customers change over time, and their knowledge is not recorded in any detail other than user stories, test cases, and the source code, important system knowledge and design decisions can be lost.

**Clarity evaluation.** User stories are too short to provide enough detail. For example, the user story for the travel booking system above does not mention whether the user gets a full or partial refund, how far ahead a reservation must be canceled, and if the policy is the same for all users. The agile process attempts to solve this by a set of sub-stories such as "As a premium user, I can cancel my reservation up to 24 hrs in advance." The problem with this is that similar or connected stories become difficult to oversee and connect. Another problem is that there are no clear definitions or connections to domain concepts that describe entities such as user and premium user. Furthermore, user stories often contain notes such as interface drawings and class structures to provide more detail. This indicates that the modeling scheme is not rich enough to express the kind of information that is needed. The ambiguity of natural language and lack of details in user stories do promote misunderstandings.

Recent research shows that user stories are inefficient from a time perspective. Gallardo et al. describe a study, which compared the time spent to understand the requirements using use cases, user stories combined with onsite customers, and user stories combined with use cases and onsite customers [22]. The study showed that user stories combined with onsite customers was the most time consuming process and that it did not result in a significantly better product. This indicates, again, that the user stories are not rich enough to be easily understood. It is understood that onsite customers should provide the details needed, however, the inefficiency of the spoken word over the written word is as obvious as the gained efficiency education that was obtained through the printing machine.

**Testability evaluation.** Agile development is sometimes referred to as Test Driven Development, because tests are created prior to implementation. There are clear benefits to early testing and test driven development, however, there is no explicit support for any testing activities directly from user stories. User stories can at best be used to create test cases manually. Most Agile literature describe the process of setting up automatic unit tests, including the concepts of setting a start state, describing the state change, and then the goal state. Obtaining these states from the requirements is however completely unsupported by current literature. It is clear that user stories have very little impact on the actual test case design. Agile test design principles, are in themselves good principles, but are in no way related to or aided by the user stories [9, 38, 50, 56].

Although there is no documented support for testability of user stories, user stories have some positive testability traits. User stories focus on the use of the system and often describe interactions that should be testable. However, because user stories are not machine-readable, no direct return on investment can be given to developers in terms of automatically creating test cases from those interactions.

The accumulative score for test data selection is pretty low, because user stories fail to describe the kind of data that are involved in the stories. Test case coverage also receives a low score, because the coverage possible evaluates only to functional coverage, i.e., there is no support for branch or conditional coverage. Furthermore, user stories do not support the creation of test oracles very well. It is difficult to determine whether the outcome of a user interaction is correct or what actually is supposed to happen during and as a result of an interaction.

**Ease evaluation.** User stories are reported as easy to create and maintain because of their short and concise format. The term "agile" in agile development refers to being able to effectively deal with changing requirements. Proponents of agile development state that modifiability and requirements evolution is supported by the informal and brief nature of the user stories. It seems however that the decomposition of requirements into individual brief user stories makes it difficult to determine the impacts of one changing requirement with regard to all other requirements. Furthermore, particular details that are obtained through customer interaction are not reflected in the user story collection, which makes it extremely difficult to determine the

impacts of change.
**Designability & Traceability** There is not specific mention about user stories being utilized in design or whether they are easily traceable from later lifecycle artifacts. It seems that traceability could be implemented between user stories and test cases in particular.

| Category | Question | | Evaluation Score |
|---|---|---|---|
| Clarity | Understandability | | ⋆⋆⋆ |
| | Unambiguity | | ⊘ |
| Testability | Test case | Ease | ⋆ |
| | | Quality | ⋆ |
| | Test data | Ease | ⊘ |
| | | Quality | ⊘ |
| | Test coverage | Quality | ⋆⋆ |
| | Test oracle | Ease | ⋆ |
| | | Quality | ⋆ |
| Manipulability | Initial | | ⋆⋆⋆⋆ |
| | Maintenance | | ⋆ |

Table 7: Evaluation of User Stories

## 3.2 Informal Use Cases

Use cases are popular and in general more expressive than user stories. Use cases are typically structured natural language constructs that describe sequences of interactions with the software system [15, 16]. There are several different styles of writing use cases. Alistair Cockburn's style of use cases was selected for this survey [15], because it is the most common and generally accepted style used in both academia and industry. Cockburn's use cases, allow three levels of details: use case brief, casual use case, and fully dressed use case. Guidelines state that different projects and different teams will find different levels appropriate. Cockburns use case style addresses two main needs for a use case format: (1) assert that use cases really are requirements and need a basic structure, and (2) allow people to write whatever they want when they need to. Although he refers to the fully dressed use case as a semi-formal notation, this survey treats it as informal because the formalism only helps with the structure of the use case, but does not reduce the level of ambiguity of natural language in any way.

Each fully dressed use case is a natural language structure that has a name, a goal, a brief description, a flow of events, a set of special requirements - such as non-functional requirements, pre-conditions, and post-conditions. The flow of events contains the basic flow of events and the alternate flow of events. Each step in these flows should explain what the actor does and what the system does in response. A scenario is a particular path through the use case.

Use cases have gained their popularity because they allow people to express the functionality of the software system without describing its form and because use cases are typically perceived as easy to write and understandable to any stakeholder. The original developer of use cases, Ivar Jacobson, intended them to be an informal requirements method because he found that most people resist writing them whenever they become more formal [33]. At a first glance use cases seem easy to create, it has been shown however that it takes certain skills and practice to write good use cases [36].

Use cases are often used in test case creation. Heuman describes a process by which to create test cases from use cases [28]. The test case creation is an entirely manual process that systematically creates test cases that cover each use case path. The tester creates a full set of use case scenarios from each use case. Each use case scenario is then used to create at least one test case and the conditions that will make it execute. As a last step, the tester selects data to be used in the test case. The process by which to select high-quality test data and appropriate conditions is not explicitly supported by the testing process, and it is unclear whether use cases can indeed support such activities.

**Clarity:** The narrative nature of use cases makes it easy to understand particular functionality and behavior of the system and its environment. Use cases can be related to one another in a use case diagram, which aids getting a comprehensive view of system. A major problem is that there are no clear definitions or connections to domain concepts that describe entities referenced in the use cases. In a comparative study of use cases, ScenarioML scenarios, and message sequence charts, it was shown that software developers and research students found use cases to be unclear and ambiguous. Further investigation indicates that it is the structure of normal and exception flows, the use of natural language, and the lack of connection to a domain model that give rise to most of the shortcomings [2].

**Testability.** Informal use cases can be used to create test cases manually. A positive testability characteristic of use cases is that they focus on the use of the system and describe interactions that should be testable.

The main weaknesses that impede testability are lack of automation capabilities and low quality. Informal use cases are written in natural language, which means that they are not machine-readable and do not provide project participants any direct return on investment in terms of automatic processing. The overall quality of test cases, test data, test coverage criteria, and test oracles that can be produced from use cases is fairly low. The low quality is because the behavior, data, and expected result described in use cases are abstract and ill defined. Test oracles are defined to be the last event in the use case, and it is unclear how the tester can use this to create a precise and executable oracle for the test cases. The use cases cannot directly be used to select interesting data values for test cases. Although, the structure of use cases makes it possible to define path coverage, the lack of path conditions makes it unlikely to define a stricter coverage criterion.

**Ease.** Use cases are reported as fairly easy to create and maintain because of their short and concise format. However, learning to create good use cases does require some amount of training [36]. One problem when maintaining use cases is that because there are no definitions or agreed-upon standards regarding events and terms, the use case collection can become highly inconsistent and difficult to understand with time.

**Designability & Traceability.** Use cases can be used in design activities to determine system entities and processes. There is no particular traceability mechanism available in informal use cases, but they could be associated to various other artifacts through a traceability matrix.

| Category | Question | | Evaluation Score |
|---|---|---|---|
| Clarity | Understandability | | ⋆⋆⋆ |
| | Unambiguity | | ⋆ |
| Testability | Test case | Ease | ⋆ |
| | | Quality | ⋆⋆ |
| | Test data | Ease | ⊘ |
| | | Quality | ⊘ |
| | Test coverage | Quality | ⋆⋆⋆ |
| | Test oracle | Ease | ⋆ |
| | | Quality | ⋆ |
| Manipulability | Initial | | ⋆⋆⋆ |
| | Maintenance | | ⋆ |

Table 8: Evaluation of Informal Use Cases

## 3.3 Semi-formal Use Cases

The problems with informal use cases have led to a number of attempts to formalize their representation [3, 11, 24, 29, 32, 37, 42, 40]. As reported by Cockburn, most of these attempts have had mild success rates which led him to propose a semi-formal use case method instead, which according to Cockburn, prevents getting trapped in too much or too little formalism [15, 16]. Cockburn uses the term semi-formal to distinguish his most unstructured use case type, the use case brief, from his most structured use case type, the fully dressed use case. Neither of his use case types are considered semi-formal in the sense of using mathematical constructs to reduce ambiguity, and therefore not considered semi-formal in this survey. This survey considers semi-formal use cases to use some kind of mathematical notation in combination with informal notations. The following section reports on and evaluates a semi-formal use case notation developed by Nebut et al. The notation was selected from a set of semi-formal approaches [3, 40], because it was developed with a consideration for testing.

Nebut et al. describe a semi-formal use case notation, which uses first order logic pre-, and post-conditions that are called contracts [44]. The contracts are used to automatically generate a simulation model. The simulation model treats the use cases as black boxes and constructs a diagram in which the nodes are conditions (representing states) and the directed edges are use cases. Test objectives are automatically generated from the simulation model. A test objective is a path through the simulation model, i.e., a list of use cases. The method uses design information to bridge the abstraction gap between requirements and testing the code. Every use case is associated with a sequence diagram that specifies the actual classes and messages that the use case describes at a high level. Together the interaction diagrams and the test objectives are used to generate test scenarios.

**Clarity:** Nebut et al.'s use cases are structured natural language with elements of first order logic. The first order logic use case contracts make the use cases a bit more complicated to read than Cockburn's informal use cases. In general, the first order logic contracts do not obscure what the use cases describe, but restrict the number of interpretations somewhat and thereby reduces the level of ambiguity compared to informal use cases. However, the use of natural language and lack of formalization on the internal events inside the use cases still contribute to ambiguities.

**Testability:** The approach supports several testing activities and it is noticeable that the modeling scheme was designed with software testing in mind. The main disadvantage with the approach is that it is highly dependent on design artifacts, namely sequence diagrams, which indicates that once the design artifacts are created there is little incentive to keep the requirements up to date. The design artifacts could in fact replace the use cases.

Test data selection is not supported by this approach because it does not support data modeling. Test oracles are generated by the post conditions of the use cases; they can therefore check whether or not the test case produced the appropriate end result. The test oracles cannot calculate expected data or check intermediate steps and are automatically created from a specification that is to a large extent manually created. The approach can be used to create test scenarios, which are high-level test cases, only lacking concrete data. The test cases cover a lot of functionality, but fail to address boundary data or improper behavior. The test scenarios are fairly coarse in that they only treat conditions before and after a use case, no intermediate events.

Creation of test scenarios is automated to some extent, but full test case automation is not implemented. The authors describe how different coverage criteria such as all edges and all statements can be applied to the simulation model when generating the test objectives. The coverage criteria cannot be stricter to cover the domain of data possible in the logical conditions in the contracts.

**Ease:** The semi-formal use cases are fairly straightforward to create with the additional challenge being the creation of first order logic contracts. The approach relies on design elements, which increases the complexity. Furthermore, it seems a bit inefficient to use two narrative structures, first use cases and then sequence diagrams, representing the same thing at different levels of abstraction. A non-narrative mapping between use case events and system API calls would probably be sufficient and ease up the effort of requirements maintenance.

The requirements method does require some special training particularly in writing first order logic conditions and designing the contracts of the use cases. Although, maintenance means updating both use cases and sequence diagrams, the approach does have analysis support, which should help in identifying which use cases need updates.

**Designability & Traceability.** The semi-formal use cases rely on design artifacts, but the approach does not mention particular ways of deriving designs from the use cases. There are explicit traceability links between use cases and design artifacts and tests.

| Category | Question | | Evaluation Score |
|---|---|---|---|
| Clarity | Understandability | | ⋆⋆⋆ |
| | Unambiguity | | ⋆⋆ |
| Testability | Test case | Ease | ⋆⋆ |
| | | Quality | ⋆⋆ |
| | Test data | Ease | ⊘ |
| | | Quality | ⊘ |
| | Test coverage | Quality | ⋆⋆⋆ |
| | Test oracle | Ease | ⋆ |
| | | Quality | ⋆ |
| Manipulability | Initial | | ⋆⋆ |
| | Maintenance | | ⋆⋆ |

Table 9: Evaluation of Semiformal Use Cases

### 3.4 Formal Use Cases

There are many different formalization schemes with the goal of making use cases applicable to machine processing. Some researchers derive a state machine model from a set of use cases [42, 57], whereas several others focus on generating different formal sequence models such as interaction diagrams and message sequence charts from the set of use cases [3, 11, 37, 40]. A problem with these approaches is that most of them rely heavily on human translation, which becomes particularly cumbersome when trying to achieve consistency between the use cases and the formal models throughout the software development lifecycle. Because the formalization techniques listed above do not directly affect the notation scheme of use cases, but rather associate use cases with formal models, they were not chosen for this survey. This section describes and evaluates a formal use case approach in which use cases are encoded in Abstract State Machine Language (ASML) [26].

The ASML technique relies on annotating the informal use cases with ASML statements. In fact, each step in a use case must have an annotation. ASML provides mathematical types and notations for sets, maps, and sequences. The annotations describe transactions between the system and some actor. Each use case starts with having a global state; and states are a collection of variables and their current values. The authors describe an encoding technique that translates the use cases' interactions to a set of transitions that update the global state. The translation of annotated use cases to the core form that can be used to build state automata is completely automatic. Test oracles define functions that test whether a given dialogue, or path, matches a valid transition path in a use case. Test cases are generated using an all paths algorithm. If a test path produces infeasible paths, the test oracle will determine that the path is not feasible.

**Clarity:** The annotations are fairly easy to understand. The authors draw parallels between their use of annotations and pseudo-coding and make the argument that most software developers are comfortable with these types of descriptions. Pseudo-code annotations could however be difficult to understand by non-technical stakeholders. Although the annotations describe the interactions between the system and other actors by defined functions, there are still ambiguities because the entities in the use cases are not referred to.

**Testability:** The approach is clearly designed with testability in mind and provides support for test case creation, coverage criterion definition, and test oracle creation. The approach does not offer any support for test data selection. It would be possible to support this activity if a data model was given and associated with the use cases. There is explicit support for oracle creation. The quality of the oracles however is fairly low because their sole responsibility is to check whether a given path is in fact a valid path of a use case thereby ignoring any checking of intermediate and final results of operations. A problem with both the test case creation and test oracle creation is their abstract representations, which indicate significant effort of translating these into executable forms. Coverage is also supported in the form of all paths, which is a weak coverage for the amount of work that the approach requires. It would have been better to have a stronger coverage criterion that also handles conditions.

**Ease:** The approach does require training in ASML. The notation is not extremely complex, however there seem to be a large initial effort involved with defining all the interactions between the system and the actors and then referring to these in the use cases. There was for example no mention of tool support for this task and it seems that it can be both difficult and error prone. The approach does not offer any support for annotating the use cases, defining the appropriate interactions, and maintaining the use case annotations. The low score in ease of manipulation can be one reason for why formal use cases are unlikely to be accepted by software developers. Ivar Jacobson, the founder of use cases, has been quoted to say: “Oh, I have a formal model for use cases, all right. The only problem is that no one wants to use it” [16].

