

# A Lot, a Little, or Just Enough?

**David Gelperin**

ClearSpecs Enterprises  
2425 Zealand Ave. N.  
Golden Valley, MN 55427  
01-763-591-1534  
[david@clearspecs.com](mailto:david@clearspecs.com)

## Abstract

We define the requirements information that should be elicited and communicated on a specific project as “just enough” based on the risk of inadequate understanding of higher priority requirements. We show that this risk primarily depends on the initial understanding of stakeholders 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 and that the “just enough” approach entails these context-free approaches as special cases. We identify avoidance and mitigation tactics to deal with “not enough” requirements and other requirements risks. Finally, we describe the “just enough” selection process and provide four examples.

## 1.0. Introduction

Test professionals know about risk. Their understanding of defect likelihood and failure impact shapes their test designs. A set of tests can only be evaluated against a set of risk assumptions i.e., test design is context-sensitive. No risk means no need to test.

We propose that requirements developers increase their awareness of risk and adopt a similar outlook. The risk of misunderstanding by requirements users should shape the choice of requirements information. A set of requirements should be evaluated against a set of risk assumptions i.e., requirements selection should be context-sensitive. No misunderstanding risk means no need for requirements.

### 1.1. How much information?

How much requirements information should be elicited and communicated during a project? We assume this should be a fundamental goal-setting question on each project. Even if this goal is accurately 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 this goal-setting question and we find that different communities have different answers.

### 1.2. A Comprehensive Answer

The requirements engineering (RE) community answers “a lot” and provides lists such as the following of over 25 types of requirement and constraint information and over 25 types of meta-information. This list does not include the many quality attributes [12].

## Requirement & Constraint Info

1. System requirements
  - 1.1 Functions
  - 1.2 Data and flow
  - 1.3 Quality attributes
2. Environmental requirements
  - 2.1 Platforms
  - 2.2 Internationalization
  - 2.3 Integration
  - 2.4 Interfaces
3. Operational requirements
  - 3.1 Installation
  - 3.2 Operation
  - 3.3 Users
  - 3.4 Access
  - 3.5 Usage
  - 3.6 Documentation
  - 3.7 Training
  - 3.8 Support
4. Development constraints
  - 4.1 Verification and Validation
  - 4.2 Design and Implementation – e.g. standards conformance
  - 4.3 Project – e.g. budget, schedule
5. Domain constraints
  - 5.1 Constant conditions
  - 5.2 Condition dependencies
6. Definitions
  - 6.1 Phrases
  - 6.2 Words
  - 6.3 Acronyms

## Meta-information

7. Background
  - 7.1 Opportunity/Problem statement
  - 7.2 Goals, 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. Attributes
  - 8.1 Identifier
  - 8.2 Category
  - 8.3 Source
  - 8.4 Product priority
  - 8.5 Security level
  - 8.6 Assumptions
  - 8.7 Verification
  - 8.8 Implementation (value, risk, effort, priority)
  - 8.9 Owner
  - 8.10 Author
  - 8.11 Issues
  - 8.12 Modifications
  - 8.13 Allocation
9. Relationships (meta information)
  - 9.1 Prerequisite requirements
  - 9.2 Corequisite requirements
  - 9.3 Interacting requirements
  - 9.4 Derivations and satisfaction arguments [2]

### **1.3. An Agile Answer**

The Agile community answers “a little” and believes that requirements should be developed incrementally, that refactoring (i.e., rework) is unavoidable, and that user stories [6] and acceptance tests [3] are sufficient for the 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 risk [1].

However, the Agile bias toward features and working software may be a problem when a product architecture needs to be designed. Architectural choices are often driven by quality requirements (safety, security, privacy, performance, robustness ...). It is not clear how Agile integrates these product-wide concerns 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, 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. Note that a mixed approach to requirements (i.e. agile and comprehensive) could be embedded in an Agile development process.

Both communities assume that their answers to the “how much requirements information” question do not depend on stakeholder, application, or project characteristics, i.e. that their requirements approaches are predefined and static (context-free) – perhaps with minor tailoring.

#### 1.5. The Problem with Current Answers

The requirements literature provides a comprehensive answer, an agile answer, or both [5]. However, these two choices do not apply in all situations (see examples 5.1, 5.2, and 5.4).

Since inappropriate or inadequate goals can make a hard job even harder, we present a conceptual framework that provides effective goals in all situations, i.e., provides cost-effective answers to the how much question. Rather than a static goal set, this framework provides a discovery process that creates a context-sensitive goal set.

### 2. The “Just Enough” Answer

#### 2.1. The “Just Enough” Community

There is another (much smaller) development community [7, 11]. This community answers the how much question with “just enough” and proposes that information be included based on requirements-related project risk.

