Reducing Risks Through Proper Specification of Software Requirements

Al Florence
MITRE Corp.[1]

Requirement definition, specification, analysis, and validation and verification are some of the biggest challenges faced by software engineers. In many software requirements documentation, the requirements are ambiguous and inconsistent. They may not be uniquely identified, making them untraceable and difficult to test. In many cases, they are specified at a level too high or too low at the system or at the design level, not at the software requirements level. If these challenges are addressed, the risk of developing systems that do not satisfy requirements will be mitigated.

This article presents several examples that address the challenges faced by individuals specifying software requirements. For instance, while redeveloping legacy systems, a government agency reverse engineered the existing software requirements. With knowledge of the application domain, several teams reverse engineered and defined the requirements. They represented the user, the contractors, and the acquisition organization. This author was assigned as a consultant to guide the teams in the proper specification of requirements. The requirements were analyzed and validated against the following critical attributes:

- Complete: Requirements should be as complete as possible. They should reflect system objectives and specify the relationship between the software and the rest of the subsystems.
- Traceable: Each requirement must be traceable to some underlying source such as a system-level requirement. Each requirement should have a unique identifier allowing the software design, code, and test procedures to be precisely traced back to the requirement.
- Testable: All requirements must be testable to demonstrate that the software end product satisfies its requirements. To be testable, requirements must be specific, unambiguous, and quantitative whenever possible. Vague, general statements must be avoided.
- Consistent: Requirements must be consistent with each other; no requirement should conflict with any other requirement. Check requirements by examining all requirements in relation to each other for consistency and compatibility.
- Feasible: It must be feasible to develop software that will fulfill each software requirement. Requirements that have questionable feasibility should be analyzed during requirements analysis to prove their feasibility. If they cannot be implemented they should be eliminated.
- Uniquely Identified: Uniquely identifying each requirement is essential if requirements are to be traceable and are able to be tested. Uniqueness also helps in stating requirements in a clear and consistent fashion.
- Design Free: Software requirements should be specified at the requirements level and not at the design level. Describe the software requirement functionally from a requirement point of view, not from a software-design point of view, i.e., describe the system functions that the software must satisfy. A requirement reflects "what" the software shall accomplish while the design reflects "how" the requirement is implemented.
- Using "Shall" and Related Words: In specifications, using the word "shall" indicates a binding provision, i.e., one that must be implemented by the specification users. To state non-binding provisions, use "should" or "may." Use "will" to express a declaration of purpose (e.g., "The government will furnish ...") or to express future tense [1].

> *"Each requirement should have a unique identifier allowing the software design, code, and test procedures to be precisely traced back to the requirement."*

If projects allocate sufficient time and effort to validate requirements against these critical attributes during their definition and specification, projects will mitigate the risks associated with inadequate requirements.

Requirement Effort Examples

The following examples represent several legacy systems that were in the process of redevelopment in a modernization effort. They depict the requirements effort only and do not reflect any other life-cycle activities: design, implementation, test, or operation. These examples show some of the requirements as initially specified by the teams, followed by this author's critique of the requirements against the critical attributes, and finally the resulting re-specification.

Example 1

Initial specification: Software will not be loaded from unknown sources onto the system without first having the software tested and approved.

Critique: If the software is tested and approved, can it be loaded from unknown sources? If the source is known, can it be loaded if it has not been tested and approved? This requirement is ambiguous, which makes it difficult to implement and test. It is stated as a negative requirement making it difficult to implement. A unique identifier is not provided, which makes it difficult to trace. The word "shall" is missing.

Re-specification: 3.2.5.2 Software shall be loaded onto the operational system only after it has been tested and approved.

Example 2

Initial specification: 3.4.6.3 The system shall prevent the processing of duplicate electronic files by checking a new SDATE record. An e-mail message shall be sent.

Critique: There are two "shalls" under one requirement number. This is a vague requirement. What is the e-mail message? The requirement has design implications [SDATE record]. A requirement should specify what the data in the record are and not the name of the record. The name of

the record should appear in the design and code not in the requirement. As specified it cannot be implemented or tested.

