NFRs: Fact or Fiction?

Janet E. Burge and David C. Brown

Artificial Intelligence in Design Research Group
Computer Science Department,
WPI, Worcester, MA 01609, USA

jburge@cs.wpi.edu, dcb@cs.wpi.edu

1 Introduction

Customer requirements are the foundation upon which a software system is built. These requirements, derived from the customers' needs and desires, are used to both guide the development of the system and to determine if the completed system is what the customer requested. Because of its importance, requirement specification has become a research area known as Requirements Engineering (RE) both in Software Engineering [Zave, 1997; Nuseibeh & Easterbrook, 2000] and Systems Engineering [Dorfman, 1990; Chandrasekaran & Kaindl, 1996].

The primary goal of requirements engineering is to capture and represent system requirements so that they can be traced through to both implementation and testing to ensure that the resulting system does what the customer has requested. Requirements are commonly broken into two types: functional requirements (FRs) that correspond to desired functional capabilities of the system and non-functional requirements (NFRs) that describe desirable overall properties that the system must have (such as being "cost-effective" or "user-friendly"). NFRs generally are not directly related to specific system components and often involve aggregate system behavior [Manola, 1999].

The primary focus of requirements engineering has been on the functional requirements: ensuring that the necessary functionality of the system is delivered to the user. The NFRs, however, are still important since they contribute to the overall quality of the resulting system. RE research has treated NFRs as something separate and distinct from the functional requirements. This raises several questions:

1. What does "functional" really mean?
2. Are NFRs really non-functional or are they a form of functional requirements?
3. Are NFRs really requirements?
4. Can NFRs be represented as functional requirements?
5. If an NFR cannot be represented as a functional requirement, how do you ensure that it is met?
6. Can methods of representing and testing functional requirements be applied to NFRs?

In this paper, we examine the first four questions. First, we begin by discussing the definitions of "functional requirement", "non-functional requirement", "software requirement", and "function." We then examine NFRs to see if the reasons that normally distinguish them from functional requirements are valid in the context of the definition of function.

2 Definitions

In many cases, terms such as function, functional requirements, and non-functional requirements are used without being defined. Before discussing and comparing them, it is useful to describe what they are and give some examples. The following sections present definitions of functional requirements and non-functional requirements.

2.1 Functional Requirements

Functional requirements (FRs) define what the system does [Yeh & Ng, 1990; Thayer, 1990]. Roman, [1985], describes FRs as capturing "the nature of the interaction between the component and its environment." These are both somewhat vague definitions since there are likely to be many things that a system "does," not all of which may be needed by the user. Since the FRs are required things, they must be desired by the customer, and intended by the designer.

If the system being designed was an air traffic control system, a functional requirement might be that aircraft positions must be reported with an accuracy of within two nautical miles. The requirement does not specify how to achieve that accuracy, only that the accuracy is necessary. Another important feature of the requirement is that it is "testable" (i.e. it is possible to demonstrate that the requirement has been met).

2.2 Non-Functional Requirements

While functional requirements describe what the system or device should do, non-functional requirements are concerned with how the system or device should accomplish that function given "the constraints of a non-ideal world" [Thayer & Dorfman, 1990]. Roman [1985] describes NFRs as restricting the types of solutions under consideration. Roman also refers to the NFRs as "constraints." For example, there are likely to be any number of ways in which a given functional requirement can be met. NFRs provide guidance on differentiating between these solutions. For example, if the NFR concerns performance, solutions that are faster will be preferred and solutions with a poor performance should be rejected.

NFRs are typically described as *attributes* of the system or device that contribute to the overall *quality* of the product. These attributes are not confined to one portion of the product's functionality. In a software system, these include the "ilities" such as reliability, security, scalability, extensibility, manageability, maintainability, interoperability, composability, evolvability, survivability, affordability, understandability, and agility [Fillman, 1998], as well as other system-wide properties,

such as performance, security, availability, modifiability, adaptability, nomadicity, survivability, evolvability, and responsiveness [Chung & Yu, 1998]. Other types of NFRs include quality of service parameters (QoS) such as performance parameters [Manola, 1999] and system attributes (also referred to as properties) such as “ease-of-use” that apply to the system in general but cannot be phrased as a task that the system performs [Larman, 1997]. Roman views NFRs as constraints (referring to them as both non-functional requirements and non-functional constraints). Roman lists several constraint categories: interface constraints, performance constraints, operating constraints (such as personnel availability), life-cycle constraints (maintainability, enhanceability, etc.), economic constraints (such as cost), and political constraints (such as avoiding use of a competitor’s device).

NFRs tend to be very general and are likely to be desirable to varying degrees in different systems. For example, in a air traffic control system, safety may be extremely important, while in a banking system there are few safety concerns. There are often conflicts between the NFRs that result in tradeoffs being made. Typical examples would be trading off cost versus safety or resources used versus performance.

