FURPS +

4.3.2 Nonfunctional Requirements

Nonfunctional requirements describe aspects of the system that are not directly related to the functional behavior of the system. Nonfunctional requirements include a broad variety of requirements that apply to many different aspects of the system, from usability to performance. The FURPS+[2] model used by the Unified Process [Jacobson et al., 1999] provides the following categories of nonfunctional requirements:

- **Usability** is the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component. Usability requirements include, for example, conventions adopted by the user interface, the scope of online help, and the level of user documentation. Often, clients address usability issues by requiring the developer to follow user interface guidelines on color schemes, logos, and fonts.
- **Reliability** is the ability of a system or component to perform its required functions under stated conditions for a specified period of time. Reliability requirements include, for example, an acceptable mean time to failure and the ability to detect specified faults or to withstand specified security attacks. More recently, this category is often replaced by **dependability**, which is the property of a computer system such that reliance can justifiably be placed on the service it delivers. Dependability includes reliability, **robustness** (the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environment conditions), and **safety** (a measure of the absence of catastrophic consequences to the environment).
- **Performance** requirements are concerned with quantifiable attributes of the system, such as **response time** (how quickly the system reacts to a user input), **throughput** (how much work the system can accomplish within a specified amount of time), **availability** (the degree to which a system or component is operational and accessible when required for use), and **accuracy**.
- **Supportability** requirements are concerned with the ease of changes to the system after deployment, including for example, **adaptability** (the ability to change the system to deal with additional application domain concepts), **maintainability** (the ability to change the system to deal with new technology or to fix defects), and internationalization (the ability to change the system to deal with additional international conventions, such as languages, units, and number formats). The ISO 9126 standard on software quality [ISO Std. 9126], similar to the FURPS+ model, replaces this category with two categories: **maintainability** and **portability** (the ease with which a system or component can be transferred from one hardware or software environment to another).

2. FURPS+ is an acronym using the first letter of the requirements categories: Functional, Usability, Reliability, Performance, and Supportability. The + indicates the additional subcategories. The FURPS model was originally proposed by [Grady, 1992]. The definitions in this section are quoted from [IEEE, 1990].

The FURPS+ model provides additional categories of requirements typically also included under the general label of nonfunctional requirements:

- **Implementation requirements** are constraints on the implementation of the system, including the use of specific tools, programming languages, or hardware platforms.
- **Interface requirements** are constraints imposed by external systems, including legacy systems and interchange formats.
- **Operations requirements** are constraints on the administration and management of the system in the operational setting.
- **Packaging requirements** are constraints on the actual delivery of the system (e.g., constraints on the installation media for setting up the software).
- **Legal requirements** are concerned with licensing, regulation, and certification issues. An example of a legal requirement is that software developed for the U.S. federal government must comply with Section 508 of the Rehabilitation Act of 1973, requiring that government information systems must be accessible to people with disabilities.

Nonfunctional requirements that fall into the URPS categories are called **quality requirements** of the system. Nonfunctional requirements that fall into the implementation, interface, operations, packaging, and legal categories are called **constraints** or **pseudo requirements**. Budget and schedule requirements are usually not treated as nonfunctional requirements, as they constrain attributes of the projects (see Chapter 14, *Project Management*). Figure 4-3 depicts the nonfunctional requirements for `SatWatch`.

Quality requirements for `SatWatch`

- Any user who knows how to read a digital watch and understands international time zone abbreviations should be able to use `SatWatch` without the user manual. [Usability requirement]
- As the `SatWatch` has no buttons, no software faults requiring the resetting of the watch should occur. [Reliability requirement]
- `SatWatch` should display the correct time zone within 5 minutes of the end of a GPS blackout period. [Performance requirement]
- `SatWatch` should measure time within 1/100th second over 5 years. [Performance requirement]
- `SatWatch` should display time correctly in all 24 time zones. [Performance requirement]
- `SatWatch` should accept upgrades to its onboard via the Webify Watch serial interface. [Supportability requirement]

Constraints for `SatWatch`

- All related software associated with `SatWatch`, including the onboard software, will be written using Java, to comply with current company policy. [Implementation requirement]
- `SatWatch` complies with the physical, electrical, and software interfaces defined by WebifyWatch API 2.0. [Interface requirement]

Figure 4-3 Nonfunctional requirements for `SatWatch`.

Table 4-3 Example questions for eliciting nonfunctional requirements.

Category	Example questions
Usability	• What is the level of expertise of the user? • What user interface standards are familiar to the user? • What documentation should be provided to the user?
Reliability *(including robustness, safety, and security)*	• How reliable, available, and robust should the system be? • Is restarting the system acceptable in the event of a failure? • How much data can the system loose? • How should the system handle exceptions? • Are there safety requirements of the system? • Are there security requirements of the system?
Performance	• How responsive should the system be? • Are any user tasks time critical? • How many concurrent users should it support? • How large is a typical data store for comparable systems? • What is the worse latency that is acceptable to users?
Supportability *(including maintainability and portability)*	• What are the foreseen extensions to the system? • Who maintains the system? • Are there plans to port the system to different software or hardware environments?
Implementation	• Are there constraints on the hardware platform? • Are constraints imposed by the maintenance team? • Are constraints imposed by the testing team?
Interface	• Should the system interact with any existing systems? • How are data exported/imported into the system? • What standards in use by the client should be supported by the system?
Operation	• Who manages the running system?
Packaging	• Who installs the system? • How many installations are foreseen? • Are there time constraints on the installation?
Legal	• How should the system be licensed? • Are any liability issues associated with system failures? • Are any royalties or licensing fees incurred by using specific algorithms or components?

cannot ensure that we identify all the essential nonfunctional requirements. To ensure completeness, we use the FURPS+ categories we described in Section 4.3.2 (or any other systematic taxonomy of nonfunctional requirements) as a checklist for asking questions of the client. Table 4-5 depicts the nonfunctional requirements we identified in ARENA after detailing the AnnounceTournament use case.

Table 4-5 Consolidated nonfunctional requirements for ARENA, after the first version of the detailed AnnounceTournament use case.

Category	Nonfunctional requirements
Usability	• Spectators must be able to access games in progress without prior registration and without prior knowledge of the Game.
Reliability	• Crashes due to software bugs in game components should interrupt at most one Tournament using the Game. The other Tournaments in progress should proceed normally. • When a Tournament is interrupted because of a crash, its LeagueOwner should be able to restart the Tournament. At most, only the last move of each interrupted Match can be lost.
Performance	• The system must support the kick-off of many parallel Tournaments (e.g., 10), each involving up to 64 Players and several hundreds of simultaneous Spectators. • Players should be able to play matches via an analog modem.
Supportability	• The Operator must be able to add new Games and new TournamentStyles. Such additions may require the system to be temporarily shut down and new modules (e.g., Java classes) to be added to the system. However, no modifications of the existing system should be required.
Implementation	• All users should be able to access an Arena with a web browser supporting cookies, Javascript, and Java applets. Administration functions used by the operator are not available through the web. • ARENA should run on any Unix operating system (e.g., MacOS X, Linux, Solaris).
Operation	• An Advertiser should not be able to spend more advertisement money than a fixed limit agreed beforehand with the Operator during the registration.
Legal	• Offers to and replies from Advertisers require secure authentication, so that agreements can be built solely on their replies. • Advertisers should be able to cancel sponsorship agreements within a fixed period, as required by local laws.