JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2003**

Vol. 2, No. 4, July-August 2003

Specifying Good Requirements

Donald Firesmith, Software Engineering Institute, U.S.A.

Abstract

Many of the characteristics of properly specified requirements have been well known for many years, at least among professional requirements engineers. Yet most requirements specifications seen today in industry still include many poor-quality requirements. Far too many requirements are ambiguous, incomplete, inconsistent, incorrect, infeasible, unusable, and/or not verifiable (e.g., not testable). To combat this sad state of affairs, this column provides a questionnaire that can be used when specifying and technically evaluating requirements.

1 WHAT MAKES A GOOD REQUIREMENT?

It is not difficult to find checklists and questionnaires for ensuring the quality of requirements. Many such checklists and questionnaires of varying degrees of completeness and usefulness are printed in books on requirements engineering (e.g., [Sommerville97]), presented at conference tutorials, and published on the Web. Thus, many of the characteristics of properly specified requirements have been well known for many years, at least among certain academics, consultants, and professional requirements engineers.

Unfortunately, it is also very easy to find numerous requirements defects in almost any requirements specification one reads. Almost all requirements specifications being developed in industry today contain many poorly specified requirements. Far too many requirements are ambiguous, incomplete, inconsistent, incorrect, infeasible, unusable, and/or not verifiable (e.g., not testable). So what's the problem? Why are so many poor-quality requirements still being specified?

Although I have not seen much in the way of scientifically valid research to answer this question, anecdotal evidence abounds. Although many good requirements engineering books have been written, there are far more books published on programming languages and the latest infrastructure technologies. This would not be so bad if all requirements were specified by professional requirements engineers who had mastered the best requirements engineering books, but they aren't. In fact, the vast majority of requirements are elicited, "analyzed," and specified by managers, subject

Cite this column as follows: Donald Firesmith: "Specifying Good Requirements", in *Journal of Object Technology*, vol. 2, no. 4, July-August 2003, pp. 77-87.
http://www.jot.fm/issues/issue_2003_07/column7

matter experts (SMEs), or developers who have had little or no training in requirements engineering.

The prevailing “wisdom” seems to be that because most requirements are specified in textual English (or other natural languages) and because managers, SMEs, and developers obviously know how to read and write, then they must also intuitively know how to specify requirements. However, just as we all had to learn the rules for writing grammatically correct English, we also have to learn the rules for writing high-quality requirements. And just as not everyone who can read and write can also author a publishable book, not everyone who can write individual requirements can organize them into a high quality requirements specification. Whereas the rules for properly specifying individual requirements are relatively easy to use once you learn them, experience shows that they are also not obvious to most people who actually specify real requirements on real projects. After all, how many people are able to write good technical documents? In normal speech, we are used to relying on the give and take of conversation to ensure that the people we talk to understand what we say. And if a misunderstanding occurs, it will usually be discovered and if it’s not, there are typically no serious negative consequences.

The same cannot be said when you are specifying the requirements for a major system. Failure to correctly specify the requirements can lead to major delays, cost overruns, commercial consequences including the loss of money, property, layoffs, and even the loss of lives. That is why I am writing this column which really only summarizes information that is readily available elsewhere (if you know where to look and if you know to look for it in the first place). It is probable that much of the following information will be new to many of you, and with any luck, it will open your eyes to requirements defects that you have made in the past. Hopefully, it will also help you avoid similar mistakes in the future. And even if you have read some of the books and checklists out there, you will probably still find some new and useful material. Besides, all of us need a booster shot every now and then if we are not to fall back into our bad old habits.