*the software lifecycle exists for only one reason: to mitigate the risk that requirements-related issues will prevent a successful project outcome. If there were no such risks, then it would be far more efficient to go straight to code and eliminate the overhead of requirements-related activities.*

**Dean Leffingwell**, Agile Requirements Methods [11]

Unfortunately, the specifics of this risk were 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”.

We show that with suitable definitions, **“just enough” entails comprehensive and Agile requirements as special cases** in the sense that either may be “just enough” sometime.

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

### **2.2.1. Why requirements information?**

Among project stakeholders, we find **requirements users** (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 users about solution needs and constraints so they can do their jobs effectively and efficiently.* But what if the users are already informed at the beginning of the project? Or what if the providers don’t have enough understanding to inform the users? We consider these situations and others and provide a process for deciding what requirements information should be communicated.

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

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. Initial Understanding and Remaining Information**

Before considering the risk of inadequate understanding, we consider the relationship between initial (accurate) understanding of higher-priority requirements by users and providers and the remaining requirements information that should be communicated. The following grid shows this relationship. We assume that 100% represents the communication of all requirements information.

<b>Shallow PU</b>	5%		(Research)
<b>Medium PU</b>	5-10%	30-35%	
<b>Deep PU</b>	5-15%	45-55%	70-85%
	<b>Deep UU</b>	<b>Medium UU</b>	<b>Shallow UU</b>

**Table 1. Initial Understanding Influences the Amount of Remaining Information**

**PU** – Requirements **Provider Understanding** of problem domain and specific needs and wants

**UU** – Requirements **User Understanding** of problem and solution domains

**%** – estimates of remaining information that should be communicated

An entire application may be in one cell or different aspects of an application may be in different cells.

If requirements users have a shallow understanding of the application domain (right column), they need to learn domain concepts and terminology to understand the requirements.

### 2.2.3. Risk of Inadequate Understanding

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

Risk of inadequate provider understanding is lowest 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 users – 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 user learning styles. Perhaps communication should be incremental, or comprehensive, or a mixture.

Outside column 1 and row 3, understanding risk is highest because neither users nor providers have deep understanding. This risk can be reduced by using tactics that increase understanding and tactics that mitigate the impact of misunderstanding.

Notice that “just enough” analysis, because it entails determining a project’s location(s) in understanding space (i.e. Table 1), naturally leads to partial awareness of requirements risk [10] and the potential need for risk management tactics.

### 3. Managing Requirements Risk

#### 3.1. Requirements Risk

What if –

- both users and providers have inadequate initial understanding of requirements?
- assessment of initial stakeholder understanding or requirements priorities is inaccurate?
- selected requirements information or its communication is defective?

Communication will be defective whenever effective specification techniques are not used. For example, when a complex conditional expression (Boolean function) needs to be specified, but derived conditions [9] are not used, communication will be defective.

Despite our best efforts, understanding will be inadequate, assessments will be inaccurate and requirements and their communication will be defective along with many other types of requirements risk [10]. Tactics to manage this risk can be organized into the following categories:

- Monitor reqts and project feasibility
- **Avoid** reqts-related mistakes
- **Mitigate** impact of reqts-based problems
- Minimize reqts-based problems
- Monitor reqts and risk status
- Staff for reqts development
- Plan for reqts risk mgmt
- Prepare for reqts risk mgmt
- Compensate for reqts-based problems

Effective risk management is inherently iterative i.e., you need to monitor, learn, and adapt your strategy as you get smarter. This applies to managing the risk of inadequate understanding as well as other types of requirements risks. We now discuss avoidance and mitigation tactics.

#### 3.2. Avoiding Mistakes

Mistake avoidance tactics reduce the likelihood of problems. The following examples deepen or validate stakeholder understanding to avoid requirements-related mistakes.

- 3.2.1. Include stakeholders with a deep understanding of the solution domain.

- 3.2.2. Study solutions to similar problems to identify unforeseen or misprioritized requirements.
- 3.2.3. Have developers maintain similar applications.
- 3.2.4. Immerse stakeholders i.e., customers in system development and developers in customer activities
- 3.2.5. Provide training in domain concepts and terminology.
- 3.2.6. For vague requirements, identify alternative solutions and their costs (see intentional vagueness[9]).
- 3.2.7. Prototype unfamiliar functions and interfaces.
- 3.2.8. Have users specify information needed to supplement “just enough” e.g., needed for legal reasons . The accuracy of this added information shows the quality of user understanding.
- 3.2.9. Have experienced technical writer author larger specs.
- 3.2.10. Until the project ends, have each stakeholder aggressively try to learn and share understanding.