**Designability & Traceability.** The approach does not mention explicit support for design activities, hence same support as informal use cases apply. There are traceability links between use cases and test cases.

| Category | Question | | Evaluation Score |
|---|---|---|---|
| Clarity | Understandability | | ⋆⋆ |
| | Unambiguity | | ⋆⋆ |
| Testability | Test case | Ease | ⋆⋆⋆ |
| | | Quality | ⋆⋆ |
| | Test data | Ease | ⊘ |
| | | Quality | ⊘ |
| | Test coverage | Quality | ⋆⋆⋆ |
| | Test oracle | Ease | ⋆ |
| | | Quality | ⋆ |
| Manipulability | Initial | | ⋆⋆ |
| | Maintenance | | ⋆ |

Table 10: Evaluation of Formal Use Cases

### 3.5 Shall Statements

Shall statements are natural language descriptions of system properties. An example shall statement from an avionics system states: “If this side is active and the mode annunciations are off, the mode annunciations shall be turned on when the onside FD is turned on. [41]”. Shall statements are used by a various software development projects in NASA, Rockwell Collins, and Lockheed Martin [41], and have been reported as useful to capture initial ideas regarding the software system to be constructed.

The apparent flaws of shall statements are that the natural language is highly ambiguous and that the list of properties often is incomplete and inconsistent. Another problem is that shall statements refer to concepts that might or might not be explained by other documentation. In the example above, concepts such as “mode annunciations” and “onside FD” might not be understandable to everyone. Furthermore, shall statements are often written from a system’s view rather than from the environments view, although this does not have to be the case.

It has been reported that these problems, unless detected early, lead to costly corrections in later development stages [41]. As an attempt to remedy this, shall statements can be translated into formal logic statements and then analyzed for inconsistencies, ambiguities, and incompleteness by model checkers and theorem provers [27, 41]. The translation is however manual and therefore both expensive and error-prone, i.e. the same kind of problems that happen when implementing shall statements in the software system happen when translating them into formal models because an implementation is a formal model. The attempts to translate shall statements into formal models seem mostly like attempts to patch the problem with shall statements rather than reform the requirements notation and address its problems.

**Clarity:** Shall statements are easy to read and understand by most project participants, because the requirements are written in natural language and formatted to describe one feature per statement. One major problem with shall statements is that they are not user-centric but rather tend to take a systems view, which means that it is difficult to determine whether or not the requirements address actual user needs. The use of natural language implies that the requirements are ambiguous.

**Testability:** Shall statements cannot easily be tested because a test plan for how to show that each statement is implemented in the system and behaves correctly must be created. This is obviously very difficult and also time consuming. If such a plan were constructed, the test cases would cover each functionality specified by the requirements.

**Ease:** There is not much effort to initially create a set of shall statements, however, it is cumbersome to analyze and maintain these because they are individual and disconnected statements.

**Designability & Traceability** There is no explicit or implicit support for either design activities or traceability.

| Category | Question | | Evaluation Score |
|---|---|---|---|
| Clarity | Understandability | | ⋆⋆⋆ |
| | Unambiguity | | ⊘ |
| Testability | Test case | Ease | ⊘ |
| | | Quality | ⊘ |
| | Test data | Ease | ⊘ |
| | | Quality | ⊘ |
| | Test coverage | Quality | ⋆ |
| | Test oracle | Ease | ⊘ |
| | | Quality | ⊘ |
| Manipulability | Initial | | ⋆⋆⋆⋆ |
| | Maintenance | | ⋆⋆ |

Table 11: Evaluation of Shall Statements

## 3.6 (RETNA) Natural Language w. Natural Language Processing

Most of the approaches so far have used natural language descriptions or structured natural language descriptions as the heart of the notation scheme. This section describes another natural language approach that makes use of natural language processing (NLP) to create a formal model of the requirements. There have been several attempts to employ natural language processing for requirements analysis [4, 39]. This section will describe and evaluate a more recent approach, RETNA [10], which was developed with regard to requirements-based testing.

RETNA uses several steps to transform the highly informal natural language sentences to a formal model, which is a state machine. The natural language requirements are first classified as either conditional or non-conditional. The complexity of the statements is then calculated as the sum of the number of verbs (tasks) and the number of nouns (roles). These statements are then translated into an intermediate predicate argument structure. A different translation then translates the statements to discourse representation structure. Then the anaphoric pronouns are resolved (the cataphoric pronouns lack implementation) [1]. Then there is a search in the undefined predicates library and a user validates the definitions found. The structures become

[1] An anaphoric pronoun is a word in a sentence that refers back to a noun and a cataphoric pronoun is a word in a sentence that refers forward to a noun.

predicate calculus definitions, which are translated into FMONA and then compiled into MONA, which finally undergo a conversion to state-machines [2].

The logical specification is divided into input specifications and output specifications. The input specifications are used to generate test cases where each path from an initial state to a rejecting state becomes a test case. The output specifications are used to generate test oracles. The approach relies on supplemental documentation in the form of a dictionary and ontology to define roles and relationships among roles.

**Clarity:** The natural language requirements are easy to read and understand for anyone involved with the software development project. The original requirements are highly ambiguous and the requirements resulting from a complicated translation process which relies on several manual steps has a fairly low level of ambiguity. To show the full advantage of RETNA, this evaluation will use the informal representation when measuring understandability and the formal representation when measuring unambiguity.

**Testability:** The use of natural language processing was proposed and implemented with the goal of achieving requirements-based analysis and testing. The main problem with RETNA is the involvement of humans in the translation process. This involvement is both costly and error-prone. The other major problem with RETNA is that the natural language requirements need to be almost written in first order logic as it is, which suggests that perhaps there is a faster manual translation process of the natural sentences to a state machine. Testing with RETNA relies on other documents in the form of ontologies and dictionaries, and also the formal model that is created through the NLP.

A problem with the test case generation is that a very large number of test cases are generated (reported in millions). It is also apparent that the test cases and test oracles need manual translation to become executable on the system under test.

The main advantages of RETNA lie in automation possibilities and the documented support for test case generation and test oracle creation. The obvious disadvantages are the amount of effort required to use RETNA for testing and that the quality of the test cases is reduced a lot by the considerable volume of them.

**Ease:** RETNA requires a substantial amount of effort. The natural language requirements need to be almost in first order logic to begin with and even so the many translation steps in the natural language processing method require a lot of manual intervention. Maintenance becomes particularly cumbersome because the entire translation process has to take place every time a requirement needs updating.

**Designability & Traceability.** There is no explicit support for design activities. The transformation into a formal model could possibly be used in design validation. There are explicit traceability links between requirements and tests.

| Category | Question | | Evaluation Score |
|---|---|---|---|
| Clarity | Understandability | | ⋆⋆⋆ |
| | Unambiguity | | ⋆⋆⋆ |
| Testability | Test case | Ease | ⋆⋆⋆ |
| | | Quality | ⋆⋆ |
| | Test data | Ease | ⊘ |
| | | Quality | ⊘ |
| | Test coverage | Quality | ⋆⋆⋆ |
| | Test oracle | Ease | ⋆⋆⋆ |
| | | Quality | ⋆⋆ |
| Manipulability | Initial | | ⊘ |
| | Maintenance | | ⊘ |

Table 12: Evaluation of RETNA (Natural Language Processing)

[2] FMONA is a tool for expressing validation techniques over infinite state systems and MONA is a tool that translates formulas to finite-state automata.

## 3.7 Scenario-based Requirements

Scenarios represent the operation of some proposed system. Operational scenarios require the requirements engineer or other project participant to describe and envisage behaviors and courses of events. There are numerous scenario-based requirements approaches [5, 48, 49, 51, 52, 53]. Many of the approaches provide means for describing lower-level scenarios at a system level [51, 53]. This survey describes and evaluates a scenario-based requirements approach developed by Potts et al. [49]. The notation scheme was developed to describe requirements from a high-level composite system's view and is associated with a requirements engineering process, which has been successfully evaluated.

Potts et al. use scenarios in a requirements engineering process called the Inquiry Cycle Model [5, 49]. The scenarios are natural language descriptions of events that occur in the world and interact with the proposed system. The main difference between scenarios and use cases, user stories, and just natural language requirements is that the scenario notation scheme allows scenarios to be related to one another through sub-classing and episodes. Sub-classing indicates that two or more scenarios are related through specialization relations and are said to belong to the same scenario family. Episodes indicate the common events shared between two or more scenarios. Case studies in scenario-based requirements engineering have shown that scenarios play a major role in uncovering goals during requirements exploration. Scenarios have also been shown as highly useful for weeding out ambiguities and conflicts early in the development cycle because they are in a format that many different stakeholders can relate to and discuss. Furthermore, it has been shown that the more concrete the scenarios are the better they are for stakeholder discussions.

**Clarity:** The scenarios are in natural language, so they are easy to understand for any non-technical and technical stakeholder. Because scenarios are written in natural language they do inherently contain ambiguities. It seems that the narrative structure, used in a process such as the Inquiry Model, could aid in discovering ambiguities. However, the notation scheme does not support documenting these in a less ambiguous way for future reference.

**Testability:** There is currently no support for testing activities using scenarios. However, one could see that user scenarios could be useful sources of test cases, but such translation would be entirely manual. Since there is no associated data model for the scenarios, there is no support for test data selection. If test cases were manually created from scenarios, the test cases would cover each usage path described by a scenario, which would be equivalent to path coverage of the requirements model.

**Ease:** Natural language scenarios are fairly easy to create. The use of inheritance and episodes also supports re-use of the requirements, which could reduce the effort. The explicit relation of inheritance and episodes also facilitates maintenance of the requirements to some extent.

**Designability & Traceability.** There is not explicit support for design activities nor for traceability between scenarios and other lifecycle artifacts. Scenarios, could similarly to use cases, be used to identify design entities and processes.

| Category | Question | | Evaluation Score |
|---|---|---|---|
| Clarity | Understandability | | ⋆⋆⋆ |
| | Unambiguity | | ⋆ |
| Testability | Test case | Ease | ⋆ |
| | | Quality | ⋆⋆ |
| | Test data | Ease | ∅ |
| | | Quality | ∅ |
| | Test coverage | Quality | ⋆⋆⋆ |
| | Test oracle | Ease | ∅ |
| | | Quality | ∅ |
| Manipulability | Initial | | ⋆⋆⋆⋆ |
| | Maintenance | | ⋆⋆⋆ |

Table 13: Evaluation of Scenario-Based Requirements

## 3.8 Functional Documents

Parnas' functional documents represent its own category because there has not been much work in the area since his introduction of the idea. Functional documents are included in the survey because the approach describes a meta-model of what kind of information the requirements specification should contain, but is impartial of the particular notation of the specification. The idea is interesting because it focuses on the essence of requirements engineering, which is focusing on specifying the problem.

According to Parnas, the requirements document should identify the environmental quantities to be measured or controlled and the representation of those quantities by mathematical variables and the association between environmental quantities and their mathematical variables must be carefully defined [47]. Often diagrams are essential to clarify the correspondence between physical quantities and assigned variables. It is useful to characterize each environmental quantity as either monitored, controlled, or both. Monitored quantities are those that the user wants the system to measure, and controlled quantities are those whose values the system is intended to control. Each of the environmental quantities has a value that can be recorded as a function of time. The environment places constraints on the values of environmental quantities, and these constraints must be recorded in the requirements document. This relationship is known as NAT (for nature). The computer system is also expected to impose further constraints on the environmental quantities, and these constraints should also be recorded and are known as the relation REQ (permitted behavior). Feasibility of the requirements is then determined by investigating the relation between the domain of NAT and the domain of REQ.

Although no particular notation is prescribed for the functional documents, it seems to restrict the kind of expressions that can be used. It also restricts the requirements engineer from using any non-mathematical notations. Functional documents seem most useful for describing requirements of control systems and embedded systems, but the requirements approach is not the most appropriate for other kinds of systems. It is also unclear what the mathematical constructs are used for beyond checking the feasibility of the requirements. Models created by using the approach have not reportedly been used for testing or other analysis.

**Clarity:** It seems difficult for non-technical stakeholders to understand functional documents, because they are entirely mathematical. The mathematical notation clearly limits the amount of ambiguities in the documentation, and any inconsistencies can be found through model checking.

**Testability:** Software testing activities are not explicitly supported by the functional documents. This is mainly because the approach does not prescribe any particular modeling language. However, the mathematical constructs can possibly be used to come up with test cases of acceptable and non-acceptable behavior. The notion of controlled and monitored variables could be used to select test data. None of the testing activities have automated support; it is however possible, that some testing activities could be automated based on the modeling scheme chosen for the functional document.

**Ease:** Functional documents seem difficult to create because they limit the level of expressiveness and restrict the kind of information that the document can contain. The notation scheme has to be mathematical which inevitably requires training. Maintenance is difficult because the requirements statements are not explicitly related within the document. Manual work must go in to evaluating the impacts of change over the entire set of requirements.

**Designability & Traceability.** Models generated by functional documents could possibly be used in design activities, although this has not been reported on explicitly. There is no explicit support for traceability between the requirements model and other lifecycle artifacts.

## 3.9 RML – Requirements Modeling Language

RML (Requirements Modeling Language) is an interesting, object-oriented requirements modeling notation in its own category. There is no larger category of similar approaches, however RML is the predecessor of other approaches such as i* and TROPOS.

RML (Requirements Modeling Language) is an object-oriented framework for describing requirements [12, 25]. The notation is to be used for describing the world in which the system under development will

| Category | Question | | Evaluation Score |
|---|---|---|---|
| Clarity | Understandability | | ⊘ |
| | Unambiguity | | ⋆⋆⋆ |
| Testability | Test case | Ease | ⋆ |
| | | Quality | ⋆⋆ |
| | Test data | Ease | ⋆⋆ |
| | | Quality | ⋆⋆ |
| | Test coverage | Quality | ⋆⋆ |
| | Test oracle | Ease | ⋆ |
| | | Quality | ⋆⋆ |
| Manipulability | Initial | | ⋆ |
| | Maintenance | | ⋆ |

Table 14: Evaluation of Functional Documents

live and focuses on providing a means for describing that world. Everything in an RML model can be represented by classes, this includes even activities and assertions as well as entities. An important feature in requirements models is the notion of time. This is particularly important when describing objects and their interactions. RML uses Allen's intervals to represent time, and every object should specify its own start and end. This could be in absolute terms or in relation to other objects. Particular classes of time can also be used in the modeling.

The main ideas underlying RML come from knowledge representation in AI. RML was given semantics by translation into First Order Predicate Calculus (FOPC). The underlying representation is in terms of states, which are represented by predicates and functions. RML uses default deductions, or frame problems, which means that during a specified state change anything that is not explicitly mentioned is interpreted as staying the same [13]. The literature on RML does not describe what kind of analysis is appropriate on the model, however, state-based representations can be used to show that certain properties hold or do not hold. Other work in specification-based testing has used state-based specifications to generate test cases. This work has, to my knowledge, not included RML specifications in particular.

**Clarity:** RML requirements are fairly difficult to understand. The object oriented modeling notations are approximately equivalent to reading programming language constructs. Ambiguities are limited because of the formal notation and can be found by model checking the FOPC properties.

**Testability:** RML does not explicitly support any testing activities. The use of objects could perhaps be used for test data selection, but a fair amount of manual labor in selecting important and appropriate data would be needed. The FOPC constructs could possibly be used for creating test cases that follow particular paths through the state space. It is however unclear whether entire state spaces are modeled and what is intended for the FOPC constructs. The low level of detail in this work does not provide enough information to speculate what might be possible.

**Ease:** RML is complex to use for writing requirements. It is unclear why RML is any better than existing object-oriented languages. The notation scheme would require a substantial amount of work and it is unclear whether the product of the requirements approach is actually useful for anything. Maintaining the requirements would not be any easier than initially creating them. There is no explicit support for maintenance activities.

