Defining Non-Functional Requirements

A system has properties that emerge from the combination of its parts. These emergent properties will surely be a matter of accident, not design, if the non-functional requirements, or system qualities, are not specified in advance. This white paper lays out important concepts and discusses capturing non-functional requirements in such a way that they can drive architectural decisions and be used to validate the architecture.

by Ruth Malan and Dana Bredemeyer
Bredemeyer Consulting
ruth_malan@bredemeyer.com
dana@bredemeyer.com

Introduction

One development team, being close to its function-complete checkpoint, was frantically scrambling to meet benchmark targets that the marketing team was just then putting together for system test. An architecture assessment revealed that some of these quality requirements could not be met by the current architecture without significant rework. This problem of attempting to work quality in at the end of the development phase has been around as long as we have been doing software development.

Another team started out with the goal of creating a system that would satisfy current user requirements and provide the basis for quickly developing other applications. After putting the engineers through object-oriented training and spending months on analysis and design, the project started to feel the pressure of the impending release date. As this pressure intensified, design and code reviews were scuttled and key architects and engineers left the team disgruntled by the long workdays and corruption of the vision of creating an extensible, evolvable system that would solve the development pressure problem in future releases. More and more engineers were added to the team to make up for this attrition. Under this pathological cycle, the design degenerated and was pretty much abandoned. Quality problems quickly emerged and escalated out of control. Somehow, through sheer heroics on the part of the engineers, the application was eventually released. It met the critical customer requirements, but came nowhere close to the organization's goal of reducing the time-to-market of follow-on releases. This is a project scenario that, unfortunately, will sound reminiscent to far too many development teams.

Simply put, either the non-functional requirements were not specified (in time), or compromised without explicit attention to the trade-offs involved. Not paying attention to eliciting, documenting and tracking non-functional requirements makes it harder to prioritize and make trade-offs between the quality[1] of the product, the cost to develop and enhance it, and the time-to-market of current and future releases. Without quality targets to guide the architects and engineers, design choices are arbitrary, and it is hard to assess the system during architecture and design reviews and system test.

In this paper we explore what non-functional requirements are and how to define them as part of the requirements process. We draw upon the lessons we have learned working with software architecture and product development teams, and have been influenced by the work of Kazman and others at the Software Engineering Institute at CMU (e.g., Kazman and Bass, 1994, Bass et al., 1998).

After laying the groundwork of concepts and distinctions, we discuss sources of non-functional requirements, and principles and templates for defining non-functional requirements.

Concepts and Distinctions

We begin by defining requirements concepts and making some distinctions among them. Foremost is the distinction between functional requirements and what, in practice, are generally called non-functional requirements. Functional requirements describe the behaviors (functions or services) of the system that support user goals, tasks or activities. Non-functional requirements include constraints and qualities. *Qualities* are properties or characteristics of the system that its stakeholders care about and hence will affect their degree of satisfaction with the system. *Constraints* are not subject to negotiation and, unlike qualities, are (theoretically at any rate) off-limits during design trade-offs. C*ontextual constraints* are characteristics of the "super-system" (i.e., the larger system into which the system under development will fit) or the development organization that constrain the development in some way. Examples include the target operating system or hardware platform in the case of the user environment, or the skill-set of available developers in the case of the development organization.

1. Where quality is the degree of match between the product requirements (stated or otherwise) and the actual product. It is defined from the point of view of the user's perception, expectation and goals or need.

Another useful distinction is between the executing system and the work products (e.g., architecture, design and code) that are generated in its creation. In the case of the executing system, the qualities of interest are relative to user goals. We will refer to these as *run-time qualities*. In the case of the work products, the qualities are driven by the development organization's goals. We will refer to these as *development-time qualities*.[1]

Run-time Qualities. Informally we can think of functional requirements capturing *what* the system must do, and the run-time qualities as describing *how well* these functional requirements are satisfied—where "how well" is judged by some externally observable/measurable property of the system behavior, not its internal implementation. In other words, "how well" may be judged by the user in terms of some characteristic that the user values or is concerned about. Run-time qualities include:

- usability (ease-of-use, learnability, memorability, efficiency, etc.)
- configurability and supportability
- correctness, reliability, availability
- quality of service requirements such as performance (throughput, response time, transit delay, latency, etc.)
- safety properties (so-called because they "prevent bad things from happening"), such as security and fault tolerance
- operational scalability including support for additional users or sites, or higher transaction volumes

The scope of these run-time requirements may be system-wide, or local to specific behavior.