### 3.3. Mitigating Impact

Risk mitigation focuses on reducing the impact of problems. Requirements mitigation tactics include:

- 3.3.1. Require **frequent demos** of understanding e.g., early test design, incremental development – frequent demos provide frequent feedback and enable earlier correction of vision defects and drive deeper understanding of the “real” requirements earlier.
- 3.3.2. Carefully **analyze results** of each increment to detect variances from expectations and to gain deeper understanding.
- 3.3.3. Use **technologies that match** a solution – strong matches e.g., using content management and shopping cart systems to build e-commerce websites, significantly reduce the cost of rework. Making change cheaper reduces the economic need to get requirements right the first time.

### 4. “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.

#### 4.1. **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 customer group or between customers and suppliers? Even if significant understanding risk exists, are large specs 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 [3] and user guides [4] 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 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?

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

Considering specification goals and information needs, select the problem aspects to specify and the details needed for each aspect. Consider using a RE information list when deciding what to specify.

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

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. If “just enough” information needs to be supplemented, consider using mistake avoidance tactic 3.2.8

#### 4.3. **How** to specify?

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

- Text, free-form & structured (profiles, lists)
- Scenarios, free-form & structured (use cases)
- Stories
- Tests
- Diagrams (graphs, charts)
- Tables
- Equations
- Rules

- User Guides
- Screen shots
- Mockups
- Simulations
- 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.

#### 4.4. Who specifies?

Effective specing 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 little formal training in technical communication is required to author a spec [8]. Unless a set of templates has proven effective in an organization that repeatedly develops similar systems, it is difficult to imagine the decisions described in 4.3 will be made effectively by most authors without such training. Consider using mistake avoidance tactic 3.2.8.

#### 4.5. When to specify?

This seems obvious. Spec before you design and implement. But what if a requirements spec 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) spec information is not needed for development – left column of Table 1 or (3) both i.e., different aspects of a single system. In these cases, it may be smarter to spec after design and implementation or to spec some aspects before and some after. Specs done after implementation are more likely to match the system.

### 5. Four Examples

We now apply the “just enough” selection process to corner-cases in “understanding space” i.e., the grid below.

<b>Shallow PU</b>	1		4	<b>PU</b> – Requirements <b>Provider Understanding</b> of problem domain and specific needs and wants
<b>Medium PU</b>				
<b>Deep PU</b>	2		3	<b>UU</b> – Requirements <b>User Understanding</b> of problem and solution domains
	<b>Deep UU</b>	<b>Medium UU</b>	<b>Shallow UU</b>	

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

Table 2. Mapping the Examples

## 5.1 Spec info 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.

Initial Position in Understanding Space: Merchant has a shallow understanding and does not know application concepts or terminology while developers have deep understanding.

Profile:

Question	Answer
<b>Why?</b>	To communicate details about the site and the project. To use details as parameter values in content management and shopping cart systems e.g. Joomla and VirtueMart.
<b>What and How much?</b>	Detailed preferences for site characteristics such as optional functions, interfaces, and internationalization and details about project characteristics such as schedule and budget.
<b>How?</b>	Developers might use a questionnaire to identify required details and to begin the elicitation process. Some information will be communicated in conversation, recorded in code, and approved by the merchant.
<b>Who?</b>	Developers create a questionnaire and the merchant fills it out. Note that some questions may never be answered in writing.
<b>When?</b>	At the beginning.

## 5.2 Spec info for new release of established product

Situation: A product manager describing changes needed in the next release of an established product to the group of experienced developers that 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.

Initial Position in Understanding Space: 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.

Profile:

Question	Answer
<b>Why?</b>	To communicate and align understanding about changes and the project.
<b>What and How much?</b>	Scope and depth of details depend on the manager's understanding of the changes that is not shared by developers. Some information will be communicated in conversation, recorded in code, and approved by the manager.
<b>How?</b>	With forms that match the nature of the changes.
<b>Who?</b>	Product manager and possibly developers. Neither manager nor developers may understand some change options. Developers may research and spec these options so the manager can choose the one to implement.
<b>When?</b>	At the beginning and possibly during the project, if developers spec options.

### 5.3 Spec info for Rich Glossary Manager

Situation: An experienced software engineer with a detailed design for a Rich Glossary Manager goes to an experienced software development company that is unfamiliar with the new application.

Initial Position in Understanding Space: Software engineer has a deep understanding while developers have shallow understanding and do not know application concepts or terminology [9].