**Designability & Traceability.** It seems that RML could be useful in design activities, as it is highly object oriented and could translate nicely into design entities and processes. There is no explicit support for traceability.

| Category | Question | | Evaluation Score |
|---|---|---|---|
| Clarity | Understandability | | ⊘ |
| | Unambiguity | | ⋆⋆⋆ |
| Testability | Test case | Ease | ⊘ |
| | | Quality | ⋆⋆ |
| | Test data | Ease | ⊘ |
| | | Quality | ⋆⋆ |
| | Test coverage | Quality | ⋆⋆⋆ |
| | Test oracle | Ease | ⊘ |
| | | Quality | ⋆⋆ |
| Manipulability | Initial | | ⋆ |
| | Maintenance | | ⋆ |

Table 15: Evaluation of RML

### 3.10 KAOS (Goal-based)

KAOS is selected as a representative approach within a group of goal-based requirements. Other well-known approaches include work from Antn and Potts [6], the NFR Framework by Mylopoulos et al. [43], and i* [58] and TROPOS [14], by Mylopoulos and Yu et al. TROPOS is not included in this survey because it is a software development process rather than a requirements engineering approach. i*, on the other hand, is surveyed in the next section. i* is evaluated in its own category because it is an actor-based approach that relies on goal models to some extent. The NFR framework is not included explicitly in this survey because it is a part of i*, and thus reported implicitly on in the next section.

The KAOS (Knowledge Acquisition in autOmated Specification) approach focuses on the satisfaction of goals [18, 55, 54]. Goals are needs that the proposed system should fulfill, or needs that are in some way important to consider when developing the proposed system, and can be both functional and non-functional. A reason to use goals in requirements models is that they inherently support requirements engineers to focus on the underlying reasons for the system and various stakeholders wishes. The underlying notation scheme of KAOS supports creation of complete, conflict-free goal-based requirements models that describe the composite system, i.e., the proposed system and its environment. The modeling language explicitly supports requirements acquisition and elaboration. The language consists of a set of abstractions for capturing requirements in terms of goals, constraints, objects, agents, events, and actions. It also offers support to express links between these abstractions to capture the refinements, conflicts, responsibility assignments, and operationalizations. The KAOS framework consists of a conceptual model with a specification language for acquiring and structuring requirements models, a set of strategies for elaborating requirements models, and an automated assistant to provide guidance in the acquisition process according to such strategies.

In KAOS keyword verbs such as “Achieve”, “Maintain”, and “Avoid” specify temporal logic patterns for goals. The verbs implicitly specify that a condition should hold some time in the future, always in the future unless some other conditions holds, or never in the future. Formal assertions in this form of temporal logic can be associated with domain level concepts, and thereby one can make inferences regarding state. It is unclear what a complete requirements model in KAOS looks like and what kind of analysis and testing tasks can be accomplished. van Lamsweerde reports however that that KAOS has been used in at least 11 industrial projects.

**Clarity:** The diagrammatic portion of the requirements notation is fairly readable, although it is unclear what each drawing means. The temporal logic portions are difficult to read for any stakeholder and require particular training. The use of temporal logic and diagrammatic syntax and semantics reduces the level of ambiguities.

**Testability:** KAOS does not explicitly support any testing activities. However, if test cases were produced from such requirements, they would be of high quality because they would show that the system either

implements stakeholder goals or fail to implement stakeholder goals. KAOS supports definition of domain knowledge regarding actors, roles, and other entities. These domain concepts could be used for test data selection if KAOS were to be used in a testing activity. However, the manual work for any testing activity is fairly substantial, since there is no current support for any of the activities. With some effort it is possible that several of the testing activities could be automated to some extent.
**Ease:** It does not appear easy to create requirements using the KAOS framework. The approach requires substantial manual work to create requirements that are correctly formulated and in accordance with stakeholder needs. The requirements are highly connected, and every update would require changes throughout the model. There is no explicit support for maintenance activities. However, the explicit relations among goals and refinements of them, does provide some guidance during maintenance.
**Designability & Traceability.** KAOS provides some support for design activities. There seem to be some traceability links, particularly between high-level goals and low-level functional requirements.

| Category | Question | | Evaluation Score |
|---|---|---|---|
| Clarity | Understandability | | ⋆ |
| | Unambiguity | | ⋆⋆⋆ |
| Testability | Test case | Ease | ⋆ |
| | | Quality | ⋆⋆⋆⋆ |
| | Test data | Ease | ⋆ |
| | | Quality | ⋆⋆ |
| | Test coverage | Quality | ⋆⋆⋆⋆ |
| | Test oracle | Ease | ⋆ |
| | | Quality | ⋆⋆⋆ |
| Manipulability | Initial | | ⊘ |
| | Maintenance | | ⋆ |

Table 16: Evaluation of KAOS

### 3.11 i* (Actor-based)

The i* framework is an actor-based requirements engineering approach [58], and is the only approach in this category of requirements approaches. The actor-based approach is however closely related to RML and several goal-based requirements approaches. The main difference between i* and a purely goal-based approach is that i* focuses on the organizational actors and roles. Goals are an important component of the approach, but the goals are derived from the viewpoint of actors and their interactions.

In the i* framework, various types of agents are defined and model situations through a set of dependency links with the objective to achieve some goal or task. To get a better understanding of how the proposed system might be embedded in the organizational environment, the Strategic Dependency model focuses on the intentional relationships among organizational actors. The Strategic Dependency model provides one level of abstraction for describing organizational environments and their embedded information systems. Intentional elements (goals, tasks, resources, and softgoals) appear in the Strategic Rationale model not only as external dependencies, but also as internal elements linked by means-ends relationships and task-decompositions. The Strategic Rationale model thus provides a way of modeling stakeholder interests, and how they might be met, and the stakeholders evaluation of various alternatives with respect to their interests. Task-decomposition links provide a hierarchical description of intentional elements that make up a routine. The means-ends links in the Strategic Rationale provides understanding about why an actor would engage in some tasks, pursue a goal, need a resource, or want a softgoal. From the softgoals, one can tell why one alternative may be chosen over others. The i* models offer different kinds of analysis, in terms of ability, workability, viability and believability. The approach uses a knowledge base to represent early requirements knowledge [59]. This knowledge base can be searched based on rule properties. There is an underlying

formal representation and a graphical representation. There is tool support, OME2 and OME3, however the notation resembles object oriented programming and does not seem fitting for high- level stakeholders.
**Clarity:** The diagrammatic portions of models created using i* are fairly understandable. The notation does require training, and is not intuitively understandable to anyone. The underlying formal representation reduces the possibility for ambiguities, but does not eliminate them entirely. It is unclear how the diagrammatic models stay consistent with the mathematical models.
**Testability:** There is no explicit support for any testing activities. If test cases were created from the requirements, they would be of fairly high quality because they focus on stakeholder and organizational needs and could thereby verify or refute if such were met. i* uses knowledge representation that could be used to select test data, and the representation of knowledge could be useful for automating the task. However, all the necessary relations and representations are not currently there for any automation, so any testing activity would require a substantial amount of manual work at this point.
**Ease:** It does not appear easy to create i* models. The notation is complicated and it is unclear how diagrammatic and mathematical notations are related and maintained consistent. There is tool support for i*, but it seems that the tools go beyond just requirements, and the tools themselves would require a significant amount of training.
**Designability & Traceability.** i* provides some support for design activities, particularly when used in the software engineering process model TROPOS. There seem to be some traceability links, particularly between high-level goals and low-level functional requirements.

| Category | Question | | Evaluation Score |
|---|---|---|---|
| Clarity | Understandability | | ⋆ |
| | Unambiguity | | ⋆⋆ |
| Testability | Test case | Ease | ⋆ |
| | | Quality | ⋆⋆⋆⋆ |
| | Test data | Ease | ⋆⋆ |
| | | Quality | ⋆⋆⋆⋆ |
| | Test coverage | Quality | ⋆⋆⋆⋆ |
| | Test oracle | Ease | ⋆ |
| | | Quality | ⋆⋆⋆ |
| Manipulability | Initial | | ⊘ |
| | Maintenance | | ⋆ |

Table 17: Evaluation of i*

## 3.12 Problem Frames

The problem frames approach is centered around problems rather than solutions and is used to decompose problems to manageable pieces [30, 31]. Problem frames is the only representative from this type of requirements approach. It is, in fact, one of the few requirements approaches that is truly designed to model the problem space rather than a amalgamation of the problem and solution spaces.

Some of the main benefits of problem frames are the focus on the composite system rather than the system functionality and scalability which is reached by working on solutions to sub-problems instead of tackling a much larger problem all at once. Problem frames are similar to design patterns in that there exists a defined set of re-usable problem frames to which a requirements engineer can map every problem. The five problem frames defined are:

1. **Required behavior.** This frame is used when there is some part of the physical world whose behavior is to be controlled so that it satisfies certain conditions. The problem in need of a solution is to build a machine that will impose that control.

2. **Commanded behavior.** This frame is used when there is some part of the physical world whose behavior is to be controlled in accordance with commands issued by an operator. The problem in need of a solution is to build a machine that will accept the operator's commands and impose the control accordingly.
3. **Information display.** This frame is used when there is some part of the physical world about whose states and behavior certain information is continually needed. The problem is to build a machine that will obtain this information from the world and present it at the required place in the required form.
4. **Simple workpieces.** This frame is used when a tool is needed to allow a user to create and edit a certain class of computer-processable text or graphic objects, or similar structures, so that they can be subsequently copied, printed, analyzed or used in other ways. The problem is to build a machine that can act as this tool.
5. **Transformation.** This frame is used when there are some given computer-readable input files whose data must be transformed to give certain required output files. The problem is to build a machine that can perform this translation process.

In the process of problem decomposition a large general problem frame is broken down into parallel frames rather than hierarchical frames. The problem frames are far from interchangeable. The chosen frame must fit the problem. A good software development method prescribes a very specific problem frame and exploits its properties to the fullest.

There are many deficiencies of problem frames such as the notation does not allow detail regarding timing of events, it is not possible to describe connections to stakeholder goals, and it is difficult to determine any interference between decomposed sub-frames. In [34] the authors describe the translation of one small problem frame into an operational state machine. The problem frames themselves do not contain enough information to automate such a translation. Instead the problem frames are useful as a discussion framework to determine the details needed for a manual translation. It is however unclear whether problem frames in fact lend themselves to better discussion and translation.

**Clarity:** The problem frames present the decomposition of the problem space in neat and fairly understandable diagrams. One problem though is that the problem frames are so general that they do not provide much of the information that is actually needed to fully understand the problem. Also, the notation is fairly ambiguous. It is less ambiguous than natural language, but still require clarification in sub-sequent development activities.

**Testability:** Problem frames do not directly support testability. The structure of the problems, however, can be used to understand what to test. The inability of modeling domain information reduces the chances for using the approach for test data selection. Tests could be created to cover the different paths in the problem frame. These paths however are too high-level to be very useful for testing purposes.

**Ease:** The notation itself does not seem difficult to learn. It does seem tricky to learn how to decompose the problem space into different problem frames and relate these. Once the problem frames have been specified, relations among them could aid in maintenance activities.

**Designability & Traceability.** There is no explicit support for either design activities or traceability.

# 4 Analysis of Results

Table 19 shows the results of each RE approach with regard to each of the sub-categories. A simple comparison of the total number of stars of each RE approach shows that i* has the most at 23 stars followed by KAOS at 21 stars. This type of comparison is not saying much however since testability contains more sub-categories than clarity and manipulability. RE approaches that score well in testability thus inevitably perform better than approaches that perform exceedingly well in the other categories. Furthermore, the simple comparison does not account for the relative importance of sub-categories with regard to their overall category.

Table 20 on the other hand shows the normalized scores for each RE approach incorporating the weights of importance determined by the industry study. The results are also visualized by the bar graph in Figure 9.

| Category | Question | | Evaluation Score |
|---|---|---|---|
| Clarity | Understandability | | ⋆⋆ |
| | Unambiguity | | ⋆⋆ |
| Testability | Test case | Ease | ⋆ |
| | | Quality | ⋆⋆ |
| | Test data | Ease | ⊘ |
| | | Quality | ⊘ |
| | Test coverage | Quality | ⋆⋆⋆ |
| | Test oracle | Ease | ⋆ |
| | | Quality | ⋆ |
| Manipulability | Initial | | ⋆ |
| | Maintenance | | ⋆⋆ |

Table 18: Evaluation of Problem Frames

| | US | iUC | sUC | fUC | Sha | Ret | Sce | FD | RML | KAOS | i* | PF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Understandability | ⋆⋆⋆ | ⋆⋆⋆ | ⋆⋆⋆ | ⋆⋆ | ⋆⋆⋆ | ⋆⋆⋆ | ⋆⋆⋆ | ⊘ | ⊘ | ⋆ | ⋆ | ⋆⋆ |
| Unambiguity | ⊘ | ⋆ | ⋆⋆ | ⋆⋆ | ⊘ | ⋆⋆⋆ | ⋆ | ⋆⋆⋆ | ⋆⋆⋆ | ⋆⋆⋆ | ⋆⋆ | ⋆⋆ |
| Test case ease | ⋆ | ⋆ | ⋆⋆ | ⋆⋆⋆ | ⊘ | ⋆⋆⋆ | ⋆ | ⋆ | ⊘ | ⋆ | ⋆ | ⋆ |
| Test case quality | ⋆ | ⋆⋆ | ⋆⋆ | ⋆⋆ | ⊘ | ⋆⋆ | ⋆⋆ | ⋆⋆ | ⋆⋆ | ⋆⋆⋆⋆ | ⋆⋆⋆⋆ | ⋆⋆ |
| Test data ease | ⊘ | ⊘ | ⊘ | ⊘ | ⊘ | ⊘ | ⊘ | ⋆⋆ | ⊘ | ⋆ | ⋆⋆ | ⊘ |
| Test data quality | ⊘ | ⊘ | ⊘ | ⊘ | ⊘ | ⊘ | ⊘ | ⋆⋆ | ⋆⋆ | ⋆⋆ | ⋆⋆⋆⋆ | ⊘ |
| Test coverage | ⋆⋆ | ⋆⋆⋆ | ⋆⋆⋆ | ⋆⋆⋆ | ⋆ | ⋆⋆⋆ | ⋆⋆⋆ | ⋆⋆ | ⋆⋆⋆ | ⋆⋆⋆⋆ | ⋆⋆⋆⋆ | ⋆⋆⋆ |
| Test oracle ease | ⋆ | ⋆ | ⋆ | ⋆ | ⊘ | ⋆⋆⋆ | ⊘ | ⋆ | ⊘ | ⋆ | ⋆ | ⋆ |
| Test oracle quality | ⋆ | ⋆ | ⋆ | ⋆ | ⊘ | ⋆⋆ | ⊘ | ⋆⋆ | ⋆⋆ | ⋆⋆⋆ | ⋆⋆⋆ | ⋆⋆ |
| Initial manip. | ⋆⋆⋆⋆ | ⋆⋆⋆ | ⋆⋆ | ⋆⋆ | ⋆⋆⋆⋆ | ⊘ | ⋆⋆⋆⋆ | ⋆ | ⋆ | ⊘ | ⊘ | ⋆ |
| Maintenance manip. | ⋆ | ⋆ | ⋆⋆ | ⋆ | ⋆⋆ | ⊘ | ⋆⋆⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆⋆ |
| Total | 14 | 16 | 18 | 17 | 10 | 19 | 17 | 17 | 14 | 21 | 23 | 16 |

Table 19: Results of evaluating twelve RE approaches with regard to clarity, testability, and ease of artifact manipulation and their sub-categories.

The figure shows each approach's level of support in each of the three dimensions. The score in a dimension is calculated by a weighted and normalized sum, so that the relative importance of sub-categories is accounted for and so that a category with a larger number of distribution points is not favored.