2 QUESTIONS FOR ENSURING REQUIREMENTS QUALITY

Quality Characteristics

A good-quality requirement should exhibit the following characteristics that are missing from poorly specified requirements:

- Cohesiveness
- Completeness
- Consistency
- Correctness
- Currency
- Customer/User Orientation
- External Observability

- Feasibility
- Lack of Ambiguity
- Mandatory
- Metadata
- Relevance
- Usability
- Validatability
- Verifiability

Cohesiveness

Individual requirements should be cohesive, although the type of cohesion varies with the different type of requirements being specified:

- Does each requirement specify only one thing?
- Do all parts of the requirement belong together:
  - Do all parts of a data requirement involve the same data abstraction?
  - Do all parts of a functional requirement involve the same functional abstraction?
  - Do all parts of an interface requirement involve the same interface?
  - Do all parts of a quality requirement involve the same quality factor or subfactor?

Completeness

Just as an entire requirements specification should be complete and contain all relevant requirements and ancillary material (e.g., as specified in its template or content and format standard), individual requirements should also be complete. This is often a problem because subject matter experts (SMEs) who specify requirements often take certain information for granted and omit it, even though it is not obvious to other stakeholders of the requirement.

- Is each requirement self contained with no missing information?
  - Does each requirement contain all relevant information? For example, does the requirement include all relevant preconditions such as the relevant state of the application or component?
  - Does each requirement need no further amplification or clarification?
  - Does each requirement provide sufficient information to avoid ambiguity?
- If the requirement is **not** a part of the current release, then is it specified as completely and as thoroughly as is currently known?
- Is each identified "requirement" actually a single requirement and not actually multiple requirements?

  - Is the use of conjunctions (“and” and “or”) restricted to preconditions and invariants?

Consistency

Because collections of inconsistent requirements are impossible to implement, individual requirements should be consistent:

- Is each requirement externally consistent with its documented sources such as higher-level goals and requirements?
- Is each requirement externally consistent with all other related requirements of the same type or at the same requirements specification? For example, two requirements should neither be contradictory nor describe the same concepts using different words.
- Are the constituent parts of each requirement internally consistent? For example, are all parts of a compound precondition or postcondition consistent?

Correctness

Defects in requirements will naturally lead to corresponding defects in the resulting architectures, designs, and implementations. Thus, individual requirements should obviously be correct:

- Is each requirement semantically correct?
  - Does each requirement meet all or part of an actual need of its relevant stakeholder(s)?
  - Is each requirement an accurate elaboration of a documented business objective or goal?
  - Is each requirement an accurate elaboration of a higher-level requirement?
  - Do all numbers associated with each requirement have correct values?
- Is each requirement syntactically correct?
  - Does each requirement use the proper standard format (if any)?
  - Does each requirement properly use the words “shall” or “must” rather than “will” or “may”?
  - Are all words used in each requirement correctly spelled?
  - Is each textual requirement grammatically correct?

Currency

All too often, requirements specifications are not updated when requirements change. They are also frequently not updated as the architecture is produced, sometimes resulting in changes in the underlying requirements. Both of these problems make testing and maintenance much more difficult. Thus, individual requirements should not become obsolete:

- Is each requirement a specification of a current or anticipated customer or user need?

- Has each requirement not been obsoleted (e.g., due to changing business goals)?

Customer/User Orientation

Too often, requirements (especially derived requirements) are specified by developers who use their technical jargon that is not understandable to other stakeholders, especially customers, users, and managers. But individual requirements should be oriented around the needs of the customers and users if they are to be understandable and validatable:

- Is each requirement phrased in the language of the customer and user organizations?
- Does each requirement avoid the technical jargon of the development organization?

External Observability

Requirements should not unnecessarily specify the internal architecture and design of an application or component. Thus, individual requirements should only specify behavior or characteristics that are externally observable:

- Does each requirement only specify behavior and/or characteristics that are externally observable when treating the application or component as a black-box?
- Does each requirement avoid specifying any internal architecture, design, implementation, or testing decisions?
- If a requirement does specify one or more internal architecture, design, implementation, or testing decisions, is the requirement clearly identified as a constraint rather than as a pure requirement?