Development-time Qualities. In addition to developing systems that satisfy their users, the development organization has a vested interest in the properties of the artifacts (architecture, design, code, etc.) of the development process. Qualities of these artifacts influence the effort and cost associated with current development as well as support for future changes or uses (maintenance, enhancement or reuse). Examples of development-time quality requirements are:

- localizability—ability to make adaptations due to regional differences
- modifiability or extensibility—ability to add (unspecified) future functionality
- evolvability—support for new capabilities or ability to exploit new technologies
- composability—ability to compose systems from plug-and-play components
- reusability—ability to (re)use in future systems

This distinction between run-time and development-time qualities has important implications for how the non-functional requirements are specified. Also, trade-offs may have to be made between run-time and development-time qualities. For example, performance and modifiability may be in tension in the system design, so that the users' desire (often influenced by competitive pressure) for performance may have to be traded off against the development organization's goal of having a more maintainable architecture.

In general, run-time qualities provide value to the user and have more to do with short-term competitive differentiation. Development-time qualities, for the most part, provide business value (as opposed to direct value to the end user) and have to do with the long-term competitiveness of the business. Everyone who is measured on short-term success will put short-term requirements ahead of those that contribute toward long-term effectiveness. In particular, developers and project managers are working against the release clock. This is why architects have to be vigilant about taking development-time qualities into account.

1. Bennet (1997) calls these "build-time" requirements.

Sources of Non-Functional Requirements

Run-time non-functional requirements arise from the operating environment, the user(s), and competitive products:

System Constraints. Here one is looking for elements of the environment into which the system must fit, that may serve as constraints on the system. This may be true of the installed infrastructure (e.g., hardware and OS platforms) or legacy applications, or may be in the form of organizational factors or the process that the system will support.

User Objectives, Values and Concerns. In establishing the run-time qualities for a system, it is important to identify all the categories of user (including other systems) that will interact with the system, and understand what quality attributes they care about. A quality attribute such as performance may surface for one user as a concern, and another as a value, so it is useful to direct elicitation of both values and concerns for any (group of) user(s). It is important to focus on creating *just* what users want, with the qualities they care about—low priority whiz-bang features or qualities only increase complexity for the development organization and/or for the user. The requirements team should nonetheless be alert to requirements that users take for granted or are not able to articulate directly. Understanding the users' objectives and forces that impact their success and sense of utility, will help surface and establish the priorities of system qualities—as well as functionality, of course.

In addition to discovering what qualities are important to users at the system level, qualities associated with particular functionality/user goals should be elicited. The qualities may need to be translated by developers from user-level objectives, values and concerns into specific technical quality requirements. For example, a user's requirement not to be impeded by slow system performance in conducting a task may be translated into requirements on transaction throughput and network latency.

Competitive Analysis of Features. Run-time qualities are often associated with product features. *Features* are generally thought of as the characteristics of the product that establish its competitiveness, frequently distinguishing the product functions (base-line and unique product differentiators) with at least one quality attribute. For example, many web-based catalog services have on-line payment options. To allay market concern, the electronic payment feature includes transaction security as an essential attribute.

Feature requirements may be carried over from past products, be driven reactively by competitors' products or proactively by the development team. Competitors' products, or the trade press's evaluation of them, may raise user expectations both in terms of functionality and in terms of system qualities. For this reason, marketing plays an important role in setting non-functional requirements by conducting competitive analyses to understand the competitive profile on qualities that customers value, or the trade press emphasizes (for product reviews influence the purchasing decision without necessarily reflecting the users' top priority objectives, values and concerns). The development team also plays a role in influencing expectations for these qualities, by understanding what new opportunities are afforded by technological advances.

This is not just true in product development, but in IT/IS development as well. In the latter case, you will typically be conducting the competitive analysis at the level of the services that your systems support. For example, are your competitors able to ship more quickly because their systems link more seamlessly to the distribution channel? Are they able to tailor services to a customer on-the-fly?