It is noteworthy that several of the surveyed RE approaches score moderately or higher (0.5 corresponding to moderate in a normalized Likert item) in one or two categories, but none scored moderately or higher in all three categories. Since no RE approach in this survey scores moderately or higher in all three categories, it is difficult to declare a winner, i.e., a RE approach that is useful and usable.

| | US | iUC | sUC | fUC | Sha | Ret | Sce | FD | RML | KAOS | i* | PF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Clarity | 0.405 | 0.52 | 0.635 | 0.5 | 0.405 | 0.75 | 0.52 | 0.345 | 0.345 | 0.48 | 0.365 | 0.5 |
| Testability | 0.245 | 0.341 | 0.383 | 0.424 | 0.055 | 0.503 | 0.289 | 0.433 | 0.36 | 0.621 | 0.711 | 0.368 |
| Manipulability | 0.535 | 0.44 | 0.5 | 0.345 | 0.69 | 0 | 0.845 | 0.25 | 0.25 | 0.155 | 0.155 | 0.405 |

Table 20: Normalized results, evaluating the twelve RE approaches with regard to clarity, testability, and manipulability. The results incorporate the relative weights of importance of sub-categories within the three categories.

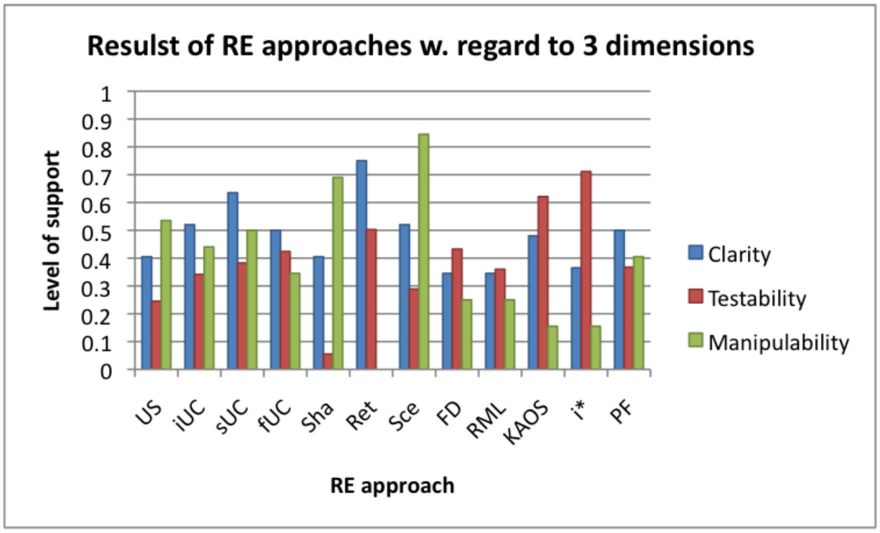


Figure 9: Results showing level of support for clarity, testability, and manipulability.

Another attempt to calculate an overall score is presented in Figure 10. The chart in the figure shows each RE approach's three scores combined in an overall normalized score. This data representation shows several interesting phenomena. First, it is clear that an overall winner still cannot be determined. The net difference between the highest scoring RE approach, scenarios, and the lowest scoring RE approach, RML, is only about 0.2 on a scale between 0 and 1. The difference is too low to be considered significant and could be explained by imprecision in data assignment among other things. It is interesting to compare the volatile peaks of the chart in Figure 9 and the small difference among the bars in the chart in Figure 10. Second, the error bars represent one standard deviation. The size of a RE approach's error bars is thus an indication of the variance between the scores in the three categories. Large error bars show that the RE approach scores very well and very poorly in two different categories. Third, two of the RE approaches, scenarios and semi-formal use cases, have an overall score that is moderate or higher (0.5).

Since the combined scores of the RE approaches are fairly inadequate to determine a winner among the

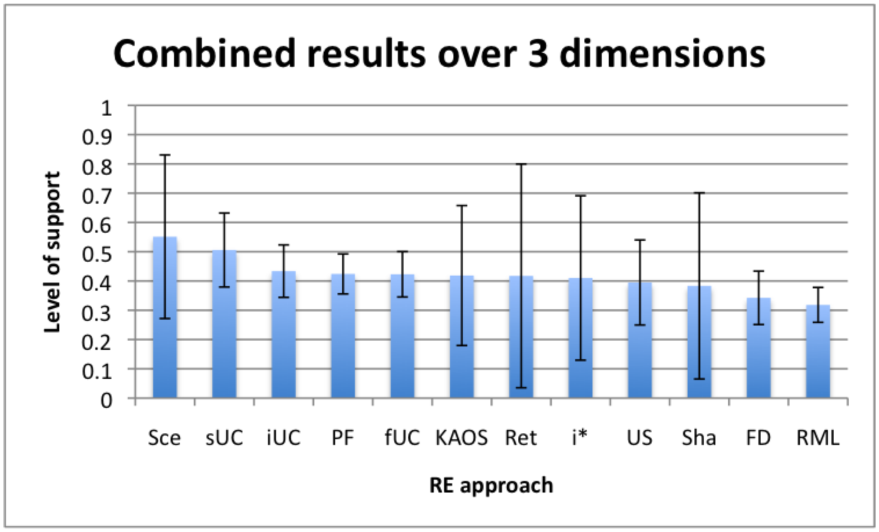


Figure 10: Combining scores and normalizing over the three categories.

RE approaches, the following paragraphs will show the scores for each category. Figure 11 compares each RE approach's score with regard to only clarity. The difference between the highest scoring approach, RETNA, and the lowest scoring approaches, FD and RML, is about 0.4, which is more significant than the difference shown in Figure 10. Another interesting observation is that the RE approaches appear to cluster into three groups depending on their scores. The highest scoring group contains RETNA and semi-formal use cases. The middle group contains informal use cases, formal use cases, scenarios, KAOS, and problem frames. The lowest scoring group consists of user stories, shall statements, functional documents, requirements modeling language, and i*.

It is also noteworthy that all the approaches in the two highest scoring groups score moderately or higher with regard to clarity. The average support for clarity by the RE approaches is 0.48. This could indicate that clarity as a characteristic is fairly well supported by current RE approaches. The two highest scoring approaches are interesting because they both combine informal natural language descriptions as well as formalizations. The approaches in the other two groups are more diverse in the sense that they do not share any particular common feature.

Figure 12 shows the results of each RE approach with regard to testability. The difference between the highest scoring approach, i*, and the lowest scoring approach, shall statements, is quite large, approximately 0.7. The RE approaches cannot be grouped into a small number of similarly scoring groups. Instead, their scores follow an almost linear decline from top to bottom. This indicates that there is quite a lot of diversity among current RE approaches with regard to their testability. It is also noticeable that only the three highest scoring approaches, i*, KAOS, and RETNA, score moderately or higher. The average level of support offered by current approaches is 0.39.

Figure 13 shows the results for each RE approach with regard to manipulability. The difference between the highest scoring RE approach, scenarios, and the lowest scoring approach, RETNA, is approximately 0.85. The scores are almost linearly decreasing and only four of the approaches score moderately or higher with regard to manipulability. It seems that less formal approaches generally score better than formal approaches in this category.

The following paragraphs describe the results of each sub-category. Figure 14 shows the results for understandability and unambiguity. It is clear that understandability received slightly better results overall. The difference between the best scoring approaches in understandability, user stories, semi-formal use cases, shall statements, scenarios, RETNA, and informal use cases, and the worst scoring approaches, requirements

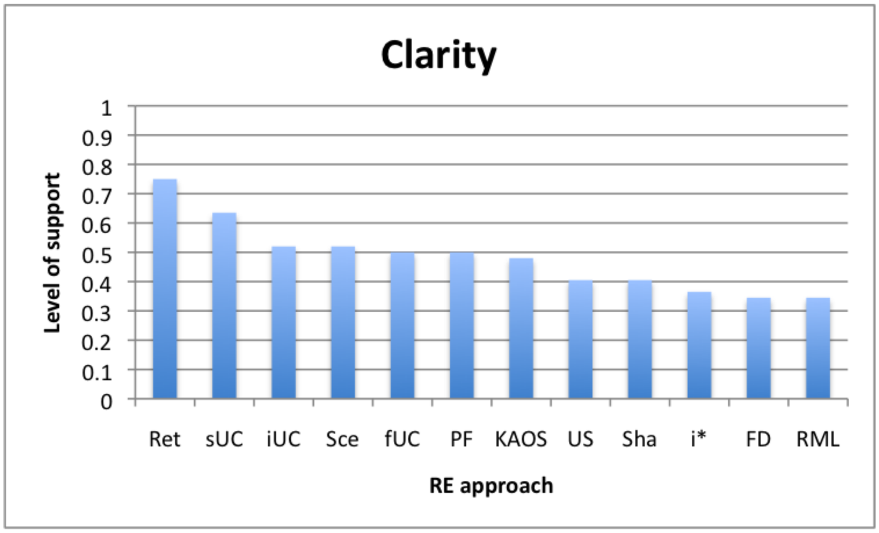


Figure 11: Clarity results for each RE approach.

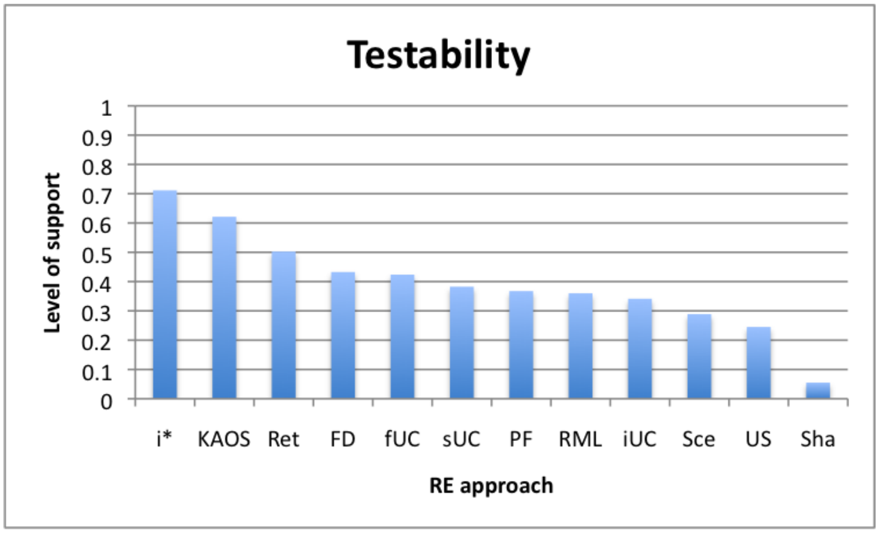


Figure 12: Testability results for each RE approach.

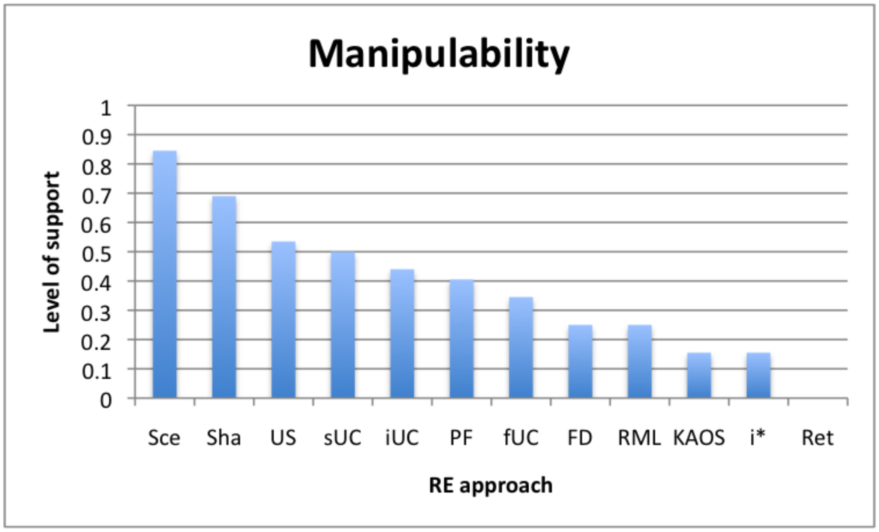


Figure 13: Testability results for each RE approach.

modeling language, and functional documents, is approximately 0.75 indicating a wide spread. In fact the two lowest scoring approaches received zero in understandability. The difference between the best scoring approaches in unambiguity, requirements modeling language, RETNA, KAOS, and functional documents, is also approximately 0.75 with the two lowest scoring approaches, user stories, and shall statements, receiving scores of zero.

It is interesting to note that in both cases two thirds of the RE approaches scored moderately or higher. It is however, also interesting to notice that, in general, approaches that score high in understandability score low in unambiguity. Figure 15 further explores the relation between data received for understandability and data received for unambiguity. The correlation graph shows that there is a negative correlation between the data obtained for the two sub-categories. Since correlation is calculated using data obtained from the twelve surveyed approaches, the relation is not causal but merely illustrates that in general existing RE approaches show a tradeoff relation between understandability and unambiguity.

Figure 16 illustrates each RE approach's score on each of the four testing activities: test case creation, test data selection, test coverage criteria definition, and test oracle creation. It is apparent that, overall, the quality of test case coverage of the different RE specifications is fairly good. It is also rather noticeable that the majority of RE approaches offer little support for test data selection. This is quite surprising when comparing to specification-based testing, in which test data is often selected from the specification. Test case creation and test oracle creation are not supported sufficiently either.

Figure 17 illustrates the average support by current RE approaches for each of the testing activities. The averages of test case creation, test data selection, test coverage criteria definition, and test oracles creation, are 0.41, 0.16, 0.71, and 0.30 respectively. It is clear that there is a large difference between quality of test coverage (0.71) and support for test data selection (0.16). This could indicate room for improvements in low-scoring areas, particularly because the weights of importance received from the questionnaire indicate that the different activities are perceived approximately equally important. Had the weights been different, it could indicate that RE approaches are rightfully not providing support for particular testing activities.

Figure 18 shows the results for each RE approach with regard to their initial and maintenance manipulability. On average, RE approaches scored higher in initial manipulability than in maintenance manipulability, indicating that current approaches do not offer sufficient support for maintainability. The top chart in Figure 18 shows that there is a fairly large difference between the approaches that score moderately or higher and the low scoring approaches; it shows that half of the approaches are in the high scoring group and the

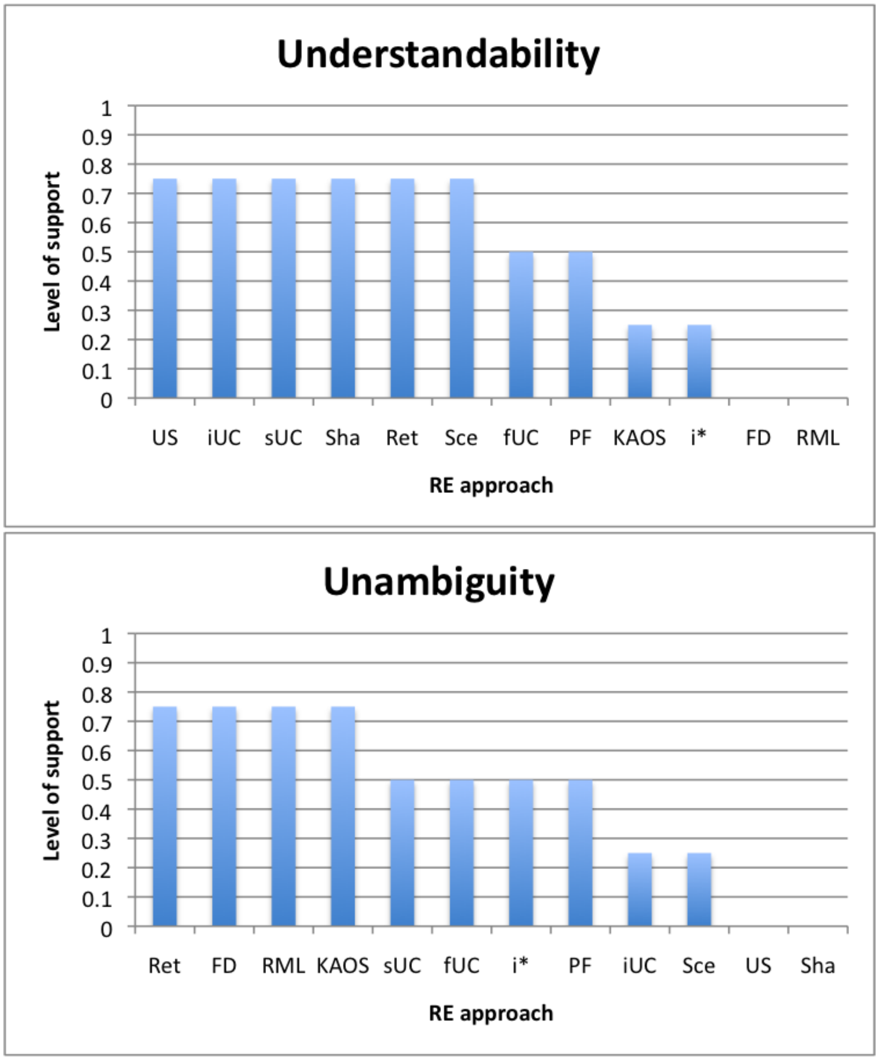


Figure 14: Understandability and unambiguity results for each RE approach.

