CS445 / SE463 / ECE 451 / CS645

Software requirements specification & analysis

8. Non-Functional Requirements (NFRs)

Fall 2010 — Mike Godfrey and Dan Berry

Non-Functional Requirements

Non-Functional Requirements (NFRs) of a system are attributes and characteristics of the system.

Think of functional requirements as verbs, and NFRs or attributes as adjectives or adverbs.

Two products could have exactly the same functions, but their attributes can make them entirely different products. A Rolls Royce has more or less the same functions as a Yugo, but many, many different attributes.

Overview

- Non-functional requirements

  [aka NFRs aka "quality attributes" aka "the -ilities"]

  – Types of NFRs

  – Quantifying NFRs

  – Assessing compliance

NFRs: Quality requirements

- *Functional* requirements describe what the software is supposed to do
  - What services or tasks the software should provide
  - Black box input/output behaviour

- *Non-functional* requirements describe (extra) constraints on the way in which the SUD should satisfy the functional requirements

  e.g., How fast system should respond

  What deployment platforms should be supported

  What security / authentication goals should be met

NFRs: Quality requirements

- … are properties that the sw system must possess, but don't correspond to "buttons on a black box"
  e.g., performance, reliability, maintainability

- NFRs often measured in relative terms.
  - *"How much XXX?"*, not *"Add feature XXX".*

- NFRs strongly affect how a product is experienced by the user, esp. when base functionality is similar to other products
  - Rolls Royce, OS X, Firefox

Customers and requirements

- During elicitation interviews, customers naturally tend to focus on functional requirements
  - Expectations about NFRs may be implicit or just unknown as yet … but they're out there somewhere, eventually …
- Since NFRs strongly affect the "user experience", it makes sense to address them carefully
  - Get explicit models of acceptable and unacceptable quality values where possible
  - But often, this is hard or even impossible to do directly, so we must be creative.

NFRs: Quality requirements

- Performance
  - execution speed
  - response time
  - throughput
    e.g., “up to 30 simul. calls”
- Usability
  - how easy to learn / use
  - user productivity
- Reliability
  - fault-tolerant
  - mean-time to failure
  - data backups
- Security
  - controlled access to system or data
    e.g., Amazon browse vs. buy
  - isolation of data, programs
  - protect against theft, vandalism

NFRs: Quality requirements

- Robustness
  - tolerates invalid input
  - fault-tolerant
  - fail-safe / -secure
  - degrades gracefully under stress
- Adaptability
  - ease of adding new functionality (e.g., plugins)
  - reusable in other environments
  - self-optimizing
  - self-healing
- Scalability
  - workload
  - number of users
  - size of data sets
  - peak use
- Efficiency (capacity)
  - user productivity
  - utilization of resources
- Accuracy / precision
  - tolerance of computation errors
  - precision of computation results

Other types of NFRs

- *Process requirements* are restrictions on the techniques or resources that can be used to build the software
  e.g., development process, personnel

- *Design constraints* are design decisions that have already been made and that restrict the set of acceptable software solutions
  e.g., choice of platform, interface components

- *Product family requirements* concern how a particular product must integrate into a larger family of products
  e.g., Nokia phones, Sony electronics, iOS apps

Other types of NFRs

- Process Requirements
  - Resources
    - personnel development
    - costs
    - development schedule
  - Documentation
    - audience
    - conventions
    - readability
  - Complexity (of code)
    - comments / KLOC
    - max LOC / fcn or cyclomatic complexity
    - use of mult inh, overloading, templates
  - Standards compliance

    e.g., testing coverage

Other types of NFRs

- Design constraints
  - interfaces to other systems
  - COTS components
  - programming language
- Product-family requirements
  - modifiability
  - portability
  - reusability
  - UI
- Operating Constraints
  - location
  - size, power consumption
  - temperature, humidity
  - operating costs
  - accessibility (for maintenance)

Other NFRs

- What about fun? Is fun an attribute? It can be – especially if the product is a computer game. But you are not likely to find "fun" on any checklist of non-functional requirements to consider. So while lists of possible attributes are useful and may prompt a customer to reveal an important requirement, the analyst may be better off having the customer brainstorm his or her own non-functional requirements.
- See *ACM Interactions* Volume XI, Number 5, September + October 2004, for an issue on Funology!

