

# Guidelines for Functional Specifications

## 1. Overview

This document gives guidelines for the creation of Functional Specification documents. It describes the content and qualities of a good Functional Specification (FS). It is to be used in conjunction with the "Template for Functional Specifications" document.

The recommendations in this document are aimed at specifying requirements of software to be developed. An FS should be created to guide and document the creation of all software developed inside of Quarterdeck's development department and on behalf of Quarterdeck by outside developers. The creation of an FS for already-developed software, either developed or acquired by Quarterdeck, should also be considered, although it could prove counterproductive. It should also be noted that Functional Specifications can be applied to assist in the selection of in-house and commercial software products.

### 1.1 Credits

Both this document and the accompanying template are refinements of the "IEEE Recommended Practice for Software Requirements Specifications" document released by the IEEE Standards Board as IEEE Std 830-1993. The IEEE document was reviewed and discussed by an inter-departmental group of Quarterdeck employees. Included in this group were Brian Breidenbach, Suzanne Dickson, Phil Glosserman, Anthony Godfrey, Larry Mayer, Mark Radcliffe, Dan Spear, Dan Trivison, Gene Venable, and Mike Wacker. The results of those discussions have been authored into this document and the "Template for Functional Specifications" document by Larry Mayer, Director of Development at Quarterdeck Corporation.

## 2. Nature of the Functional Specification

The FS is a specification for a particular software product, program, or set of programs that performs certain functions in a specific environment.

The basic issues that the FS addresses are the following:

- a) *Functionality.* What is the software supposed to do?
- b) *External interfaces.* How does the software interact with people, the system's hardware, other hardware, and other software?
- c) *Performance.* What is the speed, availability, response time, recovery time of various software functions, etc.?
- d) *Attributes.* What are the portability, correctness, maintainability, security, etc. considerations?
- e) *Design constraints imposed on an implementation.* Are there any required standards in effect, implementation language or tools, resource limits, operating environment(s) etc.?

The FS writer(s) should avoid placing either design or project requirements in the FS.

### 2.1 The purpose of the FS

The FS is created to increase the efficiency and coordination of the product development process. It does this by

- a) Simplifying and formalizing the dissemination of information about the project
- b) Creating a more accurate means of evaluating the progress of software development
- c) Minimizing rework, drifting, and marginally useful work for all those who use the document
- d) Improving the quality of
  - 1) up-front estimates of the time and money it will take to create the product
  - 2) the software
  - 3) testing done on the software
  - 4) end-user documentation
  - 5) release planning for the software
- e) Allowing people new or external to the product to gain an accurate understanding of the product

### 2.2 Usage of the FS

Following is a list of people who create and/or use the FS.

#### 2.2.1 Product Manager / Development Coordinator

These people are responsible for coordinating the creation of the whole product, not just the software. They need to see the FS prior to signing-off on the project. They must verify that the FS fulfills the needs of the Marketing Requirements Document (MRD). Additionally, they should help guide and review the user interface design and product prototype to make sure it matches with market tastes, is synergistic with other products in the product line, and foresees future directions of the product. These people will typically be responsible for approving any changes to the FS, in particular those changes that affect the software development schedule.

#### 2.2.2 Program Manager

The Program Manager manages the development team. This person has the primary responsibility for creating the FS. They will work with the Product Manager / Development Coordinator and the Developers to create the FS and to verify that the FS fulfills the MRD and is technically accurate. As the FS contains user interface designs and implementation details, the Program Manager will have to organize and guide the development team towards completing the research and prototyping necessary to create the FS. The Program Manager will use the FS to estimate the staffing requirements for writing the software and to create

a Gantt chart for developing the product. The Program Manager should use the FS to identify the software tools and hardware that must be used to develop the product.

### **2.2.3 Developers**

The Developers will have to flesh out the FS. They will verify the feasibility of implementing the product and will work with the Program Manager to create a Development Gantt chart from the FS. The developers must use the FS to know what they should be designing and coding. The FS should contain the user interface and feature set of the program. The FS should contain enough detail for the developers to begin the design (including programming interfaces) of the project and thereafter to begin coding immediately. The developers must be given the Gantt chart for the project shortly after they start coding, so they will understand the scope of the work and determine whether they are on schedule.

### **2.2.4 Localization Manager**

The Localization Manager must verify that the FS takes into account localization requirements. The Localization Manager must use the FS to estimate costs and staffing requirements to localize the product. The FS should be provided to the translators and should be sufficient to guide them on translation issues with minimal further input from Development.

