

# Does One Size Fit All?

## Determining Requirements Information Requirements

**David Gelperin**  
ClearSpecs Enterprises  
2425 Zealand Ave. N.  
Golden Valley, MN 55427  
01-763-591-1534  
david@clearspecs.com

### Abstract

How much requirements information should be elicited and communicated on a specific project? We answer “just enough” based on the risk of insufficient understanding and misunderstanding of higher priority requirements. We assume this risk depends on stakeholder’s initial understanding i.e., that “just enough” decisions are context-dependent. We show that no context-free approach to requirements (e.g. comprehensive or Agile) is best in all situations. We describe a decision process for identifying “just enough” information and provide four examples.

### 1.0. Introduction

Test professionals know about risk [1]. Their understanding of defect likelihood and failure impact guides their test designs. A set of tests should be evaluated against a set of risk assumptions i.e., test design is context-sensitive. When a tester correctly knows there is very little defect likelihood or failure impact (an unlikely situation), there may be no need to test.

We propose that requirements developers increase their awareness of requirements risk and adopt a similar outlook. The risk of inadequate understanding by requirements receivers should shape the choice of requirements information i.e., requirements selection should be context-sensitive. When there is negligible risk of inadequate understanding (e.g. requirements provider and receiver are the same person and that person has a very deep understanding of the needs), there may be no need for a requirements specification. However, a specification might be developed because of customer management strategies, contracts, regulations, or personal preferences.

### 1.1. How much information?

How much requirements information should be elicited and communicated during a project (not necessarily at the beginning)? We assume this to be a fundamental goal-setting question on every project. Even if this goal is well-defined, it may not be reached or even reachable. A comprehensive discussion of managing this attainment risk is outside the scope of this paper.

Here we focus on answers to the “how much” question and find that different communities have different answers.

## 1.2. Comprehensive Answers

The requirements engineering (RE) community answers “a lot” and provides lists such as the following of over 30 types of requirement and constraint information and over 20 types of meta-information. This list does not include the many quality characteristics [2].

### Requirement & Constraint Info

1. System requirements
  - 1.1 Actors, roles, and goals
  - 1.2 Functions, behavior rules, and scenarios
  - 1.3 Data, relationships, and flow
  - 1.4 Quality characteristics
2. Environmental requirements
  - 2.1 Interfaces
  - 2.2 Integration
  - 2.3 Timing
  - 2.4 Platforms
  - 2.5 Internationalization
3. Operational requirements
  - 3.1 Installation
  - 3.2 Operation
  - 3.3 Access
  - 3.4 Documentation
  - 3.5 Training
  - 3.6 Support
4. Domain models and constraints
  - 4.1 Concept and relationship models
  - 4.2 Constant conditions
  - 4.3 Condition dependencies
5. Development constraints
  - 5.1 Verification and Validation
  - 5.2 Design and Implementation – e.g. standards conformance
  - 5.3 Project – e.g. budget, schedule
6. Definitions
  - 6.1 Phrases
  - 6.2 Words
  - 6.3 Acronyms

### Meta-information

7. Requirements Background
  - 7.1 Opportunity/Problem statement
  - 7.2 Intentions, and expectations
  - 7.3 Assumptions – about domain, environment, system, and change drivers
  - 7.4 Alternative solutions
  - 7.5 Environmental impacts
  - 7.6 System justification
8. Requirements Attributes (examples)
  - 8.1 Category
  - 8.2 Source
  - 8.3 Product priority
  - 8.4 Security level
  - 8.5 Owner
  - 8.6 Author
  - 8.7 Modifications
  - 8.8 Allocation
9. Requirement Relationships
  - 9.1 To goals
  - 9.2 To architecture and design elements
  - 9.3 Derived requirements and satisfaction arguments [3]
  - 9.4 Interacting requirements

Common practice in the RE community is to provide less than the listed information.

“In a recent review we conducted of 15 publicly available software requirements specifications, we found an almost universal lack of any requirements describing non-functional qualities.”

Jane Cleland-Huang [4]

### **1.3. Agile Answers**

The Agile community answers “a little” to the how much question and believes that requirements should be developed incrementally, that refactoring (i.e., rework) is unavoidable, and that user stories [5] and acceptance tests [6] are sufficient for effective development and testing of the next feature. The Agile community also believes that Big Requirements Up Front i.e. the RE answer, is itself a project hazard [7].

As with RE practices, there is variance within the Agile community when just using user stories and acceptance tests is deemed insufficient [8, 9].