“Motherhood” requirements

- Terms such as “reliable”, “user-friendly”, and “maintainable” are *motherhood requirements*.
  - No one would explicitly ask their opposite

  e.g., *slow, unreliable, user-hostile, unmaintainable*, …

- (Virtually) every software system must have attributes such as “reliable”, “user-friendly”, and “maintainable”; what differs from product to product is:
  - the *degree* to which each attribute is required, and
  - the *relative importance* of one attribute over another.

Fit for use

- "Quality" is not a measure of intrinsic value; rather, software quality is a measure of how *well* the software fits its intended purpose:
  - What is the system's purpose?
  - What environment will it run in?
  - What quality attributes will matter the most?

- It is not so much a question of the software being good, but of it being *good enough*.

Fitness criteria

- A *fitness criterion* quantifies the *extent* to which a quality requirement must be met.

  For example:
  - *75% of users shall judge the system to be as usable as the existing system*
  - *After training, 90% of users shall be able to process a new account within 4 minutes*
  - *A module will encapsulate the data representation of at most one data type*
  - *Computation errors shall be fixed within 3 weeks of being reported*

Cannot escape the need for a boundary, which is often arbitrary. What do you do if you are just past the boundary, e.g., by 0.1%?

Fitness criteria

- Fitness criteria express quality requirements in a way that makes it possible — objectively — to divide solutions into those that are acceptable and those that are not.

Quality Attribute	Metric	Test
Efficiency	maximum allowed load on resouces	run software in environment with limited resources
Reliability	mean-time to failure	run software and measure time between faiulres
Readability	Flesh Reading Ease Score	rate the documentation on the average number if syllable per word and words per sentence.

$$206.876 - 1.015\left(\frac{\text{total words}}{\text{total sentences}}\right) - 84.6\left(\frac{\text{total syllables}}{\text{total words}}\right)$$

Measurable metrics

Quality Attribute	Metric
Performance	response time throughput capacity
Efficiency	maximum allowed load on resouces
Reliability	mean-time to failure (MTTF)
Security	categories of users percentage of attacks that are successful
Robustness	percentage of failures on invalid input minimum performance under heavy loads
Scalability	size of input data sets
Cost	maximum costs to buy, install, or operate
Portability	collection of target platforms
Readability	Flesh Reading Ease Score
Maintainability	mean-time to fix bugs, add features
Usability	amount of training needed to perform tasks on own time to perform tasks at expected speed number of calls to help desk rate at which users adopt software approval rating user error rates

Example: Measuring reliability

- Reliability can be defined in terms of a percentage likelihood of success, downtime, absolute number of failures, …

- Reliability may have different meanings for different kinds of applications

  e.g., Telephone network:

  - *The entire network can fail no more than, on average, 1 hour per year, but failures of individual switches can occur much more frequently*

  e.g., Patient monitoring system:

  - *The system may fail for up to 1 hour per year, but in those cases doctors or nurses should be alerted of the failure. More frequent failure of individual components is unacceptable.*

Richer fitness criteria

Requirement	Outstanding	Target	Minimum
Response time	0.1s	0.5s	1s
CPU utilization	20%	25%	30%
Usability	20 tasks/hr	30 tasks/hr	40 tasks/hr

Measurements

- What gets measured gets done!
  - Therefore, unless a quantified requirement is unrealistic, it will probably be met.

- Its value will:
  - determine how hard the developer will have to work to achieve the requirement, and
  - may determine how many design alternatives from which the developer has to choose and still meet the requirements.

- There is a danger of focusing on what is measurable, and not on the true requirement