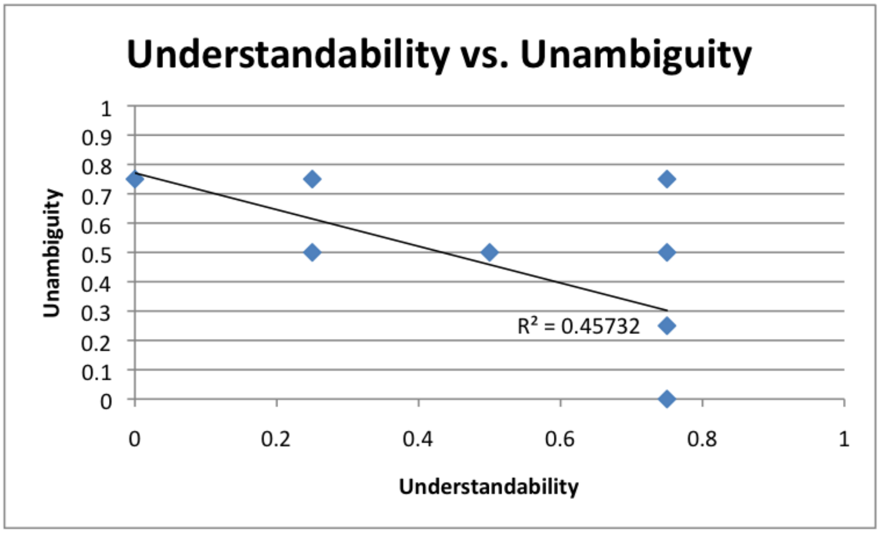


Figure 15: Correlation between understandability and unambiguity.

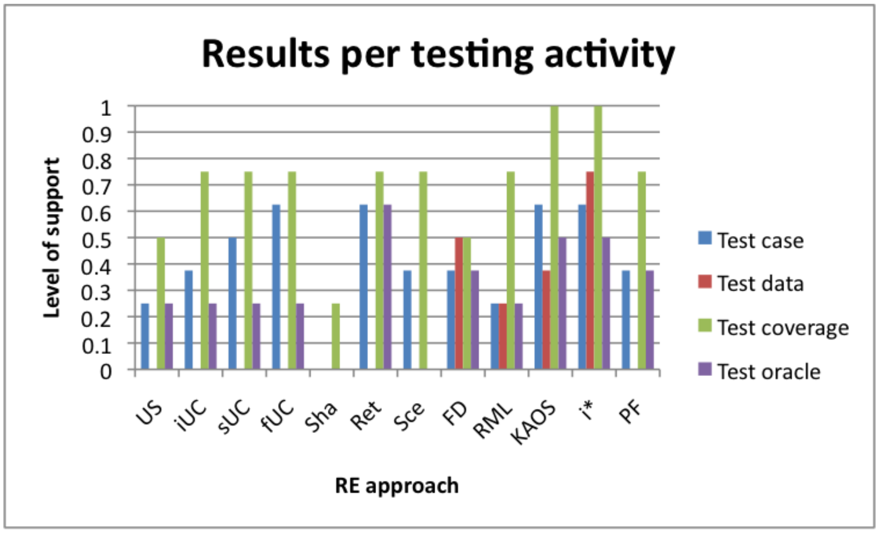


Figure 16: RE approaches evaluated per testing activity.

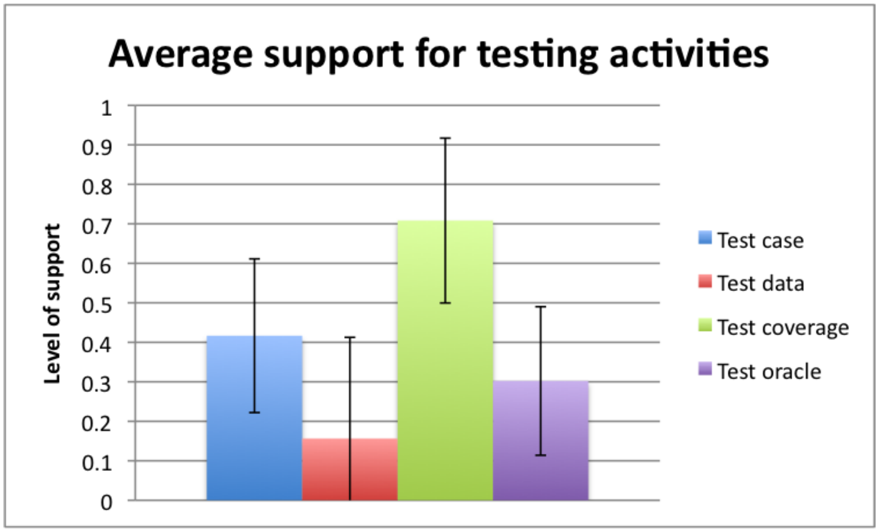


Figure 17: Average support for testing activities by current RE approaches.

other half are in the low scoring group. It is also interesting to note that the difference between the highest scoring approaches, user stories, shall statements, and scenarios, and the lowest scoring approaches, RETNA, KAOS, and i*, is 1.0.

The bottom chart in Figure 18 shows that only four of the twelve approaches score moderately or higher with regard to maintenance manipulability. There is a fairly large difference between the highest scoring approach, scenarios, and the next highest scoring approaches, semi-formal use cases, shall statements, and problem frames. The difference between the highest scoring approach, scenarios, and the lowest scoring approach, RETNA, is approximately 0.75, showing that there is a large difference among various approaches. The top chart in Figure 18 also shows that four different groups with similar scores exist. The first group consists of only the highest scoring approach, scenarios. The second highest scoring group consists of semi-formal use cases, shall statements, and problem frames. The third highest scoring group is the largest group and consists of user stories, requirements modeling language, KAOS, informal use cases, i*, formal use cases, and functional documents. The lowest scoring group consists of only the one approach, RETNA.

It is also noticeable that there is no tradeoff relation among approaches' with regard to initial manipulability and maintenance manipulability. Figure 19 shows that the correlation is in fact slightly positive, indicating that approaches that score well in one sub-category also score well in the other sub-category. The correlation is however too small to point out any trends among the surveyed approaches.

The following sections will provide data correlations between the different dimensions. Figure 20 shows the correlation between clarity and testability. The correlation is approximately 0.18 indicating a very small positive correlation. The correlation is too small however to show any trends among the data points. Figure 21 instead plots the correlation between unambiguity and testability and shows a much higher positive correlation of 0.68. This indicates that in general the more unambiguous a RE approach is the more testable it is. The correlation does not imply a causal relation in general, but merely shows trends among the data collected here. The negative correlation between understandability and unambiguity in Figure 15 and the results in Figure 20 and Figure 21 indicate that the surveyed RE approaches show a negative correlation, or tradeoff relation, between understandability and testability. It is also important to note that there are outliers in the data that show that the correlations are not necessary but common.

Figure 22 shows the correlation between clarity and manipulability. The correlation is -0.27, which is too small to indicate any kind of trends in the data. Since the correlation between clarity and manipulability does not provide much information regarding trends among the surveyed RE approaches, Figure 23 instead

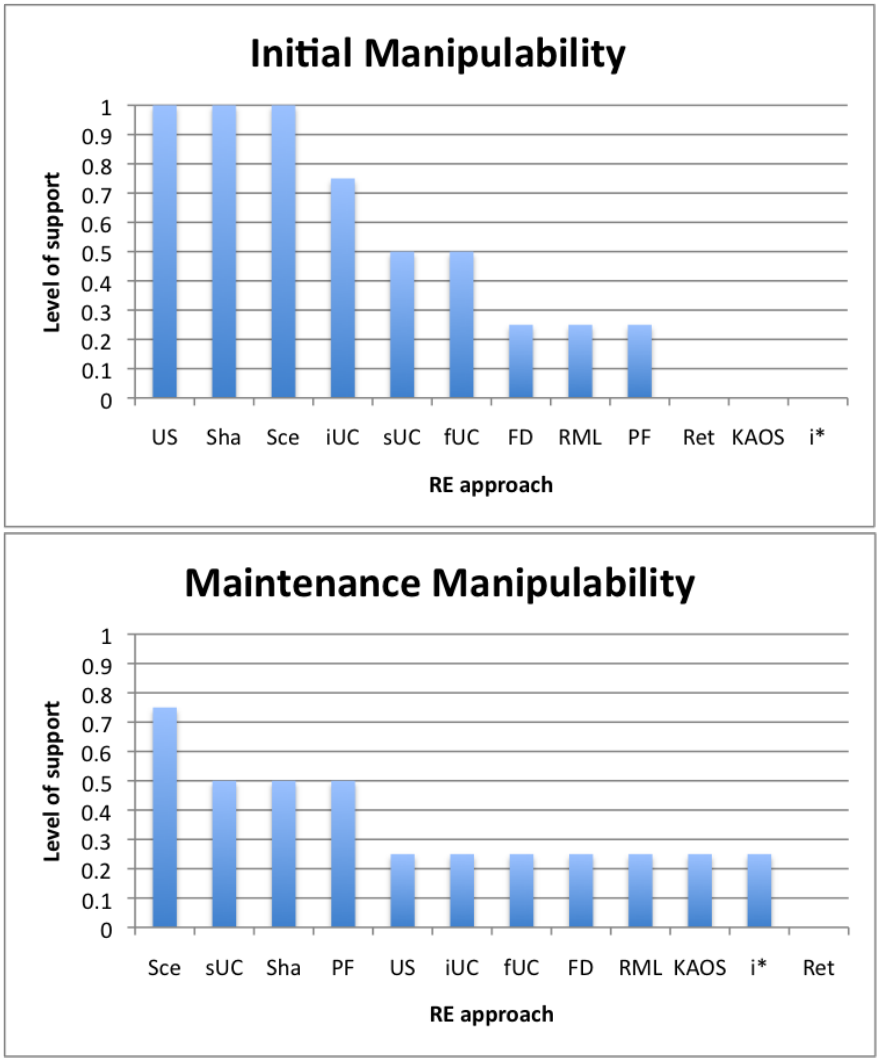


Figure 18: Results of initial and maintenance manipulability for the RE approaches.

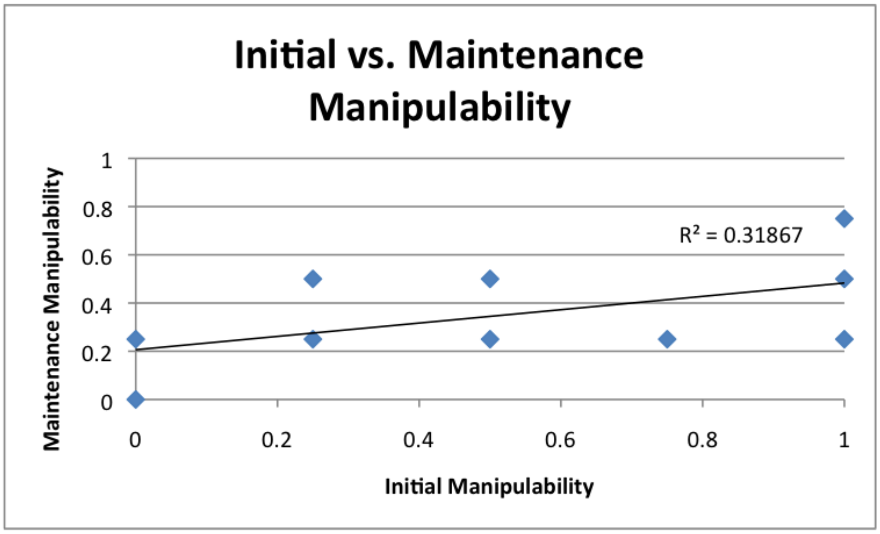


Figure 19: Correlation between initial ease of artifact manipulation and ease of maintenance of the requirements artifact.

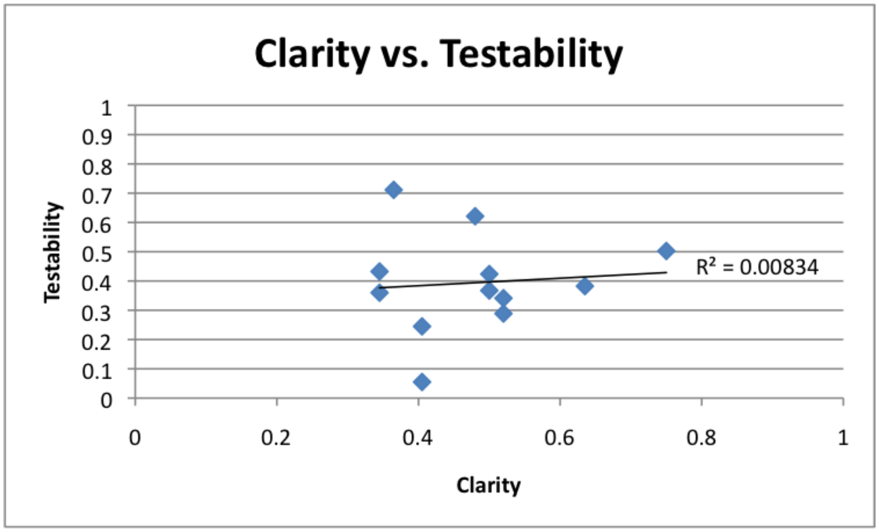


Figure 20: Correlation between clarity and testability.