Feasibility

Requirements are of no value if the development team cannot implement them. Thus, individual requirements should be feasible given all relevant constraints:

- Can each requirement be implemented given the existing hardware or software technology?
- Can each requirement be implemented given the endeavor's budget?
- Can each requirement be implemented given the endeavor's schedule?
- Can each requirement be implemented given the endeavor's constraints on staffing (e.g., staff size, expertise, and experience)?
- Can each requirement be implemented given the limitations of physics, chemistry, etc?

Lack of Ambiguity

Individual requirements for an application or component should never be ambiguous. Even if the requirement is intended to be highly reusable (e.g., across a product line) and therefore general, it should still be unambiguous although it may well have precise flexibility points (e.g., it can contain parameters that must be filled in with specific values when being reused). Yet, this critical characteristic of a good requirement is often missing, resulting in requirements that are subject to misinterpretation and that are inherently not verifiable (e.g., they are untestable).

- Is each requirement clear (i.e., not vague) and precise?
- Is the meaning of each requirement objective rather than subjective?
- Is each requirement concise (i.e., without unnecessary and irrelevant information)?
- Does each requirement have only a single interpretation?
- Is the interpretation of each requirement obvious?
- Is each requirement understandable to its intended audiences?
- Will the interpretation of each requirement be the same for both those who wrote it and the members of the different audiences who will read it?
- Does each requirement use specific concrete terms?
- Does each requirement avoid the use of inherently or *potentially* ambiguous words such as:
  - Vague **subjects** that can refer to multiple things:
    - Pronouns such as “it” or “they”?
    - Demonstrative adjectives such as “this”, “these”, “that”, and “those”?
  - Vague **adjectives** that may mean different things to different readers**:**
    - Intrinsic characteristics such as “soft”, “hard”, “fast”, “slow”, “hot”, “cold”, “strong”, and “weak”?
    - Judgmental characteristics such as “easy”, “hard”, “clear”, “efficient”, “acceptable”, “adequate”, “good”, “bad”, “reasonable”, “sufficient”, “useful”, “significant”, “adequate”, and “user-friendly”?
    - Location characteristics such as “near”, “far”, and “close”?
    - Ordering adjectives such as “first”, “previous”, “next”, “following”, and “last”?
    - Temporal characteristics such as “new”, “old”, “recent”, “future”, “past”, “soon”, and “today”?
  - Vague **prepositions**:
    - Prepositions such as “above”, “below”, “in front of”, “in back of”, “over”, “under”, “high”, and “low”?
  - Vague **verbs** that are more qualitative than quantitative**:**
    - Prepositions such as “increase”, “decrease”, “maximize”, and “minimize”?
  - Subjective **phrases**:
    - “If possible”, “when cost-effective”, and “where appropriate”?
- Is each requirement specified in a quantitative manner whenever possible and practical?
- Does each requirement include all necessary assertions:

    - Invariants?
    - Preconditions?
    - Postconditions?
- Does each quality requirement go beyond merely "requiring" that the application or component exhibit the associated quality factor? Thus, it is inadequate to merely state that the application shall be portable; one must be explicit and specify how portable (e.g., maximum amount of effort to port) and portable to what (e.g., operating system, hardware platform, or infrastructure including version)?
- Does each requirement only include abbreviations, acronyms, and/or technical terms that are uniquely defined in either the associated glossary or requirements specification?
- Does each requirement only include explicit references to other documents?
- Does each diagram associated with a requirement include a legend that defines its icons and arcs?

Mandatory

Although requirements can and should be prioritized to help negotiate and schedule them, individual requirements should by their very nature be mandatory (i.e., required):

- Is each requirement essential to the success of the application or component?
- Is each requirement truly mandatory (i.e., a true requirement that must be met and implemented)?
- Is each requirement truly required by some stakeholder, typically the customer or user organization?
- Is each requirement free from *unnecessary* constraints (e.g., architecture, design, implementation, testing, and other technology decisions)?
- Does each requirement specify a "what" rather than a "how"?
- Is each requirement clearly differentiated from:
    - A "nice to have" item on someone's wish list (i.e., gold-plating)?
    - Constraints?