The Agile bias toward features and working software is a problem when a product architecture needs to be designed. Architectural choices are often driven by quality requirements (safety, security, privacy, usability, performance, reliability, verifiability, maintainability, portability, ...). It is not clear how Agile integrates these product-wide concerns (aspects) into its approach. Microsoft discovered how expensive it is to retrofit security.

In addition, in a situation where customers have a deep understanding of system requirements and developers have a shallow understanding, the Agile bias toward incremental requirements may result in a great deal of unnecessary refactoring.

### **1.4. Both Communities Assume a Static Approach is Best**

The RE community assumes that RE is embedded in a software engineering development process, while Agile requirements are assumed to be embedded in an Agile development process. A mixed approach to requirements (i.e. agile and comprehensive) can also be embedded in an Agile process.

Many proponents in both communities appear to assume that their answers to the “how much requirements information” question do not depend on the stakeholder’s prior knowledge, misunderstanding risk, or application risk i.e., that their requirements approaches are relatively static – perhaps with minor tailoring – and always effective.

### **1.5. Problem with Current Answers**

The software literature provides a comprehensive answer, an agile answer, or both [10]. However, these two choices do not apply in all situations (see examples 4.1, 4.2, and 4.4).

Since inappropriate goals can make a hard job even harder, we present a conceptual framework that provides cost-effective goals in all situations, i.e., provides cost-effective answers to the how much

question. Rather than a static prescription, the “just enough” approach provides a discovery process that creates context-sensitive goals.

### 1.6. What is a Best Answer?

As prolog to a description of the “just enough” approach and to show that “just enough” is the obvious thing to do, consider the following story:

A motorist stops her car and asks a woman sitting at a bus stop if she knows how to get to a restaurant called “The Eatery”. The woman at the bus stop asks if the motorist knows where the Post Office is. The motorist says “no” and the woman says to go straight for 3 blocks, take a right onto Main and go 2 blocks, take a left onto Maple and go 5 blocks and then look on the right. If the motorist had said she knew where the Post Office was, she would have been told “The Eatery is directly across the street “.

Many people with the same information as the woman at the bus stop would respond in the same way and without much thought.

When a question has only one answer (i.e. is context-free), e.g., “how many degrees in a right angle?” the answerer has only one correct response.

When an answer has multiple forms (story above) depending on the prior understanding, misunderstanding, and abilities of the asker, **the answerer needs to know about asker’s knowledge and abilities**. The best answers are not only accurate, but match asker’s abilities and information needs i.e. are necessary, sufficient, and clear (i.e., require minimal effort to understand). Compare these characteristics of best answers to Grice’s description of the four “commonly accepted traits of successful cooperative communication” [11].

A similar customization of response is required of professional financial planners when asked, “How should I invest my assets?”, because **effective answers require background information**. Planners ask about financial goals, risk tolerance, and current resources and then, based on their broad knowledge of effective financial tactics, customize a financial plan.

We believe that requirements development should entail cooperative communication. In addition, we view requirements information as an answer to the question “What do you need?” asked by those who receive and use requirements information to acquire, develop, or change a system. If question askers start with a deep understanding of requirements (e.g., because of experience with similar systems), their question will not be “What do you need?”, but will be a set of detailed questions about specific requirements e.g. “Is an average response time of 2 seconds adequate?”. Askers may know more about requirements than information providers (see example 4.1).

All of this means that successful cooperative communication of requirements information e.g., answering the “What do you need?” question, requires consideration of the background knowledge and information needs of those who will receive and use the information e.g., managers, architects, designers, implementers, and testers.

## 2.0. “Just Enough” Answers

We now describe the concept of customized requirements information.

### 2.1. “Just Enough” Community

In addition to RE and Agile, there is a third (much smaller) development community [12, 13]. This community answers the how much requirements information question with “just enough” and proposes that information be included based on requirements-related project risks. Unfortunately, the specifics of these risks are not provided.

### 2.2. Refining “Just Enough”

The Agile community claims that agile requirements are “just enough” and sometimes they are. At other times, comprehensive (RE) requirements are “just enough”. Sometimes neither is “just enough”.

This paper refines “just enough” by providing specifics about requirements-related project risks. We propose that information be included based primarily on the risk of “**inadequate understanding of higher priority requirements** “. This risk assumes that “higher priority” requirements have been identified – perhaps incorrectly. We describe the risk of inadequate understanding and show that **it depends on initial understanding and misunderstanding by stakeholders** i.e., that “just enough” decisions are context-dependent.