Development-time requirements typically are driven by the development organization (though in the case of outsourced development, they may arise from the customer).

Development Organization Constraints. In product development, constraints placed by upper levels of management typically take the form of required time-to-market of the application or release and/or fixed development resources. When both variables are fixed, the feature requirements have to be strictly scoped. This shows up in what functionality is scoped for the current release and what is deferred, and in driving trade-offs among the quality attributes of the system. Other factors of the development organization, such as the background and skill-set of the engineers, may also place constraints on what the development organization can accomplish especially given other constraints like time-to-market.

Development Organization Objectives, Values and Concerns. Stakeholders in the development organization include strategic management (e.g., general manager and R&D/IT manager), program and project managers, architects, developers, quality assurance (testers), marketing and manufacturing engineers, etc. Their objectives, values and concerns may relate to the business performance, schedules, productivity and effectiveness, work-life balance, etc. For example, strategic management establishes the product portfolio plan, including planned releases (which products in what timeframe). The architects and technical managers may translate those portfolio objectives into development-time quality requirements such as extensibility, evolvability and reuse, knowing that the portfolio cannot be accomplished without these characteristics. Developers may be concerned that reuse artifacts in fact deliver the qualities their particular product requires. And so forth.

Competitors and Industry Trends. Benchmarks of competitors' processes (e.g., how many products they release per year, with how many people) and industry trends, may drive the organization to adopt more aggressive productivity objectives which may in turn translate into development-time qualities such as evolvability and reuse.

Once you have worked with stakeholders to gather their requirements, these need to be documented in such a way that the architects, designers and implementers can all understand them and create a system that fulfills the requirements.[1]

SMART Requirements

In general, non-functional requirements have been (at best) specified in loose, fuzzy terms that are open to wide ranging and subjective interpretation. As such, they provide little guidance to architects and engineers as they make the already tough trade-offs necessary to meet schedule pressures and functionality goals. Instead, non-functional requirements need to be made precise and actionable. "SMART" requirements (Mannion and Keepence, 1995) have the following characteristics:

Specific: without ambiguity, using consistent terminology, simple and at the appropriate level of detail.

Measurable: it is possible to verify that this requirement has been met. What tests must be performed, or what criteria must be met to verify that the requirement is met?

Attainable: technically feasible. What is your professional judgement of the technical "do-ability" of the requirement?

Realizable: realistic, given the resources. Do you have the staffing? Do you have the skill? Do you have access to the development infrastructure needed? Do you have access to the run-time infrastructure needed? Do you have enough time?

Traceable: linked from its conception through its specification to its subsequent design, implementation and test.

The first of these (specific and measurable) provide criteria for each quality requirement—it is not well-specified if it is fuzzy or ambiguous or not measurable. The next (attainable and realizable) provide

1. At the end of each iteration the emerging system should be validated against the requirements.

checks you should perform with each requirement—do not include a requirement if it fails to meet either of these criteria. Note that when requirements are neither specific nor measurable, it is difficult to know if they are attainable and realizable. Gilb (1988) makes this point in the following principles:

"Projects without clear goals will not achieve their goals clearly."

"You can't hit a bullseye if you don't know where the target is."

Tools for Capturing and Documenting Non-Functional Requirements

Use cases have been widely used to specify functional requirements. By simply extending use cases with a field for all the non-functional requirements associated with the use case, run-time qualities associated with particular functionality can be captured conveniently (Table 1).

Use Case	*Use case identifier and reference number*
Description	*Goal to be achieved by use case and sources for requirement*
Actors	*List of actors involved in use case*
Assumptions	*Conditions that must be true for use case to terminate successfully*
Steps	*Interactions between actors and system that are necessary to achieve goal*
Variations *(optional)*	*Any variations in the steps of a use case*
Non-Functional	*List of non-functional requirements that the use case must meet* The nonfunctional requirements are listed in the form: <**keyword**> : < requirement> Non-functional keywords include, but are not limited to **Performance**, **Reliability, Fault Tolerance**, **Frequency,** and **Priority**. Each requirement is expressed in natural language or an appropriate formalism.
Issues	*List of issues that remain to be resolved*

Table 1: Use case template (from Coleman, 1998)[1]