### **2.2.5 Documentation Manager**

The Documentation Manager will have to estimate the cost and time required to document the product based on the FS. The FS should be provided to the document writer(s) and should be sufficient to allow them to document the product with minimal further input from Development. The documentation of a product must describe the overall purpose of the product and the details for how to use the product. The MRD should contain a discussion of the overall purpose of the product. The FS should contain the details for how to use the product. This means that the FS must contain the user interface and feature set of the program. It must detail the error conditions that might arise and must explain the product in sufficient detail to allow for the documentation to be written.

### **2.2.6 Quality Assurance Manager**

The Quality Assurance Manager must verify that the FS is sufficient to create a test plan. The FS should be provided to the testers. The FS should be sufficient for the testers to verify that the product behaves exactly as it should with minimal further input from Development. This means that every execution path that should be tested should be identified in the FS. Also, every input that the tester might give should be detailed and every output that is generated by any input should be detailed.

### **2.2.7 Technical Support Manager**

The Technical Support Manager and tech support staff should be consulted during the creation of the FS to provide input on creating features that minimize support requirements. The FS may prove useful in training the Technical Support staff on the product, although the end-user documentation might prove sufficient.

### 3. Environment of the Functional Specification

The FS is only one of the documents created during the software development process. The key document needed to create an FS is the Marketing Requirements Document (MRD).

#### 3.1 Relationship between the MRD and the FS

The MRD lists out important requirements for the product as a whole, including:

- a) *Mission statement*: an indication as to why we are making this product;
- b) *Future Directions*: the long term vision and future market considerations for the product
- c) *System Requirements*: the platforms on which the product should run and the hardware and software that must be tested for compatibility;
- d) *Localization plans*: the plans and constraints imposed by the internationalization of the product;
- e) *Implementation constraints*: any constraints on how the product may be implemented;
- f) *Licensing and OEM issues*: any plans for exploiting the technology through licensing or OEM agreements that might affect the software;
- g) *Serial number and registration issues*: any issues regarding registering the software that might affect the software.

The MRD also describes the functions and features of the software, including for each function and/or feature:

- a) *Implementation priority*: how vital the item is to the success of the product;
- b) *Look and feel*: a suggestion for the interface for the item;
- c) *Performance requirements*: a guideline for the speed of execution of the item;

The contents of the MRD are thus vital to the creation of the FS. An incomplete or inaccurate or unclear MRD will likely create an FS with similar problems.

#### 3.2 Prototypes and the FS

Prototypes will usually be created for a product and will be included in its FS. Prototypes perform a number of functions.

- a) The people planning to exploit the software may be more likely to view the prototype and react to it than to read the text in the FS and react to it. Thus, the prototype provides quick feedback.
- b) A prototype is needed for usability testing to get widespread feedback on the interface of the product before significant coding is done.
- c) The prototype displays unanticipated aspects of the systems behavior. Thus, it produces not only answers but also new questions. This helps reach closure on the FS.
- d) An FS based on a prototype tends to undergo less change during development, thus shortening development time.

Thus, prototypes of the software should be completed very early in the product creation cycle. In fact, the prototype is likely to be needed for the FS to be completed. However, the FS must be fairly robust for the prototype to be designed. This means that the prototype and FS must be created and used in tandem.

#### 3.3 Scope of the FS

Since the FS has a specific role to play in the software development process, FS writer(s) should be careful not to go beyond the bounds of that role. This means the FS

- a) Should correctly define all of the functions performed by the software and all of the constraints on the execution of those functions. A software constraint may exist because of the nature of the task to be solved or because of a special characteristic of the project.
- b) Should not describe any design or implementation details that are not apparent to the user of the software. Internal design or implementation details should be described in the design stage of the project.

### **3.3.1 Embedding project requirements in the FS**

The FS should address the software product, not the process of producing the software product. Project requirements represent an understanding between customer and supplier about contractual matters pertaining to production of software and thus should not be included in the FS. These normally include such items as

- a) Cost
- b) Delivery schedules
- c) Reporting procedures
- d) Software development methods
- e) Quality assurance
- f) Validation and verification criteria
- g) Acceptance procedures

Project requirements may be specified in other documents, typically in a software development plan, a software quality assurance plan, or a statement of work.

### **3.3.2 Embedding design in the FS**

A requirement in the FS specifies an externally visible function or attribute of a system. A design describes a particular subcomponent of a system and/or its interfaces with other subcomponents. The FS writer(s) should clearly distinguish between identifying required design constraints and projecting a specific design. Although every requirement in the FS limits design alternatives, this does not mean that every requirement is design.