Re-specification: 3.4.6.3 The system shall:

a. Prevent processing of duplicate electronic files by checking the date and time of the submission.
b. Send the following e-mail message:
   1. Request updated submission of date and time, if necessary, or
   2. That the processing was successful, when successful.

Example 3

Initial specification: 3.2.5.7 The system shall process two new fields (provides production count balancing info to the states) at the end of state record.

Critique: This requirement cannot be implemented or tested. It is incomplete. What are the two new fields? "Info" should be spelled out.

Re-specification: 3.2.5.7 The system shall provide the following data items (provides production count balancing information to the states) at the end of state record:

a. SDATE record.
b. YR-TO-DATE-COUNT.

Re-Critique: This rewrite has design implications [SDATE record and YR-TO-DATE-COUNT]. A requirement should specify what the data in the record are and not the name of the record.

Re-specification: 3.2.5.7 The system shall provide the following data items (provides production count balancing information to the states) at the end of state record:

a. Submission date and time.
b. Year to date totals.

Example 4

Initial specification: 3.2.5.9 All computer-resident information that is sensitive shall have system access controls to ensure that it is not improperly disclosed, modified, deleted, or rendered unavailable. Access controls shall be consistent with the information being protected and the computer system hosting the data.

Critique: Two "shalls" under one identifier thus two requirements. The requirement is vague and incomplete. What does "consistent" mean? The requirement needs to identify the sensitive information. As specified, it cannot be implemented or tested.

Re-specification: 3.2.5.9 All sensitive computer-resident information shall have system access controls consistent with the level of protection required to ensure that the information is not improperly disclosed, modified, deleted, or rendered unavailable. (Reference Sensitive Information Table 5.4.1 and Levels of Protection for Sensitive Information Table 5.4.2.)

Example 5

Initial specification: 3.3.2.1 The system shall have no single point failures.

Critique: This is an ambiguous requirement. It needs definition and/or identification of what components and/or functions the "no single point failures" applies. As specified it cannot be implemented or tested.

Re-specification: 3.3.2.1 The following system components shall have no single point failure:

a. Host servers.
b. Networks.
c. Network routers.
d. Access servers.
e. Hubs.
f. Switches.
g. Firewalls.
h. Storage devices.

Example 6

Initial specification: 3.2.7.1 The system shall purge state control records and files that are older than the operator or technical user-specified retention period.

Critique: This requirement cannot be implemented or tested as stated. It is vague without specifying the retention period or providing a reference as to where the information can be obtained.

Re-specification: 3.2.7.1 The system shall purge state control records and files that are older than the retention period that is input into the system by either:

a. The operator.
b. The technical user.

Example 7

Initial specification: 3.2.6.3 The system shall receive and process state data from the State Processing Subsystem. The system shall provide maintenance of the state data files and generate various reports.

Critique: There are two "shalls" under one requirement number and multiple requirements in the specification. The word "process" in the first "shall" is vague. The requirement needs to define the processing required. The second "shall" does not provide for valid requirements; they cannot be implemented or tested as stated. The requirement needs identification of type/amount of maintenance required. The term "various reports" is ambiguous.

Re-specification: 3.2.6.3 The system shall receive:

a. Production data that contain data from multiple states.
b. Financial state data for one or more states, extracted by the State Processing Subsystem.

3.2.6.4 The system shall parse multi-state data to respective state files.

3.2.6.5 The system shall display a summary screen reporting the results of processing for each state containing:

a. State totals.
b. State generic totals, and
c. State unformatted totals

Example 8

Initial specification: 3.2.7.1 The system shall not prevent individuals from entering the year for which they intend the payment, but shall provide a checkpoint for them to ensure that they are not making a mistake in entering the correct year.

Critique: This is a negative requirement; negative requirements should not be specified. They cannot be implemented. A requirement should have all conditions that are required. If conditions are not required they will not be implemented. There are two "shalls" under one requirement number. I suggest that this requirement be structured in a positive fashion.