Since development-time requirements such as extensibility or reuse generally relate to future run-time functionality, they could theoretically be captured using use cases too. However, exhaustively exploring requirements on future releases or product variants is out of the question (the first product would be delayed indefinitely), so development-time requirements need to be treated differently. Because so much is unknown and unpredictable about the future run-time environment and user needs, "what if" scenarios are used to explore the system's robustness to future requirements and the ease of adapting it to altering tech-

1. The reader is referred to Malan and Bredemeyer (1999b) and Coleman (1998) for a detailed treatment of the use case template, including definitions of the fields on the template as well as includes and extend variants of the use case.

nologies and user goals. These are described less formally than use cases, recognizing that they are only representative of categories of changes that may have to be made over the life of the product line or family.

Prioritization

Remember that you can do many things poorly or a few things well. Alternatively put, in complex systems development, you are not going to get everything right so you should make sure you get the most important things right! Therefore, be sure to prioritize your non-functional requirements and do not fall into the trap of expressing every possible quality as a requirement of your system. Collect user goals (we use *Stakeholder Profiles*, Malan and Bredemeyer, 2000), extract quality requirements, and prioritize them. It useful to at least categorize your requirements as "must", "high want", and "want."

Conclusion

A system has properties that emerge from the combination of its parts. These emergent properties will surely be a matter of accident, not design, if the non-functional requirements, or system qualities, are not specified in advance. Moreover, if architectures are intended to provide a source of competitive differentiation, architects must also take into account longer-term objectives of the business, and the non-functional requirements associated with the architecture and development artifacts.

Acknowledgments

This article has benefited from conversations with, and related work by, members of the HP Product Generation Solutions architecture consulting team, especially Derek Coleman, Ron Grace, Mark Interrante, Mike Ogush, and Steve Rhodes. We would also like to thank the many HP architects who have shared their approaches and insights with us. Publications by, and dialog with, Rick Kazman and co-authors, as well as Tom Gilb, have been highly influential in shaping our approach to non-functional requirements.

References

Bass, L., P. Clements, and R. Kazman, *Software Architecture in Practice*, Addison-Wesley, 1998.

Bennett, Douglas, *Designing Hard Software: The Essential Tasks*, Prentice-Hall, 1997.

Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, *Pattern-Oriented Architecture: A System of Patterns*. Addison-Wesley, 1996.

Coleman, Derek, "Use Case Template," 1998. Available on http://www.bredemeyer.com/papers.htm

Filman, Robert, "Achieving Ilities", On-line *Proceedings for the Workshop on Compositional Software Architecture*, January 1998. Available at http://www.objs.com/workshops/ws98/papers

Gilb, Tom, *Principles of Software Engineering Management*, Addison-Wesley, 1988.

Kazman, Rick and Len Bass, "Toward Deriving Software Architectures from Quality Attributes*", Software Engineering Institute Technical Report* CMU/SEI-94-TR-10.

Kazman, Rick and Len Bass and Gregory Abowd and Mike Webb, "SAAM: A Method for Analyzing the Properties Software Architectures", *Proceedings of the 16th International Conference on Software Engineering*, (Sorrento, Italy), May 1994, pp. 81-90.

Kazman, Rick and Paul Clements, and Len Bass and Gregory Abowd, "Predicting Software Quality by Architectural Evaluation", *Proceedings of the Fifth International Conference on Software Quality*, (Austin, TX), October 1995, pp. 485-497.

Kazman, Rick and G. Abowd, L. Bass, P. Clements, "Scenario-Based Analysis of Software Architecture*", IEEE Software*, November 1996, pp. 47-55.

Malan, R. and D. Bredemeyer, "Stakeholder Profile Action Guide", 2000. On http://www.bredemeyer.com/papers.htm

Malan, R. and D. Bredemeyer, "Functional Requirements and Use Cases", October 2001. On http://www.ewita.com

About Bredemeyer Consulting

Bredemeyer Consulting provides a range of consulting and training services focused on Enterprise, Systems and Software Architecture. We provide training and mentoring for architects, and typically work with architecture teams, helping to accelerate their creation or migration of an architecture. We also work with strategic management, providing consulting focused on developing architectural strategy and organizational competency in architecture.

BREDEMEYER CONSULTING
Bloomington, IN 47401
Tel: 1-812-335-1653
http://www.bredemeyer.com