The FS should specify what functions are to be performed on what input to produce what output at what location for whom. The FS should focus on the services to be performed, and should not normally specify design items such as the following:

- a) Partitioning the software into modules
- b) Allocating functions to the modules
- c) Describing the flow of information or control between modules
- d) Choosing data structures

#### **3.3.2.1 Necessary design requirements**

In special cases some requirements may severely restrict the design. For example, future product direction may reflect directly into design such as the need to

- a) Keep certain functions in separate modules (such as Operating System-dependent code)
- b) Create a publishable programming interface between some areas of the program
- c) Separate code and data that must be localized

Examples of valid design constraints are physical requirements, performance requirements, software development standards, and software quality assurance standards.

Therefore, the requirements should be stated from a pure external viewpoint.

### **3.4 Other documents and the FS**

There are a number of other plans and documents that rely on the FS. The process of creating a test plan for use by Quality Assurance will rely heavily on the FS. The end-user documentation will also be based on the FS. Any design documents created by the development team will be guided by the FS.

### **3.5 FS evolution**

The FS will typically need to evolve as the development of the software product progresses. It may be impossible to specify some details at the time the project is initiated. For example, it may be impossible to define all of the screen formats for an interactive program during the requirements phase. Additional changes may ensue as deficiencies, shortcomings, and inaccuracies are discovered in the FS.

Two major considerations in this process are the following:

- a) Requirements should be specified as completely and thoroughly as is known at the time, even if evolutionary revisions can be foreseen as inevitable. The fact that they are incomplete should be noted.
- b) A formal change process should be initiated to identify, control, track, and report projected changes. Approved changes in requirements should be incorporated in the FS in such a way as to
  - 1) Provide an accurate and complete audit trail of changes
  - 2) Permit the review of current and superseded portions of the FS

### **3.5.1 FS change control**

The mechanism for controlling changes to the FS is something that will have to be agreed upon by the people authoring and using the FS. A reasonable approach would be to allow the Program Manager to control modifications to the FS during the early stages of development unless those changes

- a) Conflict with higher level documents (such as the MRD)
- b) Affect the scheduling of the software milestones

Any changes that have the above effect would have to be approved by the Product Manager and/or the Development Coordinator.

Once other groups (such as QA and documentation) have begun significant work based on the FS, any changes to the FS may require rework on their part. Thus, it would be appropriate during the middle stages of development to require any changes to the FS to be approved by the Product Manager and/or Development Coordinator.

## 4. Characteristics of a Good FS

An FS should be

- a) Correct
- b) Unambiguous
- c) Complete
- d) Consistent
- e) Ranked for importance and/or stability
- f) Verifiable
- g) Modifiable
- h) Traceable

### 4.1 **Correct**

An FS is correct if, and only if, every requirement stated therein is one that the software shall meet.

There is no tool or procedure that assures correctness. The FS should be compared with any applicable superior specification, such as an MRD, with other project documentation, and with other applicable standards, to assure that it agrees. The author(s) of the MRD can determine if the FS correctly reflects the actual needs. The Quality Assurance department or other internal testers can verify that the resulting software acts exactly as is described in the FS. Any deviation should be considered an error in either the software or the FS.

### 4.2 **Unambiguous**

An FS is unambiguous if, and only if, every requirement stated therein has only one interpretation. As a minimum, this requires that each characteristic of the final product be described using a single unique term. In cases where a term used in a particular context could have multiple meanings, the term should be included in a glossary where its meaning is made more specific.

An FS is an important part of the requirements process of the software life cycle and likely to be used in design, implementation, project monitoring, verification and validation, and in training. The FS should be unambiguous both to those who create it and to those who use it. However, these groups often do not have the same background and therefore do not tend to describe software requirements the same way. Representations that improve the requirements specification for the developer may be counterproductive in that they diminish understanding to the user and vice versa.

Formal “requirements specification languages” and processors have been created to help avoid ambiguity in specification documents. Such tools are not likely to be of use at Quarterdeck. Instead, we will be writing the FS in English, a natural language which is inherently ambiguous. At least one of the authors of the FS must therefore be skilled at writing unambiguous prose.

### 4.3 **Complete**

An FS is complete if, and only if, it includes the following elements:

- a) *All significant requirements, whether relating to functionality, performance, design constraints, attributes, or external interfaces.* In particular any external requirements placed by a system specification should be acknowledged and treated.
- b) *Definition of the responses of the software to all input in all situations.* Every bit of text, graphics, or sound that can be output by the software should be described in the FS. Note that it is important to specify the responses to both valid and invalid input values.
- c) *Full labels and references to all figures, tables, and diagrams in the FS and definition of all terms and units of measure.*

#### 4.3.1 Use of TBDs

Any FS that uses the phrase “to be determined” (TBD) is not a complete FS. The TBD, however, is likely to be necessary in the earliest stages of the FS. However, any TBD must be accompanied by

- a) A description of the conditions causing the TBD (for example, why an answer is not known) so that the situation can be resolved
- b) A description of what must be done to eliminate the TBD, who is responsible for its elimination, and by when it must be eliminated

#### 4.4 Consistent

Consistency refers to internal consistency. If an FS does not agree with some higher level document, such as a system requirements specification, then it is not correct (see 2.2.1).