Re-specification: 3.2.7.1 The system shall:

a. Allow individuals to enter the payment year.
b. Provide a checkpoint to ensure that individuals enter the correct payment year.

Example 9

Initial specification: 3.2.7.3 After the system receives the validation file, the system shall:

- Notify the individual about acceptance or rejection.
- The acceptance file must contain the name control and ZIP code of the approved individual.
- Rejected validation request must include the Reason Code.

Critique: The second and third bullets do not make sense, try to read them without the first bullet:

- The system shall the acceptance file must...
- The system shall rejected validation…

The requirement uses a "shall" and a "must." Unique identifiers are not provided. The requirement uses bullets, which should not be used in specifying requirements. Bullets cannot be traced. This requirement is ambiguous and cannot be implemented or tested.

Re-specification: 3.2.7.3 When the system receives a validation file the system shall:

a. Reject the file if it does not contain the approved individual's:

1. Name.
2. ZIP code.

b. Notify the individual about acceptance or rejection with a reason code. (Reference Reason Codes Table 5.4.8.)

Example 10

Initial specification: 3.2.8.2 The enrollment process shall take from one (1) to ten (10) calendar days to complete for all enrollment types.

3.2.8.3 The enrollment process shall take no more than three (3) days to complete for:

a. Credit enrollment.
b. Note enrollment.

Critique: These two requirements are inconsistent and in conflict with each other.

Re-specification: 3.2.8.2 The enrollment process shall take:

a. One (1) to three (3) calendar days to complete for:
   1. Credit enrollment.
   2. Note enrollment.
b. One (1) to ten (10) calendar days to complete for all other enrollment types.

Example 11

Initial specification: 3.2.8.6 When doing calculations the software shall produce correct results.

Critique: Really? This is not a requirement. This type of requirements should not be specified. It should be deleted.

Re-specification: Requirement deleted.

Conclusion

The teams identified more than 1,000 requirements. The issues with their initial specification represented the entire spectrum of the critical attributes: complete, traceable, testable, consistent, feasible, uniquely identified, and design free. The teams were receptive to the critiques, resolved issues, and implemented the recommendations willingly. The requirements resulting from this effort were reviewed with senior management, accepted as specified, baselined, and allocated to development teams for implementation.

If sufficient time and proper effort is taken to validate requirements against critical attributes during their definition and specification, software projects will mitigate the risks associated with requirements and will considerably improve their probability of success. It is a well-known fact that if this is not done, projects pay the consequences during implementation and integration and test, not to mention during operation.◆

Reference

1. Military Standard Specification Practices. MIL-STD-490A. U.S. Department of Defense, 4 June 1985.

Note

1. The views expressed are those of the author and do not reflect the official policy or position of the MITRE Corporation.

Suggested Readings

1. IEEE Std. 830-1998. IEEE Recommended Practices for Software Requirements Specifications. IEEE Computer Society. 20 Oct. 1998.
2. Cook, David A., and Les Dupaix. "The Requirements for Good Requirements." Software Technology Conference Proceedings. Mar. 2001.

About the Author

Al Florence has been employed at major technology firms and is currently at the MITRE Corporation. He has been involved in all phases of the life cycle as a manager and a developer, from concept to retirement in different engineering disciplines, including systems, software, test, configuration management, quality assurance, and process improvement as a developer and as a manager. His work has involved many diversified projects: spacecraft, aircraft, missiles, weapon systems, particle accelerators, simulation, and information systems. He has been involved with the definition, specification, and validation of requirements on many of these projects. Florence has a bachelor's of science degree in mathematics and physics from the University of New Mexico, and he did graduate work in computer science at the University of California in Los Angeles and at the University of Southern California.

The MITRE Corporation
7515 Colshire Drive
McLean,VA 22102-3481
Phone: (703) 883-7476
Fax: (703) 883-1339
E-mail: florence@mitre.org