3 Function

Before examining more closely what makes a requirement functional, we will first look at the meaning of `function'. Researchers have studied the definition and representation of function for a variety of engineered artifacts, and have also studied how to reason using those representations [Umeda & Tomiyama, 1997].

Chandrasekaran and Josephson [1997; 2000] define function by viewing it as a set of effects that an entity has on its environment. These effects must be desired by some agent in order for them to have meaning as a function. Otherwise they will be spurious, just as a click's ticking doesn't help someone know the time.

Functions are usually utilized by agents in the environment (e.g., users) to achieve goals. Typically designers intend effects and users desire them. Effects may be behaviors or properties of the functioning entity: e.g., a clock's second hand moving, or chair's flat seat.

Chandrasekaran and Josephson refer to the “mode of deployment” as the way that causal interactions between the entity and the environment are instantiated. Some things only function when in certain physical relationships to the environment (e.g., “plugged in”) or when the environment acts on them in some way (e.g., “press button to start”).

If a set of behavioral constraints on the environment are satisfied when the entity is correctly deployed then that entity can be said to play a “role” in that environment. Behavioral constraints are any constraints on the behavior of the environment, such as a required voltage being achieved, or some condition producing some action.

If the entity “plays a role” in the environment *and* that role is desired by some agent in the environment, then the entity has (or performs) a function. The actual function is defined by the constraints being satisfied. The agent desiring that role can be thought of as the “user”, but it might also be the “designer”.

This view of functionality is highly compatible with Roman's [1985] statement that functional requirements involve the interaction between the object and its environment.

4 Software Requirements

When designing software, requirements fall into a sub-field of Software Engineering known as Requirements Engineering. There are several RE areas where research is being done [Nuseibeh & Easterbrook, 2000]: eliciting requirements, modeling and analyzing requirements, communicating the requirements, reaching agreement on requirements, and evolving requirements. There is also work done specifically on NFRs. For example, the NFR Framework [Chung, et. al., 1995] represents the NFRs as goals that must be satisfied by the system. The system design consists of a goal-graph giving the NFRs, alternative ways of satisfying them, and claims for and against these alternatives.

One aspect of software engineering that differs from the design and manufacture of other systems is that one of the largest costs is not the development, but instead is the maintenance. That is because software systems are mutable and can change over time, either due to finding and correcting defects in the original system or by responding to the customer’s changing needs.

One way to keep software costs down is to reuse or modify existing systems for new needs. This makes the ability to keep track of requirements even more crucial. Requirements for the new system (both FR and NFR) will consist of some or all the requirements met by the reused system (if not, then it is likely that the cost saved by reuse and modification will be minimal) plus some additional ones. It is important to ensure that the modifications result in the system meeting the new requirements but do not cause the system to violate any prior requirements.

5 Software Environments

If functional requirements can be described by their effect on the environment [Roman, 1985], then in order to examine functional requirements we must first look at what environment or environments are important for software. With its long lifecycle, software exists in a number of different environments either defined by the sequence of time (first *development*, then *operational*) or by who is interacting with the software (the *user* or the *developer/maintainer*). There are many ways that the environments could be delineated, but for this paper we will look at three different environments: development, operational, and maintenance.

The development environment may have its own requirements (functional and non-functional) that are not directly related to the task that the system performs. A typical development requirement would be one that requires the development team to write the

software in a particular language. For example, in the early 1980s, the Department of Defense mandated that government software be written in Ada. There may also be requirements on the hardware platform, especially if the software is going to be run on equipment already owned by the customer.

The operational environment is the one where most customer requirements apply. Most requirements defined as functional apply to this environment – these requirements specify what services the software must provide. An example of this is the aircraft tracking accuracy requirement described earlier in this paper.

The maintenance environment must also come into consideration. This may not always be the same as the development environment. For example, on some government contracts, the Operations and Maintenance (OEM) contractor may be a different company than the one who originally developed the software. There may be requirements that affect maintenance specifically. One such requirement would be that the developing contractor is required to train government personnel on how to build and install the software after it has been delivered.

6 Differentiation

So what differentiates an NFR from an FR? A wide variety of different types of NFRs have been described – what do they have in common?

There are a number of characteristics (or missing characteristics) of NFRs that are used to distinguish them from FRs:

- An NFR does not describe something that the system "does," i.e. they are not requirements that describe a function that the system performs.
- NFRs do not relate to a specific system component, instead they "cross-cut" functionality
- NFRs can not be evaluated without looking at the system as a whole – this particularly applies to NFRs that involve end-to-end performance

The question is: do these characteristics automatically mean that the requirement is non-functional? The following sections look at these aspects of NFRs in more detail.