#### 2.2.1. Why requirements information?

Among project stakeholders, we find **requirements receivers** (e.g., designers, developers, testers, lawyers, writers, maintainers) and **requirements providers** (e.g., customers, developers, testers, lawyers), both present and future. *The primary purpose of requirements information is to inform its receivers about solution needs and constraints so they can do their jobs effectively and efficiently.* What if receivers are already informed at the start of the project? What if providers don’t have enough understanding to inform receivers? We consider these situations and others and provide a process for deciding what requirements information should be discovered and communicated.

Requirements information should be clear, necessary, persistently accurate and sufficient so its receivers can do their jobs. While clarity and necessity can be achieved with effort, persistent accuracy and sufficiency are only achievable when both providers and receivers have sufficient understanding of satisfactory solutions.

When requirements information is necessary and sufficient, it is “just enough”. When it is necessary but not sufficient, it is “not enough”. When it is sufficient, but not necessary, it is “more than enough”.

### 2.2.2. Mapping Initial Understanding Risk

“It ain’t so much the things we don’t know that get us in trouble. It’s the things we know that just ain’t so.”

Paraphrasing Josh Billings (Henry Wheeler Shaw)

The following grid maps the risk associated with the initial understanding of higher-priority requirements by providers and receivers. This risk includes lack of understanding, awareness of this lack, and the likelihood and severity of misunderstanding.

<b>Shallow</b> Provider Understanding			
<b>Medium</b> Provider Understanding			
<b>Deep</b> Provider Understanding			
	<b>Deep</b> Receiver Understanding	<b>Medium</b> Receiver Understanding	<b>Shallow</b> Receiver Understanding

**Table 1. Initial Understanding Influences Risk**

Requirements provider understanding of problem domain and specific needs and wants

Requirements receiver understanding of problem and solution domains

Fill Colors: Red – higher risk, Yellow – medium risk, Green – lower risk

The higher risk (red) area has many opportunities for major misunderstandings as well as inadequate understanding.

Understanding varies by aspect as well as by stakeholder. An entire system may be in one cell or different aspects may be in different cells for different stakeholders.

Requirements should be mapped into “understanding space” cells. Initial understanding should be assessed by the stakeholder group assuming the project environment supports truth telling. Misunderstandings and false claims about understanding make it difficult to determine appropriate cells. Different cell proposals for a single aspect indicate differing stakeholder assumptions about initial understanding.

### 2.2.3. Risk of Inadequate Understanding

The risk of inadequate receiver understanding is lower in the first column, because requirements receivers begin with a deep understanding of the problem and its alternative solutions. This lower risk means that “just enough” would correspond to “a little” additional information. This information is needed to “fill in the blanks” in the receivers requirements knowledge and therefore is dependent on what receivers don’t know. Note that this is not the small amount of information in an Agile approach.

Risk of inadequate provider understanding is lower in the third row, because requirements providers begin with a deep understanding of the problem domain and their specific needs and wants.

Unfortunately, deep understanding by providers does not entail deep understanding by information receivers – without the Vulcan mind meld.

In particular, in the third cell of the third row, “just enough” is “a lot” and an effective communication strategy must be developed. This strategy depends on the nature of the unknown information and receiver learning styles. For example, perhaps communication about quality attributes should be comprehensive and communication about roles, functions, and data should be incremental.

Outside column 1 and row 3, understanding risk is higher because neither receivers nor providers have deep understanding.

Risk is medium in cell 3 of row 1 because everyone understands that significant research is required because little is known.

### 3.0. “Just Enough” Selection Process

We now describe a process for identifying the requirements information that should be elicited and communicated. To guide selection of “just enough” information, we use a sequence of questions about specifying. The answer to each question focuses the following questions. For example, on a specific project, identifying why you are specifying guides identification of what and how much to specify. Identifying why and what guides identification of how and when.

#### 3.1. **Who** knows and needs to know?

Begin by identifying individuals and groups of stakeholders and third parties who potentially understand customer requirements (providers) and those stakeholders who need to understand the requirements to do their jobs effectively (receivers).

#### 3.2. **Why** specify?

This question focuses on the project-specific goals of a requirements information strategy. The question is not whether the following candidate goals are good in general, but whether specification effort will be balanced by the satisfaction of project-specific needs. Typical goals for specification include:

1. Common Understanding -- What is the risk of unaligned understanding, misunderstanding, or incomplete understanding within a provider or receiver group or between providers and

suppliers? Even if significant understanding risk exists, are large specifications likely to be part of a cost-effective risk management strategy?