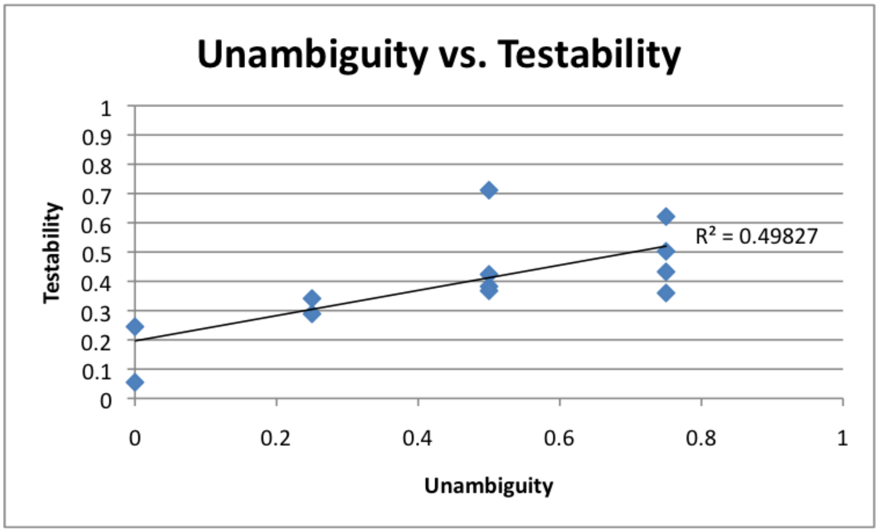


Figure 21: Correlation between unambiguity and testability.

investigates the correlation between a sub-category of clarity, namely understandability, and ease of artifact manipulation. The correlation is 0.56, a positive medium-strong correlation, which indicates that RE approaches that are understandable are more often than not also easy to create and maintain. The negative correlation between understandability and unambiguity and the results in Figure 22 also indicate that there is a negative correlation, or a tradeoff relation, between unambiguity and manipulability. This implies that RE approaches that are unambiguous are more difficult to use. Outliers in the data also show that this is a dominating trend, but not a necessary condition.

Figure 24 shows the correlation between manipulability and testability. The correlation is -0.78, which is a strong negative correlation. This indicates that there is a tradeoff relation between current approaches' level of manipulability and level of testability. It is also noteworthy that there are no clear outliers that indicate the existence of an approach that is both easy to use and testable. Since there is no notable tradeoff relation between initial manipulability and maintenance manipulability there is no positive relation between any sub-category of manipulability and testability.

A short summary of the results so far shows that there is no RE approach that is highly useful and usable. The results also show a large variation among the approaches evaluated with regard to the different categories. This large variation indicated that it is not useful to evaluate the surveyed approaches by only looking at their final combined scores. Exploring trends in the data show that there is a tradeoff relation between the two sub-categories of clarity, i.e., between understandability and unambiguity. It is also apparent that understandability and manipulability have a positive correlation, and that unambiguity and testability have a positive correlation. There is also a tradeoff relation between manipulability and testability. These trends are interesting because they describe the current state of RE approaches, i.e., no RE approach fulfills all these categories well at the same time. It is however important to distinguish these trends from causality. The results do not in any way dictate that tradeoff relations or positive relations among categories and sub-categories are unavoidable. Outliers in the data, such as RETNA's high score in both understandability and unambiguity, support this point.

The next paragraphs discuss how the evaluation framework and survey results can be used to select RE approaches. Figure 25 groups each RE approach with regard to moderate or higher support for each of the three dimensions. Each left branching specifies moderate or higher support, and each right branching specifies moderate or lower support. The ultimate group, the lowest left corner, is empty since no approach is fully usable. Since there is no optimal RE approach, the grouping scheme can be used to select a RE

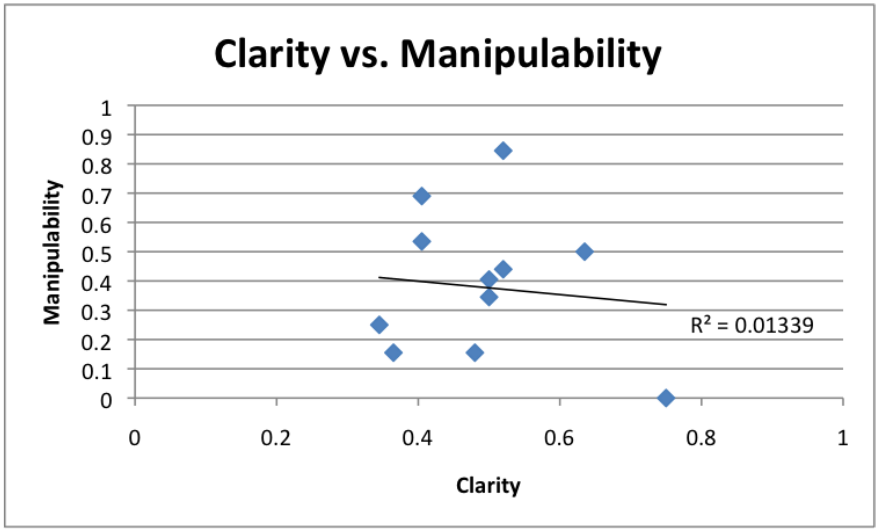


Figure 22: Correlation between clarity and manipulability.

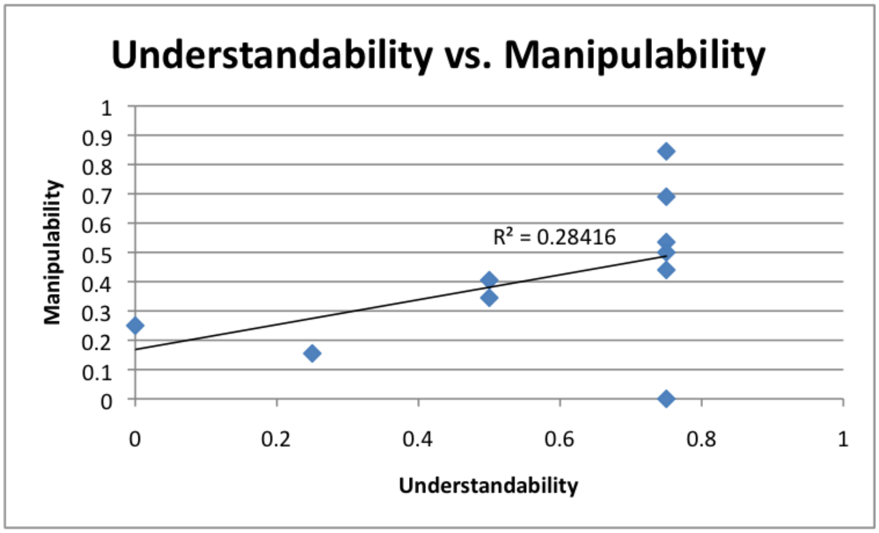


Figure 23: Correlation between understandability and manipulability.

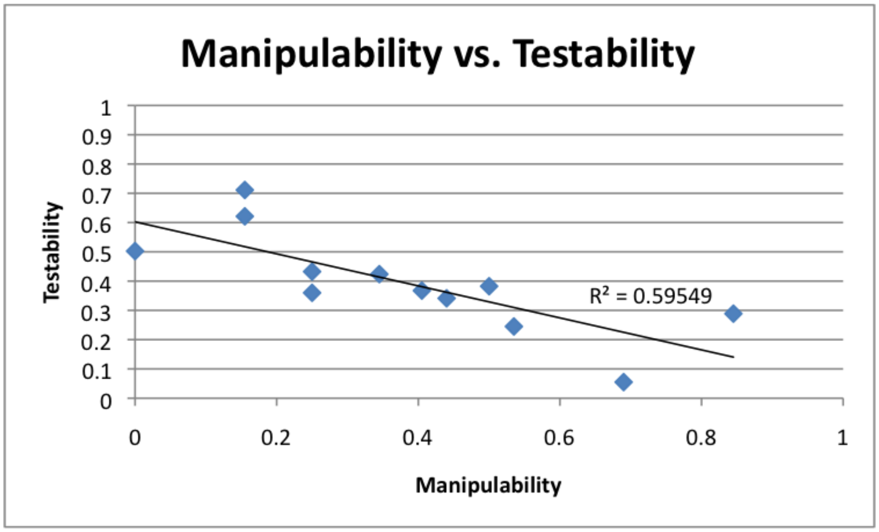


Figure 24: Correlation between manipulability and testability.

approach based on tradeoff analysis. For example, if one chooses high clarity and high manipulability, but is willing to sacrifice testability, the two RE approaches available are scenarios and semi-formal use cases. Another way to create a grouping scheme would take into account the current data and instead of a binary branching scheme, branch based on groupings visible in the data. However, for the sake of space, this survey only depicts a binary split. Also worth noting is that borderline cases are included in both left and right groups to increase the possibility of finding an appropriate RE approach. The underlying idea in the grouping in Figure 25 is to assign weights to the three dimensions. Rather than assigning concrete numerical weights, the relative importance is considered.

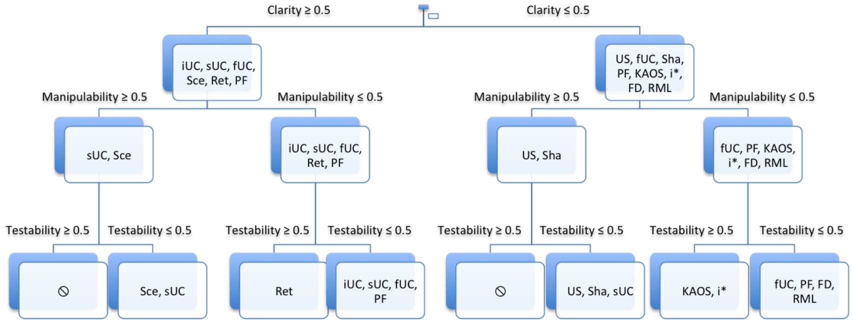


Figure 25: Grouping of RE approaches with regard to providing moderately or higher support for each of the three dimensions: clarity, testability, and ease of artifact manipulation.

As part of the questionnaire, we asked participants to rate several RE approaches. Their ratings are summarized in Figure 26, and it is evident that using industry to rate RE approaches does not result in any clear winners or losers either.

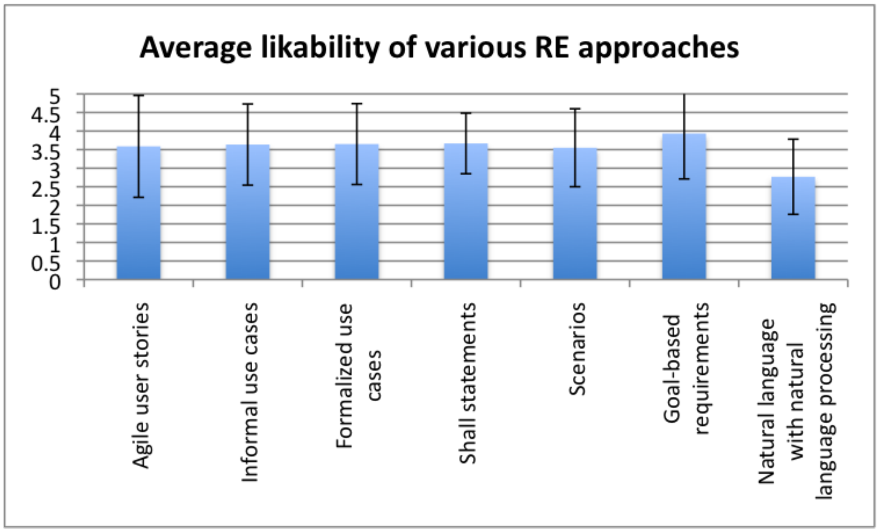


Figure 26: Ratings of RE approaches obtained through questionnaire.

# 5 Conclusions and Future Work

An indirect conclusion from this study is that requirements engineering publications typically do not contain enough information or detail to reproduce their results. Reproduction of results is a key concept in scientific research, and it is undesirable to publish RE modeling schemes without properly defining enough information, such as syntax, semantics, form, and process, so that other researchers or practitioners can use the language to actually create requirements. Not only is this an undesirable research practice, but it also has severe consequences with regard to the usability and usefulness of RE approaches since their use cannot be mediated through research publications.

Direct results from the study show that there are no RE approaches that exhibit the characteristics of usability and usefulness. The data also showed some interesting positive and negative correlations with regard to the different characteristics, indicating that it is difficult to achieve all characteristics at once, and perhaps future RE approaches should focus on achieving some optimized balance among them.

One of the most surprising observations is the lack of support for test data selection. The support was considerably lower than for any other testing activity as well as any other sub-category. The expectation was that there would be support for test data selection, but not for oracle creation, and the results showed the opposite.

One of the major contributions of this survey is that through the framework and evaluation of current approaches we can identify several areas in need of improvement that current research has not yet addressed satisfactorily. It is clear that achieving all characteristics of usability and usefulness is a very difficult goal, but it is also clear that it is an important goal. It is our goal to design a requirements approach that addresses all three dimensions of usability adequately.

The survey results shows that there are some RE approaches that fulfill certain desirable characteristics well while lacking others, and that there are other RE approaches that score averagely over all the three categories. One possible starting point for defining a usable and useful RE approach is to enhance an existing approach to gain a more optimized balance between the desirable characteristics.

For example RETNA scored well in clarity, fairly well in testability, but very poorly in ease of artifact manipulation. One route would be to work on improving that aspect of natural language processing. Another route would be to take a requirements approach that is fairly clear and easy to use, such as scenarios, semi-formal use cases, or informal use cases, and improve their testability facet.

We will base our work on scenarios. The evaluation results show that scenarios exhibit high clarity and manipulability, and our proposed work will include incrementally adding more testability while continuously verifying the conservation of clarity and manipulability. One reason to start with scenario is that they are narrative, or story-based. Stories are ubiquitous and an integral part of human life. Storytelling is one of the most common techniques for sharing information and has been used long before the invention of the written language [39]. Storytelling is also a cornerstone in teaching from kindergarten to higher education [4, 33, 39, 59] and more recently, storytelling has successfully been used in computer science education [20, 37]. Alice [21], Storytelling Alice [38], and Scratch [2, 42] are interactive programming environments that reduce the complexity of details that the novice programmer must overcome, and visualize ob jects in a meaningful context. Interestingly, stories have also long been in use in software development, for example in the form of use cases, user stories, interface design scenarios [41, 43], and in our previous work on requirements scenarios [3, 66, 68]. Another reason to start with scenarios rather than semi-formal use cases is that the approach already follows some of the practical guidelines that were derived from the industry study. One reason for not choosing one of the more formal modeling schemes, such as formal use cases, is that it is typically more difficult to modify a strictly dened approach than adding constraints to a less strict approach.

An improvement in testability is undeniably associated with formalizations and data or domain modeling. Future research will involve adding formalization and domain modeling to a fairly informal narrative requirements approach, such as use cases, without affecting the level of clarity and ease of artifact manipulation too negatively.

One route we intend to explore is to conceal formalisms under a seemingly informal editor for use cases. The editor will use similar interaction concepts to Alice, the visual programming environment, and allow users to drag and drop use case events and domain concepts into control structures to form a use case. The use case should internally be represented in a form that makes it amenable to support testing activities. The editor should also allow a debugging mode, similar to the Eclipse debugger, so that use cases can be stepped through and evaluated for correctness.

# 6 References

## References

[1] Specification and description language (SDL). ITU-T Recommendation Z.100, International Telecommunications Union, November 1999.

[2] Thomas A. Alspaugh, Susan Elliott Sim, Kristina Winbladh, Mamadou Diallo, Hadar Ziv, and Debra J. Richardson. Clarity for stakeholders: Empirical evaluation of scenarioml, use cases, and sequence diagrams. *Workshops on Comparative Evaluation in Requirements Engineering (CERE'07)*, pages 1–10, 2007.

[3] X. Alvarez, G. Dombiak, and M. Prieto. Use-cases, interaction diagrams, hypermedia and visualization. In *Workshop on Requirements Engineering: Use Cases and More*, 1995.

[4] Vincenzo Ambriola and Vincenzo Gervasi. Processing natural language requirements. In *International Conference on Automated Software Engineering (ASE'97)*, pages 36–45, 1997.

[5] Annie Antón and Colin Potts. A representational framework for scenarios of systems use. *Requirements Engineering Journal*, 3(3–4):219–241, 1999.

[6] Annie I. Antn and Collin Potts. The use of goals to surface requirements for evolving systems. In *International Conference on Software Engineering (ICSE'98)*, 1998.

[7] J. Aranda, S. M. Easterbrook, and G. Wilson. Requirements in the wild: How small companies do it. In *IEEE International Requirements Engineering Conference RE'07*, pages 39–48, 2007.