Metadata

Individual requirements should have metadata (i.e., attributes or annotations) that characterizes them. This metadata can include (but is not limited to) acceptance criteria, allocation, assumptions, identification, prioritization, rationale, schedule, status, and tracing information:

- Acceptance Criteria:
    - Does each requirement have associated acceptance criteria?
    - Is this acceptance criteria clear and objective?
- Allocation:

    - Is each requirement allocated to the team or individual who will implement it?
    - Is each system requirement allocated to the system architectural elements that will fulfill it?
    - Is each software requirement allocated to the software architectural elements that will fulfill it?
- Assumptions:
    - Are all significant assumptions associated with each requirement properly documented?
- Identification:
    - Does each requirement have its own unique identifier that can be used for tracing purposes?
    - Is each requirement not redundant with any other requirement at the same level of abstraction (e.g., within the same requirements specification)?
- Prioritization:
    - Is each requirement prioritized for scheduling and trade-off purposes?
    - Is the prioritization of the requirement based on the:
        - Criticality of the requirement to the customer, marketing, and user organizations?
        - Scheduling from an architectural standpoint?
        - Implementation precedence?
        - The minimization of project risk?
- Rationale:
    - Does each requirement have a reasonable rationale associated with it that justifies it being specified as a requirement?
- Schedule:
    - Is each requirement scheduled for implementation by a specific milestone or release?
    - Is this schedule based on the priority of the requirement?
- Status:
    - Does each requirement have an associated status (e.g., identified, analyzed, specified, approved, and frozen)?
    - Is this status updated as the requirements goes through its lifecycle?
- Trace:
    - Is each requirement traced to its source goal, document, and/or person?
    - Does each requirement include both forward and backward tracing information?
    - Does each system requirement include a trace back to system goals?
    - Does each system requirement include traces down to data, hardware, personnel, and software:
        - Components?
        - Requirements?

- Does each software requirement include a trace back to its system component and system requirements?
- Does each software requirement include traces down to data, hardware, personnel, and software components.

Relevance

Some identified and specified "requirements" actually turn out to be outside of the scope of the current endeavor. Thus, it is important to ensure that individual requirements are relevant:

- Is each requirement within the scope of the business, application, or component being specified? For example, is each requirement within the scope of the associated statement of work, the mission statement, and/or the vision statement?
- Does each application or component requirement avoid specifying the behavior and characteristics of the associated users?
- Does each application or component requirement avoid specifying the behavior and characteristics of the associated external systems (except for mandatory interfaces)?

Usability

Just like applications and components, requirements have many users (e.g., management, customer representatives, marketing representatives, user representatives, architects, developers, testers, support personnel) that use them for many purposes. Thus, individual requirements should be usable by their numerous stakeholders:

- Is each requirement understandable and usable by the customer representatives and user representatives who must use it for scope control and evaluation?
- Is each requirement understandable and usable by managers who must use it for scope control as well as cost, schedule, and progress metrics?
- Is each architecturally-significant requirement understandable and usable by the architects who will base the architecture on it?
- Is each requirement understandable and usable by the designers and programs who must implement it?
- Is each requirement usable (e.g., testable) by the testers who must verify and validate it?

Validatability

Individual requirements must actually fulfill the needs and desires of their primary stakeholders. Individual requirements should be validatable:

- Is it possible to ensure that each requirement is actually what the customer representatives really want and need?
- Is it possible to ensure that each requirement is actually user representatives really want and need?

- Is it possible to ensure that each requirement is actually what the marketing representatives really want and need?