2. Discovery -- Is specification a cost-effective strategy for discovering (1) poor understanding (unaligned, incomplete, and wrong), (2) new requirements or details and (3) incomplete or incorrect requirements i.e. an effective foundation for analysis and verification?
3. Reuse – As acceptance tests [6] and user guides [14] as well as in development of product families.
4. Education – Informing new project stakeholders (present) as well as system maintainers (future), who may need to understand the rationale for system behaviors and characteristics.
5. Leverage -- Can specified details be used as parameter values for system generation e.g., website or report generators?
6. Recording -- Does common understanding need to be documented because of customer requirements, contract specifications, regulations or laws?

### 3.3. **What and How much** to specify?

Considering specification goals and information needs, select those aspects whose requirements need to be specified and the details needed for each aspect. Aspects, whose requirements are known to be in the red area of “understanding space”, should be specified. Ask requirements receivers exactly what information and in what forms they need to do their jobs.

Not all aspects need the same amount of detail. Some required functions and quality characteristics may be better understood than others. For example, some required functions may be well understood, while some required quality characteristics may not.

Other details may be needed to create consensus within a customer group, to use advanced analytic techniques (proof of correctness) or because of third-party testing, government regulations, or legal considerations.

### 3.4. **How** to specify?

After deciding on what and how much to specify, identify suitable forms for selected aspects, details, and audience. Candidate forms include:

- Text, free-form & structured (profiles, lists)
- Scenarios, free-form & structured (use cases)
- Stories
- Acceptance Criteria & Tests
- User Guides
- Screen shots
- Mockups
- Simulations
- Diagrams (graphs, charts)
- Tables
- Equations
- Rules
- Definitions
- Formal languages

Each form is effective for specific types of information and for specific audiences. A blackboard full of equations may aid communication among theoretical physicists, but may not be understood by those who set their research goals.

### 3.5. Who specifies?

Effective specification writing requires a clear understanding of objectives and scope (why, what, and how much), a broad knowledge of forms (how) and skill at clear technical communication. Current practice reflects a view that no training in requirements or technical writing is required to author a specification [15]. Unless a set of templates has proven effective in an organization that repeatedly develops similar systems, it is difficult to imagine the decisions described above in 3.4 being made effectively by most authors without such training. Consider using an experienced technical writer to author larger specifications.

When “just enough” information needs to be supplemented, consider having receivers specify supplemental information. The accuracy of their information shows the quality of their understanding.

### 3.6. When to specify?

If a requirements specification is required e.g. by contract or regulation, but (1) you don’t know what is possible – upper right cell of Table 1 or (2) specification information is not needed for development – left column of Table 1 or (3) both i.e., different aspects of a single system, consider specifying after design and implementation or specifying some aspects before and some after. In the other five cases (cells), iterate a “specify, then design” process down through the hierarchic structure of the system.

## 4.0. Four Examples

We now apply the “just enough” selection process to corner-cases in “understanding space” i.e., grid below. These examples are based on the author’s direct experience.

<b>Shallow</b> Provider Understanding	<b>1</b>		<b>4</b>
<b>Medium</b> Provider Understanding			
<b>Deep</b> Provider Understanding	<b>2</b>		<b>3</b>
	<b>Deep</b> Receiver Understanding	<b>Medium</b> Receiver Understanding	<b>Shallow</b> Receiver Understanding

Table 2. Mapping the Specification Examples

1. e-commerce website
2. new release
3. rich glossary manager
4. drug ad analyzer

#### **4.1. Specification for e-commerce website**

Situation: A merchant, needing an e-commerce website, goes to experienced developers that specialize in such sites. Developers know the technology and the application, but need to elicit detailed preferences by asking questions. Some questions will be asked indirectly by showing the customer sample sites and asking which is preferred and why. This is similar to the process used by an architect in determining requirements for a custom home.

Profile:

**Who** knows?

Merchant has a shallow understanding of application concepts or terminology. Developers have a deep technical understanding, but have little understanding of the merchant's business.

**Why** specify?

To communicate details about the website and the project. To use details as parameter values in content management and shopping cart systems e.g. Joomla and VirtueMart.

**What** and **How much**?

Detailed preferences for site characteristics such as optional functions, interfaces, and internationalization and details about project characteristics such as schedule and budget.

**How**?

Developers may start with a template of typical requirements issues and use a questionnaire to identify required details. Some information will be communicated in conversation, recorded in code, and approved by the merchant.