6.1 NFRs and System Function

One way that NFRs are distinguished from FRs is that they do not explicitly relate to specific functions of the system. However, the truth of this statement depends on how the functions are defined. In particular, if the functional requirements define how the system must interact with its environment, then all of the different environments the system exists in must be considered.

Several of the "ilities" refer to the development (or maintenance) of the system rather than its deployment. These include scalability, maintainability, extensibility,

composability, and affordability. When the environment under consideration is development or maintenance, these requirements do have an effect.

For example, consider affordability. The affordability of a system has little or no effect on the operational environment (assuming that no other requirements were compromised in order to complete the project at the desired price) but it *does* have an effect on the development environment by putting cost constraints on the building of the system. Similarly, maintainability may not effect the system as viewed by a user during deployment but designing a system to be maintainable will clearly have an effect on the maintenance environment by making it less difficult and costly to perform system updates or fix any problems encountered.

By changing the environment from which the requirements are viewed, many of the NFRs can be considered functional in a specific environment.

6.2 "Cross-Cutting" Functionality

The reason that typical NFRs "cross-cut" functionality is that they are very general and could apply to any system, even though the means of realizing them is likely to vary widely. Many of these NFRs are not actually requirements themselves, but are really a convenient way of grouping more detailed functional requirements under a more abstract name.

An example of this would be the NFR "safety." Generally there are functional requirements that indicate how the system must function in order to be considered safe. Simply stating that safety is required is not very precise and the definition of safety will vary from system to system. Safety for an air traffic control system would require having alarms go off if planes were flying too close together, for example. Safety would also involve the functional requirement on accuracy described earlier – if the radar is not accurate then the alarms will not sound properly. Safety for an automobile may mean that it can withstand front impacts of up to 30 miles-per-hour without harming its occupant (and probably many other requirements). It will also involve requirements on what type of safety equipment is installed in the vehicle (such as air bags and seat belts). These requirements can all be grouped under the heading of safety for the system or device in question.

If safety is important for the system, then it is important to have a way to ensure that the safety requirement is met, regardless of whether that requirement is considered to be non-functional. Saying that the system "shall be safe" is not useful unless there is a set of testable functional requirements that together ensure the safety of the system.

6.3 NFR Evaluation

NFRs are often described as "cross-cutting" the functionality. It is believed that in some cases, such as performance, it may not be possible to truly determine if a requirement has been met without looking at the system as a whole. This is certainly true of some

performance parameters, especially if viewed as end-to-end performance. But does this make the requirements non-functional? If there is a requirement for a system to have a specific throughput then that requirement *is* functional, albeit not one that can be measured by summing up the performance of individual pieces of the system. If the requirement is not measurable, then it is not possible to determine if the requirement has been met or not.

For other overall qualities, such as "user friendliness," there are individual decisions made that contribute to the requirement. In fact, if the "user friendly" requirement cannot be broken down into what constitutes "user friendly" and what does not, it is impossible to look at what that means. If being "user friendly" means that the system can be operated with minimal typed input, then that is a requirement that can be applied to every portion of the user interface. Plus if we return to the definition of function as the effect on the environment, then being "user friendly" does effect the environment of user interactions.

7 Requirements vs. Preferences

If an NFR can be quantified then it can be converted into a functional requirement. For example, the NFR of affordability can be translated into specific requirements on the cost of the system if the range of acceptable costs can be made known. In some cases, however, this is not possible. For example, if the requirement was that the system be "as cheap as possible" there would not be a way to translate that into a specific dollar amount that can be tested. In this case, there may be a range of acceptable costs but the requirement cannot be expressed in terms of a specific one. The only way to determine if the system was "as cheap as possible" would be to compare the costs of alternative designs. Even such a comparison is no guarantee unless *all* possible alternatives are known.

In this case, the *degree* to which the requirement is met may vary. This is different from more concrete, functional, requirements where testing the requirement results in a pass/fail answer. Indicating that the system should be "as cheap as possible" is *not* a requirement. If it were a requirement then there would need to be a way to ensure that it has or has not been met. Instead, this is a *preference* indicating that when given a choice between a solution with a higher cost and one with a lower cost, the lower cost alternative is preferable.

This means that for some NFRs, they may actually be both requirements and preferences. There may be functional requirements that specify the threshold of acceptability and there may be preferences that indicate that solutions that "beat" the threshold should be preferred.

8 Summary

The definitions of function and functional requirement both refer back to the effect that a device has on its environment. The environment, however, need not only be the

operational environment in which the system is used. The environment varies depending on both the stage in the systems life-cycle and on who is interacting with the system.