[8] Victor R. Basili. Software development: a paradigm for the future. In *Computer Software and Applications Conference COMPSAC'89*, pages 471–485, Sep 1989.

[9] Hubert Baumeister. Combining formal specification with test driven development. *XP/Agile Universe, Lecture Notes in Computer Science*, 2418:174–184, 2002.

[10] Ravishankar Boddu, Lan Guo, Supratik Mukhopadhyay, and Bojan Cukic. RETNA: From requirements to testing in a natural way. In *IEEE International Requirements Engineering Conference (RE'04)*, pages 262–271, 2004.

[11] F. Bordeleau and D. Cameron. On the relationship between use case maps and message sequence charts. In *Workshop of the SDL Forum Society on SDL and MSC (SAM'00)*, 2000.

[12] Alexander Borgida, Sol Greenspan, and John Mylopoulos. Knowledge representation as the basis for requirements specifications. *Computer*, 18(4):82–91, April 1985.

[13] Alexander Borgida, John Mylopoulos, and Raymond Reiter. ...and nothing else changes: the frame problem in procedure specifications. In *Proceedings of the 15th international Conference on Software Engineering (ICSE'93)*, pages 303–314, 1993.

[14] Jaelson Castro, Manuel Kolp, and John Mylopoulos. Towards requirements-driven information systems engineering: the *Tropos* project. *Information Systems*, 27(6):365–389, 2002.

[15] A. Cockburn. *Writing effective use cases.* Addison-Wesley Boston, 2001.

[16] A. Cockburn. Use cases, ten years later. *STQE Magazine*, April 2002.

[17] M. Cohn. *User Stories Applied: For Agile Software Development.* Addison-Wesley Professional, 2004.

[18] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1–2):3–50, 1993.

[19] Nicolás D'Ippolito, Dario Fischbein, Marsha Chechik, and Sebastián Uchitel. MTSA: The modal transition system analyser. In *Automated Software Engineering Conference (ASE'08)*, pages 475–476. IEEE, 2008.

[20] A. Eberlein and J.C.S. do Prado Leite. Agile requirements definition: A view from requirements engineering. In *Workshop on Time-Constrained Requirements Engineering TCRE'02*, Sep 2002.

[21] Donald Firesmith. Specifying good requirements. *Journal of Object Technology*, 2(4):77–87, 2003. `http://www/jot.fm/issues/issue_2003_07/column7`.

[22] R.E. Gallardo-Valencia, V. Olivera, and S.E. Sim. Are use cases beneficial for developers using agile requirements? In *International Workshop on Comparative Evaluation in Requirements Engineering (CERE'07)*, pages 11–22, 2007.

[23] Angelo Gargantini and Constance Heitmeyer. Using model checking to generate tests from requirements specifications. In Oscar Nierstrasz and Michel Lemoine, editors, *European Software Engineering Conference / Foundations of Software Engineering ESEC/FSE '99*, volume 1687 of *Lecture Notes in Computer Science*, pages 146–162. Springer-Verlag / ACM Press, 1999.

[24] Martin Glinz. An integrated formal model of scenarios based on statecharts. In *European Software Engineering Conference (ESEC'95)*, pages 254–271, 1995.

[25] Sol Greenspan, John Mylopoulos, and Alexander Borgida. On formal requirements modeling languages: RML revisited. In *International Conference on Software Engineering (ICSE'94)*, pages 135–147, 1994.

[26] W. Grieskamp, M. Lepper, W. Schulte, and N. Tillmann. Testable use cases in the abstract state machine language. In *Asia-Pacific Conference on Quality Software (APAQS'01)*, 2001.

[27] Mats Heimdahl. Safety and software intensive systems: Challenges old and new. In *International Conference on Software Engineering (ICSE'07)*, pages 137–152, 2007.

[28] Jim Heuman. Generating test cases from use cases. *Rational Software. The Rational Edge e-zine for the rational community*, 2001. `http://www.ibm.com/developerworks/rational/library/content/RationalEdge/jun01/GeneratingTestCasesFromUseCasesJune01.pdf`.

[29] R. Hurlbut. *Managing Domain Architecture Evolution Through Adaptive Use Case and Business Rule Models.* Phd, Illinois Institute of Technology, 1998.

[30] Michael Jackson. Problems, methods and specialisation. *Software Engineering Journal*, 9(6):249–255, November 1994.

[31] Michael Jackson. *Problem Frames - Analyzing and Structuring Software Development Problems.* Addison-Wesley, 2001.

[32] I. Jacobson, M. Griss, and P. Jonsson. Software reuse – architecture, process and organization for business success. *New York: ACM Press*, 1997.

[33] Ivar Jacobson. Formalizing use-case modeling. *Journal of Object Oriented Programming (JOOP)*, 8(3):10–14, 1995.

[34] Robin Laney, Leonor Barroca, Michael Jackson, and Bashar Nuseibeh. Composing requirements using problem frames. In *international Requirements Engineering Conference (RE'04)*, pages 122–131, 2004.

[35] Rensis Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 140:1–55, 1932.

[36] Susan Lilly. Use case pitfalls: top 10 problems from real projects using use cases. *Technology of Object-Oriented Languages and Systems (TOOLS 30)*, pages 174–183, August 1999.

[37] V.M. Mendoza-Grado. Formal verification of use cases. In *Workshop on Requirements Engineering: Use Cases and More*, 1995.

[38] Gerard Meszzaros, Shaun M. Smith, and Jennitta Andrea. The test automation manifesto. *XP/Agile Universe, Lecture Notes in Computer Science*, 2753:73–81, 2003.

[39] Luisa Mich. NL-OOPS: From natural language to object oriented requirements using the natural language processing system LOLITA. *Journal of Natural Language Engineering*, 2(2), 1996.

[40] Andrew Miga, Daniel Amyot, Francis Bordeleau, Donald Cameron, and Murray Woodside. Deriving message sequence charts from use case maps scenario specifications. In *Maps Scenario Specifications. Tenth SDL Forum (SDL'01)*, pages 268–287. Springer, 2001.

[41] Steven P. Miller, Alan C. Tribble, Michael W. Whalen, and Mats P. E. Heimdahl. Proving the shalls: Early validation of requirements through formal methods. *International Journal on Software Tools for Technology Transfer*, 8(4):303–319, 2006.

[42] Ian Mitchell and Hugues Lecoeuche. On an improved approach to the elicitation of O-O state machines by use-case. *Journal of Object Oriented Programming JOOP*, 9:52–55, 1997.

[43] J. Mylopoulos, L. Chung, and B. Nixon. Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Transactions on Software Engineering*, 18(6):482–497, 1992.

[44] Clementine Nebut, Franck Fleury, Yves Le Traon, and Jean-Marc Jezequel. Automatic test generation: A use case driven approach. *IEEE)Transactions on Software Engineering*, 32(3):140–155, March 2006.

[45] M. Newman. Software errors cost u.s. economy $59.5 billion annually: NIST assesses technical needs of industry to improve software-testing. *National Institute of Science and Technology (NIST)*, 2002.

[46] K. Orr. Agile requirements: opportunity or oxymoron. *IEEE Software*, 21:71–73, 2004.

[47] David Lorge Parnas and Jan Madey. Functional documents for computer systems. *Science of Computer Programming*, 25(1):41–61, 1995.

[48] Colin Potts and Kenji Takahashi. An active hypertext model for system requirements. In *International Workshop on Software Specification and Design (IWSSD'93)*, 1993.

[49] Colin Potts, Kenji Takahashi, and Annie Antón. Inquiry-based requirements analysis. *IEEE Software*, 11(2):21–32, Mar 1994.

[50] Michael Puleio. How not to do agile testing. *IEEE Computer Society, AGILE*, pages 305–314, 2006.

[51] Johannes Ryser and Martin Glinz. A scenario-based approach to validating and testing software systems using statecharts. In *International Conference on Software and Systems Engineering and their Applications (CSSEA'99)*, 1999.

[52] Alistair Sutcliffe. Scenario-based requirements analysis. *Requirements Engineering*, 3(1):48–65, 1998.

[53] Sebastian Uchitel and Jeff Kramer. A workbench for synthesising behaviour models from scenarios. In *International Conference on Software Engineering (ICSE'01)*, pages 188–197, May 2001.

[54] A. van Lamsweerde, R. Darimont, and Ph. Massonet. Goal-directed elaboration of requirements for a meeting scheduler: Problems and lessons learnt. In *International Symposium on Requirements Engineering (RE'95)*, pages 194–203, 1995.

[55] Axel van Lamsweerde. Goal-oriented requirements engineering: a guided tour. In *International Symposium on Requirements Engineering (ICRE'01)*, pages 249–262, 2001.

[56] Bart van Rompaey and Serge Demeyer. Exploring the composition of unit test suites. Technical Report UA TR2007-01 arXiv:0711.0607v1, 2007. `http://arxiv.org/abs/0711.0607v1`.

[57] Jon Whittle and Praveen K. Jayaraman. Generating hierarchical state machines from use case charts. In *IEEE International Requirements Engineering Conference (RE'06)*, pages 16–25, Washington, DC, USA, 2006. IEEE Computer Society.

[58] Eric Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE97)*, pages 226–235, 1997.

[59] Eric Yu and John Mylopoulos. From E-R to A-R – modelling strategic actor relationships for business process reengineering. *international Conference on the Entity-Relationship Approach, Lecture Notes In Computer Science*, 881:548–565, 1994.

# A Appendix A

# Usable Requirements

## 1. Welcome

Hi!

The objective of this survey is to explore desirable requirements engineering features.

Software requirements are needs and expectations for a software system. Various requirements approaches, or notations, exist to aid for example eliciting, specifying, and analyzing the requirements. These requirements approaches and notations attempt to support the various activities performed during requirements engineering and the software development life-cycle.

It is our goal to find out what kind of requirements approaches are being used, what works well, what does not work well, and what kind of requirements characteristics would be most sought after in practical real world projects. We are also interested in how using requirements to guide software testing can provide insight into how well a software system meets stakeholder needs. We intend to use your input to shape future research in requirements engineering to address current shortcomings in requirements approaches.

Please take your time and please answer each question.

Thank you!

## 2. Study Information Sheet

University of California, Irvine
Study Information Sheet
For

Evaluation of the Effectiveness of Different Requirements Approaches

Kristina Winbladh, Lead Researcher, 949.824.4043, awinblad@ics.uci.edu
Susan Sim, Faculty Sponsor, 949.824.2373, sesim@uci.edu
Hadar Ziv, 949.824.2901, ziv@ics.uci.edu

DONALD BREN SCHOOL OF INFORMATION AND COMPUTER SCIENCES

• You are being asked to participate in a research study about software requirements. We will use your input to learn about what you think are important and valuable features of a requirement modeling scheme.

• The research procedure consists of taking a survey questionnaire that will take approximately 15-20 minutes to complete. The survey is completely anonymous.

• The only foreseeable discomfort of the study is the invasion of your privacy. There are no direct benefits from participation in the study. However, this study may explain how to make requirements modeling schemes more appealing to software developers. In addition, findings of this study will be shared with other universities through research publications.

• Participation in this study is voluntary. There is no cost to you for participating. You may refuse to participate or discontinue your involvement at any time without penalty. You may choose to skip a question.

• No compensation will be provided for participating in this research.

• All research data collected will be stored securely and confidentially. All confidential data will be stored in Kristina Winbladh’s password protected computer in her locked office and her locked apartment.

• If you have any comments or questions regarding the conduct of this research or your rights as a research participant, you may contact the University of California, Irvine, Office of Research Administration by phone at (949) 824-6662.

○ Yes, I have read the Information Study Sheet.

## 3. Demographics

Please select one of the following choices

○ I am or have been working on the development of a software project

○ I am or have been a customer of a software project

## 4. Demographics

Please answer the following questions about your background.

Please rate your experience with the following software engineering tasks. Experience can be anything such as creation, manipulation, and/or evaluation.

| | None | Very little | Reasonable | Considerable | Expert |
|---|---|---|---|---|---|
| Software requirements | ○ | ○ | ○ | ○ | ○ |
| Software architecture/design | ○ | ○ | ○ | ○ | ○ |
| Software implementation | ○ | ○ | ○ | ○ | ○ |
| Software testing | ○ | ○ | ○ | ○ | ○ |
| Software project planning | ○ | ○ | ○ | ○ | ○ |
| Other | ○ | ○ | ○ | ○ | ○ |

Other (please specify)

Experience in software development

○ < 1 year ○ 1-5 years ○ 5-10 years ○ > 10 years

# Usable Requirements

Please specify from which perspective you are taking the survey.

○ Software engineering in the software industry

○ Software engineering research (faculty)

○ Software engineering research (graduate student)

○ Other (please specify)

Name of institution.

Education

○ No formal education ○ B.S. / B.A. ○ M.S. / M.A. ○ PhD

Other (please specify)

## 5. Demographics

What is your experience working with software projects?

○ < 1 year ○ 1-5 years ○ 5-10 years ○ > 10 years

Briefly describe your role in software projects.

Please rate your experience with the following software engineering tasks. Experience can be anything such as creation, manipulation, and/or evaluation.

| | None | Very little | Reasonable | Considerable | Expert |
|---|---|---|---|---|---|
| Software requirements | ○ | ○ | ○ | ○ | ○ |
| Software architecture/design | ○ | ○ | ○ | ○ | ○ |
| Software implementation | ○ | ○ | ○ | ○ | ○ |
| Software testing | ○ | ○ | ○ | ○ | ○ |
| Software project planning | ○ | ○ | ○ | ○ | ○ |
| Other | ○ | ○ | ○ | ○ | ○ |

Other (please specify)

Name of institution.

# Usable Requirements

Education

○ No formal education ○ B.S./B.A. ○ M.S./M.A. ○ PhD

Other (please specify)

## 6. Requirements Clarity

Two characteristics of requirements clarity are understandability and unambiguity.

Relevant terms:
Unambiguity - Requirements are described precisely and only one interpretation is possible.

Understandability - Requirements are readable and comprehensible for anyone participating in the project.

How important are the following characteristics of requirements approaches/notations?
(1 - not important, 5 - very important)

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Unambiguity | ○ | ○ | ○ | ○ | ○ |
| Understandability | ○ | ○ | ○ | ○ | ○ |

Under typical software development constraints, one might need to choose between unambiguity and understandability. Which would you consider more important?

○ Understandability ○ Unambiguity

Please add any additional comments or motivations.

## 7. Requirements Testability

Relevant terms:
Test case creation - Creation of conditions under which a system can be tested to provide evidence whether it fulfills a requirement.

Test data selection - Selection of abstract representations or concrete data assigned to test case conditions.

Test coverage criteria definition - Definition of criteria to measure how well testing covers a system under test.

Test oracle creation - Creation of a mechanism for determining whether testing produces expected results.

# Usable Requirements

How important is it that requirements be used to guide the the following testing activities?
(1 - not important, 5 - very important)

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Test data selection | ○ | ○ | ○ | ○ | ○ |
| Test case creation | ○ | ○ | ○ | ○ | ○ |
| Test coverage criteria definition | ○ | ○ | ○ | ○ | ○ |
| Test oracle creation | ○ | ○ | ○ | ○ | ○ |

Under typical software development constraints, how would you rank the relative importance of using requirements to guide these testing activities?

| | Most important | Second most important | Third most important |
|---|---|---|---|
| Test case creation | ○ | ○ | ○ |
| Test coverage criteria | ○ | ○ | ○ |
| Test oracle creation | ○ | ○ | ○ |
| Test data selection | ○ | ○ | ○ |

Please add any additional comments or motivations.

## 8. Testability Ease and Quality

Relevant terms:
Testing activities - Test case creation, test data selection, and test oracle creation.