**Who** specifies?

Developers tailor a questionnaire and merchant fills it out. Some questions may never be answered in writing. Developers may specify some aspects to verify understanding.

**When**?

At the beginning of the project.

#### **4.2. Specification for new release of established product**

Situation: A product manager describing changes needed in the next release of an established product to group of experienced developers who maintain the product. He specifies additions, changes, and deletions required. Some changes may have been suggested by the developers themselves. By implication, everything else must stay the same.

Profile:

**Who** knows?

Manager and developers have deep understanding. [This may be analogous to a long-married couple, where a word or gesture can speak volumes due to the richness of their shared experience.]

**Why** specify?

To communicate and align understanding about changes and the project.

**What** and **How much**?

List of changes. Scope and depth of details depend on manager's understanding of the changes that are not known by developers. Some information will be communicated in conversation, recorded in code, and approved by the manager.

**How?**

With forms that match the nature of the changes.

**Who** specifies?

Product manager and possibly developers. Neither manager nor developers may understand some change options. Developers may research and specify these options so that manager can choose one to implement.

**When?**

At the beginning and possibly during the project, if developers specify some options.

#### **4.3. Specification for Rich Glossary Manager**

Situation: An experienced software engineer with a detailed design for a Rich Glossary Manager (manages "rich definitions" [16]) goes to an experienced software development company that is unfamiliar with the new application.

Profile:

**Who** knows?

Software engineer has a deep understanding, while developers have shallow understanding and do not know application concepts or terminology [16].

**Why** specify?

To communicate and align understanding about the application and the project.

**What** and **How much**?

Most types of requirements and constraint information in sufficient detail to allow developers semi-familiar with application domain to work effectively. As developers increase their understanding, fewer details will be needed in the requirements for unimplemented features. In addition, specify acceptance criteria and tests for each requirement.

**How?**

By choosing forms that match the nature of the features. Using Acceptance Test Driven Development [17] helps developers understand the details.

**Who specifies?**

Software engineer, possibly working with a technical writer, specifies the requirements. Software engineer, testers, and developers specify acceptance criteria and tests.

**When?**

Using incremental development, create a high-level application specification and acceptance criteria and tests before the first increment and a detailed increment specification and acceptance criteria and tests before each increment.

**4.4. Specification for a drug ad analyzer**

Situation: A physician asks a software engineer to find out if software can be used to analyze prescription drug ads against rules specified by the Federal Drug Administration.

Profile:**Who knows?**

Physician and developer have a shallow understanding of what, if anything is possible. Physician knows some concepts and terminology of the application domain, but developer does not.

**Why specify?**

To record and align understanding of the problem. To educate new stakeholders such as future employees of a resulting business and potential clients.

**What and How much?**

FDA rules for different types of prescription drug ads divided into (1) rules software will check for ad compliance and (2) rules software will not check. Include rule priorities derived from study of FDA letters sent to drug companies. Very few aspects of a solution are specified, except "be of clear value" and "do no harm".

**How?**

Using lists of rules ordered by priority.

**Who specifies?**

Physician and developer.

**When?**

At the beginning and during the project.

#### 4.5. Implications

These examples demonstrate the results of using the “just enough” selection process in very different situations. They show that the information needed and available in the four corners of “understanding space” (Table 2) varies widely. They also indicate the challenge of requirements management when aspects of a large complex system appear in different cells of “understanding space” i.e. the challenge of heterogeneous “just enough” requirements.

The scope of “understanding space” and these examples suggest that **no static answer to the “how much” question or small set of static answers is adequate in all situations** (i.e. would provide requirements information that is both necessary and sufficient at all points in “understanding space”). This implies that a new project in an unfamiliar situation needs to identify its requirements information requirements based on project context.

#### 5.0. What about incorrect assessment of initial understanding risk?

A process based on human judgment runs the risk of incorrect decisions. This means that selection of “just enough” information may be incorrect. The most serious mistakes are communicating incorrect and insufficient information, particularly information needed to correct receiver misunderstandings.

Since any project decision can be wrong, it is prudent to include tactics in the project strategy for closely monitoring decision consequences and reducing the impact of poor decisions. For example, developer understanding can be monitored by carefully assessing their partial work products and by requiring them to demonstrate their understanding. Demonstrations may entail test designs, use cases, usage scenarios, state charts, or decision tables **created by the developers**. Aspects, whose requirements are known to be in the red area of “understanding space”, should be closely monitored.

The impact of poor decisions can be reduced using mitigation tactics such as (1) incremental and iterative development and (2) use of powerful technologies that match the system under development. For example, using a Content Management System for website development makes many types of changes (rework) inexpensive.