An FS is internally consistent if, and only if, no subset of individual requirements described in it conflict. There are three types of likely conflicts in an FS:

- a) The specified characteristic of a user-interface component may conflict. For example,
  - 1) The format of an output report may be described in one place as tabular but in another as textual.
  - 2) One requirement may state that all error messages shall be green while another states that all error messages shall be blue.
- b) There may be logical or temporal conflict between two specified actions. For example,
  - 1) One requirement may specify that the program will add two inputs and another may specify that the program will multiply them.
  - 2) One requirement may state that “A” must always follow “B,” while another requires that “A and B” occur simultaneously.
- c) Two or more requirements may describe the same thing but use different terms for that thing. For example, a program’s request for a user input may be called a “prompt” in one requirement and a “cue” in another. The use of standard terminology and definitions promotes consistency.

#### 4.5 Ranked for importance and/or stability

In every development process, choices are made about how much effort to spend on each element of the program, and when in the development process resources should be expended on the element. Some features will be carefully crafted and optimized to perfection, while others will be implemented in a quick-and-dirty fashion, and others might not get completed at all. Some features will be attacked early in the development process while others will be begun late.

##### 4.5.1 Ranking for importance

To guide the developers on which of the software requirements are the most essential, an FS could rank each requirement by degree of necessity. The requirements could be ranked as follows:

- a) *Essential*. Implies that the software will not be acceptable unless these requirements are provided in an agreed manner.
- b) *Conditional*. Implies that these are requirements that would enhance the software product, but would not make it unacceptable if they are absent.
- c) *Optional*. Implies a class of functions that may or may not be worthwhile. This guides the developer to research the item and evaluate options for its implementation.

##### 4.5.2 Ranking for stability

Another way to help guide the developers is to identify those requirements that might change before the end of the development cycle. In some cases, the people who are setting the requirements for the product might know of forthcoming events that might impact those requirements. For example, if an industry standard is being developed that would affect the protocols or programming interfaces of the product, the

developers should be warned that those parts of the software might have their specifications changed in the middle of the development process.

#### **4.6 Verifiable**

An FS is verifiable if, and only if, every requirement stated therein is verifiable. A requirement is verifiable if, and only if, there exists some finite cost-effective process with which a person or machine can check that the software product meets the requirement. In general any ambiguous requirement is not verifiable.

Nonverifiable requirements include statements such as “works well,” “good human interface,” and “shall usually happen.” These requirements cannot be verified because it is impossible to define the terms “good,” “well,” or “usually.” The statement that “the program shall never enter an infinite loop” is nonverifiable because the testing of this quality is theoretically impossible.

An example of a verifiable statement is

*Output of the program shall be produced within 20 s of event x 60% of the time; and shall be produced within 30 s of event x 100% of the time.*

This statement can be verified because it uses concrete terms and measurable quantities.

If a method cannot be devised to determine whether the software meets a particular requirement, then that requirement should be removed or revised.

#### **4.7 Modifiable**

An FS is modifiable if, and only if, its structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining the structure and style. Modifiability generally requires an FS to

- a) Have a coherent and easy-to-use organization with a table of contents, an index, and explicit cross-referencing
- b) Not be redundant; that is, the same requirement should only appear in one place in the FS
- c) Express each requirement separately, rather than intermixed with other requirements

Redundancy itself is not an error, but it can easily lead to errors. Redundancy can occasionally help to make an FS more readable, but a problem can arise when the redundant document is updated. For instance, a requirement may be altered in only one of the places where it appears. The FS then becomes inconsistent. Whenever redundancy is necessary, the FS should include explicit cross-references to make it modifiable.

#### **4.8 Traceable**

An FS is traceable if the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation. Two types of traceability are recommended.

- a) Backward traceability (that is, to previous stages of development). This depends upon each requirement explicitly referencing its source in earlier documents.
- b) Forward traceability (that is, to all documents spawned by the FS). This depends upon each requirement in the FS having a unique name or reference number.

The forward traceability of the FS is especially important when the software product enters the operation and maintenance phase. As code and design documents are modified, it is essential to be able to ascertain the complete set of requirements that may be affected by those modifications.