NFRs are typically distinguished from FRs by either referring to something other than the function the device performs for the user, by cross-cutting such functions, or by requiring that the entire system be in place before they can be evaluated. These are interesting distinctions but not necessarily ones that mean the requirement is non-functional. In some cases, the NFR can be considered functional when viewed as operating in an environment other than the operational environment, such as deployment or maintenance. In others, the NFRs can be translated into FRs by specifying which pieces of the system must work together to meet the NFR and how. And finally, requiring that the entire system be in place should not make something non-functional – the requirement is still referring to an effect on the environment produced by the system.

In some cases, however, the NFRs are not specific enough to be translated into FRs. This is not because they are non-functional, instead this implies that these are actually preferences. If something is required then it must be possible to determine when the requirement is not met. Preferences are certainly important – they help determine the degree to which the customer will be satisfied by the final system – but unless they can be quantified they are not requirements. Instead, many of the NFRs can be split into FRs with specific thresholds of acceptability and preferences used to choose between alternatives.

9 Conclusions

Every requirement affects the system in some way, otherwise it would not be worth requiring. This applies to both requirements typically classified as functional and those typically classified as non-functional. If something is required, there should be some way to determine if that requirement has or has not been satisfied.

This does not imply that un-testable NF requirements such as “minimize cost” are not valuable – such factors are important to consider so that the system provides the maximum degree of user satisfaction. Such requirements, however, are not really requirements but are actually preferences.

Treating non-functional requirements as something separate and distinct from functional ones makes this determination more difficult. Instead of simply listing a set of general desirable properties as requirements, it would be far more useful if these properties could be elaborated into specific functional requirements that outline what the system must do in order to ensure that the requirement is met. This would allow the tools and techniques developed for tracing and testing functional requirements to be used to ensure that the so-called non-functional requirements are met as well.

10 References

Chandrasekaran, B. and Josephson, J.R.: 2000, Function in Device Representation, to appear in *Journal of Engineering with Computers, Special Issue on Computer Aided Engineering*.

Chandrasekaran, B. and Josephson J. R.: 1997, Representing Function as Effect. In: *Proceedings of Functional Modeling Workshop*, Mohammed Modarres (Editor), Electricite de France, Paris, France.

Chandrasekaran, B., and Kaindl, H.: 1996. Representing Functional Requirements and User-system Interactions. *AAAI Workshop on Modeling and Reasoning about Function*, Portland, Oregon, pp. 78-84

Chung, L., Nixon, B.A., and Yu, E.: 1995, Using Non-Functional Requirements to Systematically Select Among Alternatives in Architectural Design., in *Proc., ICSE-17 Workshop on Architectures for Software Systems*, Seattle, Washington, April 24-28.

Chung, L., Yu, E.: 1998, Achieving System-Wide Architectural Qualities, *OMG-DARPA MCC Workshop on Compositional Software Architectures*, January 6-8, Monterey, California, USA.

Dorfman, M.: 1990, System and Software Requirements Engineering, in *System and Software Requirements Engineering*, first edn., R. H. Thayer and M. Dorfman, Eds. Los Alamitos, CA: IEEE Computer Society Press, pp 4-16.

Filman, R. E.: 1998, Achieving Ilities, *Workshop on Compositional Software Architectures*, January 6-8, Monterey, California, USA, URL: http://www.objs.com/workshops/ws9801/papers/paper046.doc.

Larman, C.: 1997, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*, Prentice Hall, NJ, p. 42.

Manola, F.: 1999, Providing Systemic Properties (Ilities) and Quality of Service in Component-Based Systems. URL: http://www.objs.com/aits/9901-iquos.html.

Nuseibeh, B. and Easterbrook, S.: 2000, Requirements engineering: a roadmap. In A. Finkelstein, editor, *The Future of Software Engineering*, Special Volume published in conjunction with ICSE.

Roman, G.: 1985, A Taxonomy of Current Issues in Requirements Engineering, *Computer*, pp. 14-22.

Thayer, R.H., and Dorfman, M: 1990, Introduction, Issues, and Terminology, in *System and Software Requirements Engineering*, first edn., R. H. Thayer and M. Dorfman, Eds. Los Alamitos, CA: IEEE Computer Society Press, pp. 1-3.

Umeda, Y. and Tomiyama, T: 1997, Functional Reasoning in Design, *IEEE Expert*, Vol. 12, No.2, March-April 1997, pp. 42-48.

Yeh, R.T., and Ng, P.A.: 1990, Requirements Analysis – A Management Perspective, in *System and Software Requirements Engineering*, first edn., R. H. Thayer and M. Dorfman, Eds. Los Alamitos, CA: IEEE Computer Society Press, pp. 450-461.

Zave, P.:1995, Classification of research efforts in requirements engineering, *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, March 27-29, IEEE Computer Society Press, CA.