#### 6.0. Previous Work

This paper does not cover the broad topic of project risk management [18] nor the broad subtopic of requirements risk management (RRM) [19, 20]. From among approximately 50 types of requirement hazards and approximately 60 RRM tactics, this paper focuses on assessing a single hazard (inadequate understanding of higher priority requirements ) and using a single RRM tactic (using assessment of a single hazard to decide how much requirements information to elicit and communicate).

In other work[10], Boehm and Turner have identified “*five critical factors* (project size, application criticality, project dynamism, project staffing, and project culture) *that determine the relative suitability of agile or plan-driven* (software engineering) *methods in a particular project situation*”. Since only project size and dynamism may correlate with inadequate understanding risk, the Boehm and Turner

selection process and the “just enough” selection process may result in different requirements strategies.

While project staffing and culture factors do not determine ideal choices, they do determine which development methods, if any, can be successful. For example, when a team only knows or only believes in “big requirements up front”, it won’t do “just enough” in many situations. Boehm and Turner select workable strategies, while “just enough” selects an amount of requirements information that is sufficient, but may not be workable.

## 7.0. Conclusions

“Just Enough” requirements:

- Focus primarily on the risk of **“inadequate initial understanding of higher priority requirements”** by requirements receivers
- Consider initial understanding by stakeholders and **remaining information** to be discovered and communicated that will enable requirements receivers to do their jobs
- Uncover differing stakeholder assumptions about initial understanding
- Result in a context-sensitive requirements strategy

When inadequate understanding risk is relatively constant across projects, then “just enough” information choices can be made once and tailored by experience. When risk profiles are more dynamic, cost-effective specification requires customized choices.

Sometimes lack of stakeholder understanding and misunderstandings lead to “not enough” requirements and tactics to mitigate this risk become important.

## 8.0. Acknowledgements

Thanks to Lee Copeland and two anonymous reviewers for insightful comments on earlier drafts. Special thanks to Donald Firesmith for an exceptionally thorough review of a previous draft.

## 9.0. References

1. Paul Gerrard and Neil Thompson **Risk Based E-Business Testing** Artech House 2002
2. Karl Wiegers **Software Requirements** Microsoft Press 2003 (Chapter 12)
3. Katrina Attwood, Tim Kelly, and John McDermid “The Use of Satisfaction Arguments for Traceability in Requirements Reuse for System Families” International Workshop on Requirements Reuse in System Family Engineering , 2004
4. Jane Cleland-Huang et. al. “Automated Classification of Non-functional Requirements” Requirements Engineering 12, 2, April 2007
5. Mike Cohn **User Stories Applied: For Agile Software Development** Addison-Wesley 2004
6. Kent Beck **Test Driven Development By Example** Addison-Wesley, 2002
7. Scott Ambler “Examining the Big Requirements Up-Front (BRUF) Approach”  
[www.agilemodeling.com/essays/examiningBRUF.htm](http://www.agilemodeling.com/essays/examiningBRUF.htm)

8. Scott Ambler "Requirements Specifications on Agile Projects" [www.modernanalyst.com](http://www.modernanalyst.com)
9. Jennitta Andrea "Agile Requirements for Stepsister Projects" Better Software, Sept. 2005
10. Barry Boehm and Richard Turner **Balancing Agility and Discipline** Addison-Wesley, 2004
11. ... "Gricean maxims" Wikipedia
12. Alan M. Davis **Just Enough Requirements Management: Where Software Development Meets Marketing** Dorset House, 2005
13. Dean Leffingwell "Agile Requirements Methods" The Rational Edge July 2002
14. Daniel M. Berry et. al. "User's Manual as a Requirements Specification" Requirements Engineering 9, 1 February, 2004
15. Donald Firesmith, "Specifying Good Requirements," Journal of Object Technology, vol. 2 no.4, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, July-August 2003
16. David Gelperin "What's It Mean" Better Software, April 2009
17. Antony Marcano "From Here to Acceptance Test Driven Development" Better Software, Sept. 2008
18. Tom DeMarco and Timothy Lister **Waltzing With Bears: Managing Risk on Software Projects** Dorset House , 2003
19. David Gelperin "Identifying and Controlling Requirements Risk" (Workshop Slides) among technical papers on [www.clearspecs.com](http://www.clearspecs.com)
20. David Gelperin "Requirements Risk Management: Tactic Checklists and References" among technical papers on [www.clearspecs.com](http://www.clearspecs.com)