Profile:

Question	Answer
<b>Why?</b>	To communicate and align understanding about the application and the project. To provide a basis for acceptance tests and user guide.
<b>What and How much?</b>	Most types of requirements and constraint information in sufficient detail to allow developers semi-familiar with the application domain to develop effectively. As developers increase their understanding, fewer details will be needed for unimplemented aspects.
<b>How?</b>	By choosing forms that match the nature of the aspects.
<b>Who?</b>	Software engineer, possibly working with a technical writer.
<b>When?</b>	Using incremental development, create a high-level application spec before the first increment and a detailed increment spec before each increment.

## 5.4 Spec info for drug promo 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.

Initial Position in Understanding Space: Physician and developer have a shallow understanding of what, if anything, is possible. Physician knows concepts and terminology of the application domain, but the developer does not.

Profile:

Question	Answer
<b>Why?</b>	To record and align understanding of the problem. To educate new stakeholders.
<b>What and How much?</b>	FDA rules for different types of prescription drug ads divided into (1) rules software will check for ad conformance and (2) rules software will not check. Include rule priorities derived from study of FDA warning letters sent to drug companies. Very few aspects of a solution are specified.
<b>How?</b>	Using lists of rules ordered by priority.
<b>Who?</b>	Physician and developer.
<b>When?</b>	At the beginning and during the project.

## 5.5 Implications

These examples show that the requirements information that is needed and available varies widely as well as showing the results of using the “just enough” selection process in very different situations. These examples also show that **no static approach, or small set of static approaches, is best** (i.e. provides information that is both necessary and sufficient) **in all situations** (i.e. all points in “understanding space”). This implies that a new project in an unfamiliar situation needs to dynamically develop its requirements approach based on project context.

## 6. Conclusion

“Just Enough” requirements:

- Focus primarily on the risk of **“inadequate understanding of the higher priority requirements”** by requirements users
- Consider initial understanding by stakeholders and the **remaining information** that should be discovered and communicated to enable requirements users to do their jobs
- Result in a context-sensitive requirements process

If inadequate understanding risk is fairly 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.

A similar customization is used by good financial planners. 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 based on the specific circumstances.

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

Table 3 shows where “just enough” intersects Agile and Comprehensive requirements.

<b>Shallow PU</b>			
<b>Medium PU</b>			
<b>Deep PU</b>			
	<b>Deep UU</b>	<b>Medium UU</b>	<b>Shallow UU</b>

- Agile reqts
- Agile or Mixed reqts
- Comprehensive reqts
- Neither

Table 3. Likely application areas for Agile and Comprehensive in understanding space

We use one characteristic (risk of inadequate understanding of higher priority requirements) to decide how much requirements information to elicit and communicate and therefore one characteristic to decide when predefined requirements strategies such as Agile or RE might be suitable.

For development strategy selection (not just requirements information), Boehm and Turner [5] identify “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 might correlate with inadequate understanding risk, the Boehm and Turner selection process and the “just enough” selection process may have different outcomes.

While project staffing and culture do not determine an ideal choice, they do determine which development methods if any can be successful. For example, if a team can’t do Agile, it can’t do “just enough” in some situations.

The impact of application criticality is less clear. For critical applications, a specification may need to contain information already known to users so advanced analytic techniques (proof of correctness) can be applied. However, this does not imply that (1) the spec should be developed entirely by providers (see mistake avoidance tactic 3.2.8) or (2) the spec should be comprehensive, because including too much already known information carries its own risk.

## 7. References

1. Scott Ambler "Examining the "Big Requirements Up-Front (BRUF)" Approach"  
[www.agilemodeling.com/essays/examiningBRUF.htm](http://www.agilemodeling.com/essays/examiningBRUF.htm)
2. 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
3. Kent Beck **Test Driven Development By Example** Addison-Wesley 2002
4. Daniel M. Berry et. al. "User's Manual as a Requirements Specification" Requirements Engineering 9, 1 February, 2004
5. Barry Boehm and Richard Turner **Balancing Agility and Discipline** Addison-Wesley 2004
6. Mike Cohn **User Stories Applied: For Agile Software Development** Addison-Wesley Professional 2004
7. Alan M. Davis **Just Enough Requirements Management: Where Software Development Meets Marketing** Dorset House 2005
8. Donald Firesmith, "Specifying Good Requirements," *Journal of Object Technology*, vol. 2 no.4, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, July-August 2003
9. David Gelperin "What's It Mean" Better Software April 2009
10. David Gelperin "Controlling Requirements Risk" among the technical papers on [www.clearspecs.com](http://www.clearspecs.com)
11. Dean Leffingwell "Agile Requirements Methods" The Rational Edge July 2002
12. Karl Wiegers **Software Requirements** Microsoft Press 2003 (Chapter 12)