Testing artifacts - Test cases, test data, test coverage criteria, and test oracles.

Ease - Whether a requirements approach is easy to use to guide testing activities.

Quality - Whether a requirements approach supports production of high quality testing artifacts.

How important are the following characteristics?
(1 - not important, 5 - very important)

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Quality | ○ | ○ | ○ | ○ | ○ |
| Ease | ○ | ○ | ○ | ○ | ○ |

Under typical software development constraints, one might need to choose between ease and quality. Which would you consider more important?

○ Ease ○ Quality

Please add any additional comments and motivations.

## 9. Requirements Manipulability

# Usable Requirements

Relevant terms:
Initial ease - Whether the requirements document is easy to create initially.

Maintenance ease - Whether the requirements document is easy to update when requirements change.

How important are the following characteristics of requirements approaches? (1 - not important, 5 - very important)

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Maintenance ease | ○ | ○ | ○ | ○ | ○ |
| Initial ease | ○ | ○ | ○ | ○ | ○ |

Under typical software development constraints, one might have to choose between initial ease and maintenance ease. Which would you consider more important?

○ Initial ease

○ Maintenance ease

Please add any additional comments and motivations.

## 10. Big Picture

Below is a list of activities that requirements approaches can support. Please select three alternatives that you think are the most important.

□ automatic test generation

□ cognitive support

□ model checking

□ other

□ support for informal conversations with customers and/or users

□ support for maintenance

□ support for transition to design

Other (please specify)

Below is a list of characteristics that requirements approaches can exhibit. Please select three alternatives that you think are the most important.

□ domain descriptions

□ formal notations

□ links between requirements

□ other

□ operational descriptions

□ property descriptions

□ traceability between requirements and other software artifacts

Other (please specify)

# Usable Requirements

What requirements approaches/notations do you use?

What are some weaknesses of the requirements approaches/notations you use?

What characteristics or activities do you wish your requirements approaches/notations would support?

What characteristics of a requirements approach/notation do you think are the most important and why?

What techniques do you use to validate your requirements, and what are the pros and cons of those techniques?

Do you use requirements in testing activities? If so, what techniques do you use and what are the pros and cons of your current process?

Please rate the following requirements approaches (1 - very bad and 5 - very good)

| | 1 | 2 | 3 | 4 | 5 | N/A |
|---|---|---|---|---|---|---|
| Agile user stories | ○ | ○ | ○ | ○ | ○ | ○ |
| Informal use cases | ○ | ○ | ○ | ○ | ○ | ○ |
| Formalized use cases | ○ | ○ | ○ | ○ | ○ | ○ |
| Shall statements | ○ | ○ | ○ | ○ | ○ | ○ |
| Scenarios | ○ | ○ | ○ | ○ | ○ | ○ |
| Goal-based requirements | ○ | ○ | ○ | ○ | ○ | ○ |
| Natural language with natural language processing | ○ | ○ | ○ | ○ | ○ | ○ |
| Other | ○ | ○ | ○ | ○ | ○ | ○ |

Other (please specify)

## 11. Thank you!

Thank you for participating in this survey!

If you have any questions or comments about requirements engineering, software engineering, and/or this survey, please don't hesitate to contact me <awinblad@ics.uci.edu>.

Kristina Winbladh

Comments

# A Appendix B

| Answer Options | Response Percent | Response Count |
|---|---|---|
| Yes, I have read the Information Study Sheet. | 100.0% | 26 |
| | answered question | 26 |
| | skipped question | 0 |

Q2. Please select one of the following choices

| Answer Options | Response Percent | Response Count |
|---|---|---|
| I am or have been working on the development of a software project | 100.0% | 26 |
| I am or have been a customer of a software project | 0.0% | 0 |
| | answered question | 26 |
| | skipped question | 0 |

Q3. Please rate your experience with the following software engineering tasks. Experience can be anything such as creation, manipulation, and/or evaluation.

| Answer Options | None | Very little | Reasonable | Considerable | Expert | Response Count |
|---|---|---|---|---|---|---|
| Software requirements | 0 | 2 | 7 | 9 | 7 | 25 |
| Software architecture/design | 0 | 4 | 5 | 6 | 10 | 25 |
| Software implementation | 0 | 2 | 3 | 8 | 12 | 25 |
| Software testing | 0 | 4 | 11 | 8 | 2 | 25 |
| Software project planning | 0 | 2 | 8 | 10 | 3 | 23 |
| Other | 2 | 0 | 0 | 2 | 0 | 4 |
| Comments | | | | | | 1 |
| | | | | | answered | 25 |
| | | | | | skipped | 1 |

Q4. Experience in software development

| Answer Options | Response Percent | Response Count |
|---|---|---|
| < 1  year | 0.0% | 0 |
| 1-5  years | 32.0% | 8 |
| 5-10 years | 16.0% | 4 |
| > 10 years | 52.0% | 13 |

| | | |
|---|---|---|
| | answered question | 25 |
| | skipped question | 1 |

Q5. Please specify from which perspective you are taking the survey.

| Answer Options | Response Percent | Response Count |
|---|---|---|
| Software engineering in the software industry | 92.0% | 23 |
| Software engineering research (faculty) | 0.0% | 0 |
| Software engineering research (graduate student) | 0.0% | 0 |
| Other (please specify) | 8.0% | 2 |
| | answered question | 25 |
| | skipped question | 1 |

Q6. Name of institution.

| Answer Options | Response Count |
|---|---|
| | 25 |
| answered question | 25 |
| skipped question | 1 |

Q7. Education

| Answer Options | Response Percent | Response Count |
|---|---|---|
| No formal education | 4.0% | 1 |
| B.S. / B.A. | 52.0% | 13 |
| M.S. / M.A. | 24.0% | 6 |
| PhD | 20.0% | 5 |
| Comments | | 0 |
| | answered question | 25 |
| | skipped question | 1 |

Q8. What is your experience working with software projects?

| Answer Options | Response Percent | Response Count |
|---|---|---|
| < 1 year | 0.0% | 0 |
| 1-5 years | 0.0% | 0 |
| 5-10 years | 0.0% | 0 |
| > 10 years | 0.0% | 0 |
| | answered question | 0 |
| | skipped question | 26 |

Q9. Briefly describe your role in software projects.

| Answer Options | Response Count |
|---|---|
| | 0 |
| answered question | 0 |
| skipped question | 26 |

Q10. Please rate your experience with the following software engineering tasks. Experience can be anything such as creation, manipulation, and/or evaluation.

| Answer Options | None | Very little | Reasonable | Considerable | Expert | Response Count |
|---|---|---|---|---|---|---|
| Software requirements | 0 | 0 | 0 | 0 | 0 | 0 |
| Software architecture/design | 0 | 0 | 0 | 0 | 0 | 0 |
| Software implementation | 0 | 0 | 0 | 0 | 0 | 0 |
| Software testing | 0 | 0 | 0 | 0 | 0 | 0 |
| Software project planning | 0 | 0 | 0 | 0 | 0 | 0 |
| Other | 0 | 0 | 0 | 0 | 0 | 0 |
| Comments | | | | | | 0 |
| | | | | answered question | | 0 |

| | |
|---|---|
| skipped question | 26 |

Q11. Name of institution.

| Answer Options | Response Count |
|---|---|
| | 0 |
| answered question | 0 |
| skipped question | 26 |

Q12. Education

| Answer Options | Response Percent | Response Count |
|---|---|---|
| No formal education | 0.0% | 0 |
| B.S./B.A. | 0.0% | 0 |
| M.S./M.A. | 0.0% | 0 |
| PhD | 0.0% | 0 |
| Comments | | 0 |
| | answered question | 0 |
| | skipped question | 26 |

Q13. How important are the following characteristics of requirements approaches/notations?

| | |
|---|---|
| (1 - not important | 5 - very important)" |

| Answer Options | 1 | 2 | 3 | 4 | 5 | Rating Average | Response Count |
|---|---|---|---|---|---|---|---|
| Understandability | 0 | 0 | 1 | 9 | 15 | 4.56 | 25 |
| Unambiguity | 0 | 1 | 2 | 8 | 14 | 4.4 | 25 |
| | | | | | | answered question | 25 |
| | | | | | | skipped question | 1 |

Q14. Under typical software development constraints, one might need to choose between unambiguity and understandability. Which would you consider more important?

| Answer Options | Response Percent | Response Count | |
|---|---|---|---|
| Understandability | 64.0% | 16 | 64 |
| Unambiguity | 36.0% | 9 | 36 |
| | answered question | 25 | |

| | |
|---|---|
| skipped question | 1 |

Q15. Please add any additional comments or motivations.

| Answer Options | Response Count |
|---|---|
| | 7 |
| answered question | 7 |
| skipped question | 19 |

Q16. How important is it that requirements be used to guide the the following testing activities?

(1 - not important 5 - very important)"

| Answer Options | 1 | 2 | 3 | 4 | 5 | Rating Average | Response Count |
|---|---|---|---|---|---|---|---|
| Test case creation | 0 | 0 | 0 | 7 | 15 | 4.68 | 22 |
| Test data selection | 0 | 2 | 6 | 8 | 6 | 3.82 | 22 |
| Test coverage criteria definition | 2 | 0 | 9 | 6 | 5 | 3.55 | 22 |
| Test oracle creation | 3 | 2 | 5 | 6 | 6 | 3.45 | 22 |
| | | | | | | answered question | 22 |
| | | | | | | skipped question | 4 |

Q17. Under typical software development constraints, how would you rank the relative importance of using requirements to guide these testing activities?

| Answer Options | Most important | Second most important | Third most important | Rating Average | Response Count | fourth |
|---|---|---|---|---|---|---|
| Test case creation | 16 | 5 | 1 | 3.68 | 22 | 0 |
| Test data selection | 3 | 8 | 6 | 2.82 | 17 | 5 |
| Test coverage criteria | 3 | 3 | 8 | 2.64 | 14 | 8 |
| Test oracle creation | 0 | 6 | 7 | 2.46 | 13 | 9 |
| | | | | answered question | 22 | |
| | | | | skipped question | 4 | |

Q18. Please add any additional comments or motivations.

| Answer Options | Response Count |
|---|---|
| | 8 |
| answered question | 8 |
| skipped question | 18 |

Q19. How important are the following characteristics?

(1 - not important 5 - very important)"

| Answer Options | 1 | 2 | 3 | 4 | 5 | Rating Average | Response Count |
|---|---|---|---|---|---|---|---|
| Ease | 1 | 0 | 9 | 6 | 6 | 3.73 | 22 |
| Quality | 0 | 0 | 6 | 6 | 10 | 4.18 | 22 |
| | | | | | | answered question | 22 |
| | | | | | | skipped question | 4 |

Q20. Under typical software development constraints, one might need to choose between ease and quality. Which would you consider more important?

| Answer Options | Response Percent | Response Count |
|---|---|---|
| Ease | 40.9% | 9 |
| Quality | 59.1% | 13 |
| | answered question | 22 |
| | skipped question | 4 |

Q21. Please add any additional comments and motivations.

| Answer Options | Response Count |
|---|---|
| | 7 |
| answered question | 7 |
| skipped question | 19 |

Q22. How important are the following characteristics of requirements approaches?

(1 - not important 5 - very important)"

| Answer Options | 1 | 2 | 3 | 4 | 5 | Rating Average | Response Count |
|---|---|---|---|---|---|---|---|
| Initial ease | 1 | 2 | 10 | 4 | 5 | 3.45 | 22 |
| Maintenance ease | 0 | 0 | 1 | 11 | 10 | 4.41 | 22 |
| | | | | | | answered question | 22 |
| | | | | | | skipped question | 4 |

Q23. Under typical software development constraints, one might have to choose between initial ease and maintenance ease. Which would you consider more important?

| Answer Options | Response Percent | Response Count | |
|---|---|---|---|
| Initial ease | 22.7% | 5 | 22.7272727 |
| Maintenance ease | 77.3% | 17 | 77.2727273 |
| | answered question | 22 | |
| | skipped question | 4 | |

Q24. Please add any additional comments and motivations.

| Answer Options | Response Count |
|---|---|
| | 8 |
| answered question | 8 |
| skipped question | 18 |

Q25. Below is a list of activities that requirements approaches can support. Please select three alternatives that you think are the most important.

| Answer Options | Response Percent | Response Count | |
|---|---|---|---|
| automatic test generation | 63.6% | 14 | 63.6363636 |
| cognitive support | 18.2% | 4 | 18.1818182 |
| model checking | 18.2% | 4 | 18.1818182 |
| other | 4.5% | 1 | 4.54545455 |
| support for informal conversations with customers and/or users | 59.1% | 13 | 59.0909091 |
| support for maintenance | 68.2% | 15 | 68.1818182 |
| support for transition to design | 68.2% | 15 | 68.1818182 |
| Comments | | 2 | |
| | answered question | 22 | |
| | skipped question | 4 | |

Q26. Below is a list of characteristics that requirements approaches can exhibit. Please select three alternatives that you think are the most important.

| Answer Options | Response Percent | Response Count | |
|---|---|---|---|
| domain descriptions | 54.5% | 12 | 54.5454545 |
| formal notations | 13.6% | 3 | 13.6363636 |
| links between requirements | 50.0% | 11 | 50 |
| other | 4.5% | 1 | 4.54545455 |
| operational descriptions | 81.8% | 18 | 81.8181818 |
| property descriptions | 31.8% | 7 | 31.8181818 |
| traceability between requirements and other software artifacts | 63.6% | 14 | 63.6363636 |
| Comments | | 1 | |
| | answered question | 22 | |
| | skipped question | 4 | |

Q27. What requirements approaches/notations do you use?

| Answer Options | Response Count |
|---|---|

| | |
|---|---|
| | 18 |
| answered question | 18 |
| skipped question | 8 |

Q28. What are some weaknesses of the requirements approaches/notations you use?

| Answer Options | Response Count |
|---|---|
| | 17 |
| answered question | 17 |
| skipped question | 9 |

Q29. What characteristics or activities do you wish your requirements approaches/notations would support?

| Answer Options | Response Count |
|---|---|
| | 15 |
| answered question | 15 |
| skipped question | 11 |

Q30. What characteristics of a requirements approach/notation do you think are the most important and why?

| Answer Options | Response Count |
|---|---|
| | 17 |
| answered question | 17 |
| skipped question | 9 |

Q31. What techniques do you use to validate your requirements, and what are the pros and cons of those techniques?

| Answer Options | Response Count |
|---|---|
| | 17 |
| answered question | 17 |
| skipped question | 9 |

Q32. Do you use requirements in testing activities? If so, what techniques do you use and what are the pros and cons of your current process?

| Answer Options | Response Count |
|---|---|
| | 17 |
| answered question | 17 |
| skipped question | 9 |

Q33. Please rate the following requirements approaches (1 - very bad and 5 - very good)

| Answer Options | 1 | 2 | 3 | 4 | 5 | N/A | Rating Average |
|---|---|---|---|---|---|---|---|
| Agile user stories | 2 | 1 | 5 | 3 | 6 | 3 | 3.59 |
| Informal use cases | 1 | 2 | 6 | 8 | 5 | 0 | 3.64 |
| Formalized use cases | 0 | 3 | 7 | 4 | 6 | 0 | 3.65 |
| Shall statements | 0 | 1 | 5 | 7 | 2 | 6 | 3.67 |
| Scenarios | 1 | 1 | 8 | 6 | 4 | 2 | 3.55 |
| Goal-based requirements | 1 | 1 | 2 | 5 | 6 | 4 | 3.93 |
| Natural language with natural language processing | 0 | 7 | 3 | 2 | 1 | 7 | 2.77 |
| Other | 0 | 0 | 1 | 0 | 3 | 5 | 4.5 |
| Comments | | | | | | | |
| | | | | | | | answered question |
| | | | | | | | skipped question |

Q34. Comments