Verifiability

Requirements always have sources, and it is important that requirements are consistent with them. Similarly, requirements need to be consistent with the standards, guidelines, and templates that are used in their preparation. Thus, individual requirements should be verifiable:

- Can each requirement be verified against its source?
- Can each requirement be verified against its associated standards (e.g., content and format), guidelines, and/or templates?

3 USING THE PREVIOUS QUESTIONS

Occasionally, one sees some of the previously listed questions included in checklists designed for inspecting requirements specifications. However, these questions must be asked about each individual requirement, and any non-trivial application has far too many requirements for these questions to be used that way. Not only is it impossible to spend the time necessary to methodically and manually check each individual requirement against each checklist question, it is also psychologically impossible because technical evaluators would rapidly burn out before they had evaluated more than a dozen requirements. Minimizing the number of questions to make a checklist possible would allow too many potential defects to slip through and inspectors would still have to apply each question against each requirement in the specification. Even if it were possible to overcome these obstacles, a checklist would be relatively useless because the failure box for each question would almost always end up being checked because in any reasonably-sized requirements specification, there would always be at least one requirement that would be ambiguous, incorrect, or untestable.

Another possible use for these questions would be as input to a software tool that could automate their evaluation. Such a tool could rapidly identify potential defects in the requirements specification or the presence of risk areas that need human attention. Whereas a few tools have been developed that automatically identify potential problem areas (e.g., the use of vague words and phrases), I am not aware of any tool that comes even close to being able to answer the majority of the questions listed in this column.

So if checklists are not feasible and if practical (and complete) tools are not yet available for commercial use, how then should these questions be used? Perhaps their best use is in the training of both requirements engineers who specify requirements and evaluators who technically evaluate them. By incorporating these questions into their personal mental tool set, requirements engineers will produce better requirements by avoiding the corresponding defects in the first place and technical evaluators will begin to instantly recognize violations of the implicit guidelines that these questions represent. By studying these questions, they will eventually become intuitive and automatic. The situation is similar to our leaning of natural languages. Most of us would have a hard time writing down the grammatical rules of English and haven't diagramed a sentence since middle school, but we still know when the rules are violated because the offending sentences just don't sound right. Similarly, once requirements engineers and requirements evaluators (and even managers, subject matter experts, and developers who must work with requirements) learn these questions, the underlying guidelines become internalized and poorly-specified requirements just don't sound right.

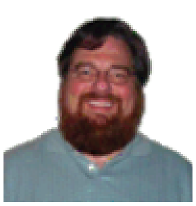
4 CONCLUSION

Hopefully by now, it is clear that specifying requirements of high quality is not trivial but it is also not rocket science either (unless you happen to be specifying requirements for NASA). The answer is not to use simplistic checklists nor is it to give up. Eliminating defects from requirements specifications is just too important. The answer is learning a few simple characteristics of high-quality requirements and then internalizing them so that the defects in poorly-specified requirements will effortlessly jump off the page. Those that specify requirements should read and study the preceding questions so that they do not insert defects into requirements specifications, and those that technically evaluate requirements should also internalize the preceding questions so that any violations will become obvious. While not a panacea, these simple questions can eliminate a great number of defects from most requirements specifications.

REFERENCES

[Sommerville97] Ian Sommerville and Pete Sawyer: *Requirements Engineering: A Good Practices Guide*, John Wiley & Sons, 1997.

About the author



Donald Firesmith is a senior member of the technical staff at the Software Engineering Institute. He has worked exclusively with object technology since 1984 and has written 5 books on the subject. He is currently writing a book on requirements engineering. Most recently, he has developed a 1000+ page informational website on the OPEN Process Framework at http://www.donald-firesmith.com. He can be reached at donald_firesmith@hotmail.com.

Steve Jones takes the question "what makes a good requirements document?" and breaks down "it depends."Â Looking for practical ways to reduce requirements defects while also improving your requirements specifications? Check out one of our business analysis training courses: Business Process Analysis.