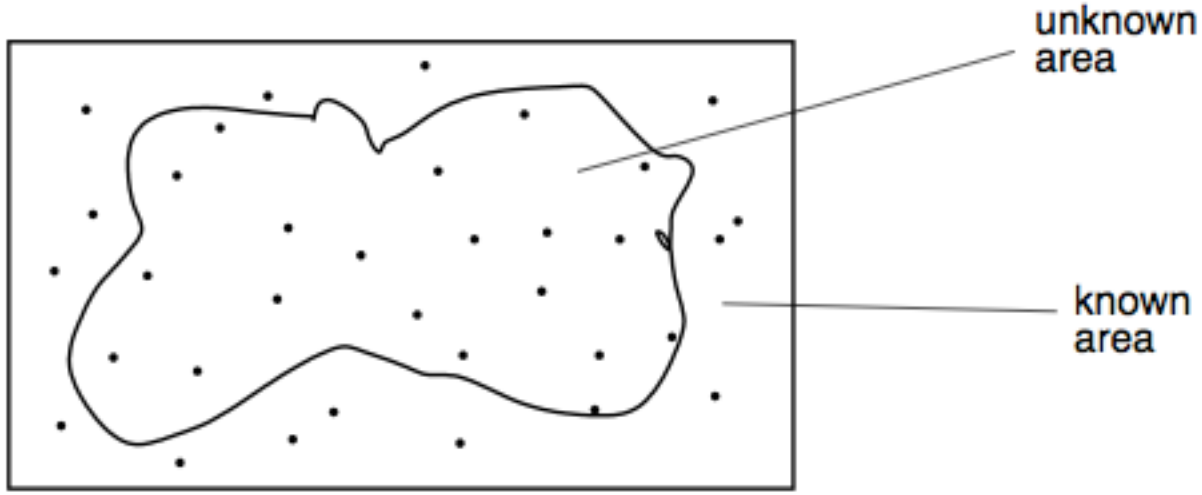
  e.g., industry benchmarks (sometimes)

When you can't test before delivery

- Fitness criteria that cannot be evaluated before the final product is delivered are harder to assess. For example:
  - *The system shall not be unavailable for more than a total of 3 minutes each year*
  - *The mean-time-to-failure shall be no less than 1 year*

- Possible approaches:
  - Measure the attributes of a prototype.
  - Measure secondary indicators
    e.g., number of user errors to assess usability
  - Estimate a system's quality attributes
  - Deliver system and pay penalty if requirements are not met

Monte Carlo techniques



- Monte Carlo techniques: estimate an unknown quantity using a known quantity.
  - For example, calculate the area of the above

Monte Carlo techniques

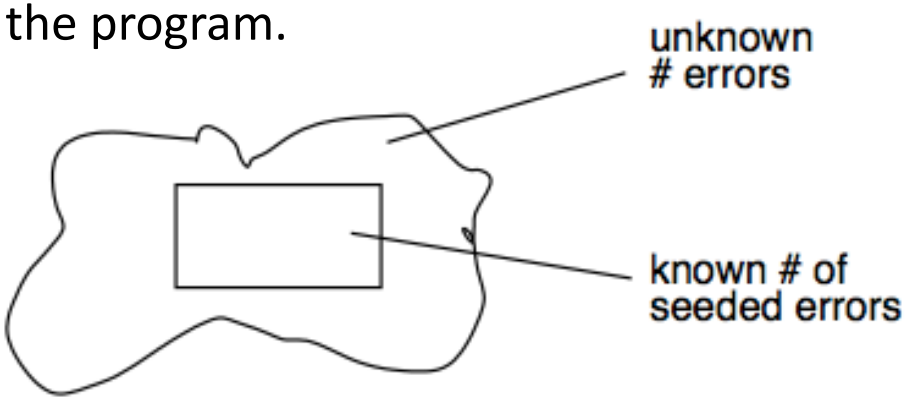
- We don't know the equation of the shape ☹

- However, if we surround this area with a shape whose area *is* known, and randomly drop points into the picture, counting the number that fall within the shape out of the number of total points.

$$\frac{\text{Number of points in shape}}{\text{Total number of points}} = \frac{\text{Area of Shape}}{\text{Known Area of Rectangle}}$$

Monte Carlo techniques

- We can use Monte Carlo techniques to estimate number of bugs remaining in a program (reliability).
  - Plant a known number of errors into the program, which the testing team does not know about.
  - Then compare the number of seeded errors the team detects with the number of total errors it detects, to arrive at an estimate of the total number of bugs in the program.



$$\frac{\text{\# detected seeded errors}}{\text{\# seeded errors}} = \frac{\text{\# detected errors}}{\text{\# errors in program}}$$

Quality–Function Deployment (QFD)

Quality–Function Deployment (QFD) is a way to relate an unmeasurable or hard-to-measure NFR to one or more functional requirements.

Quality–Function Deployment

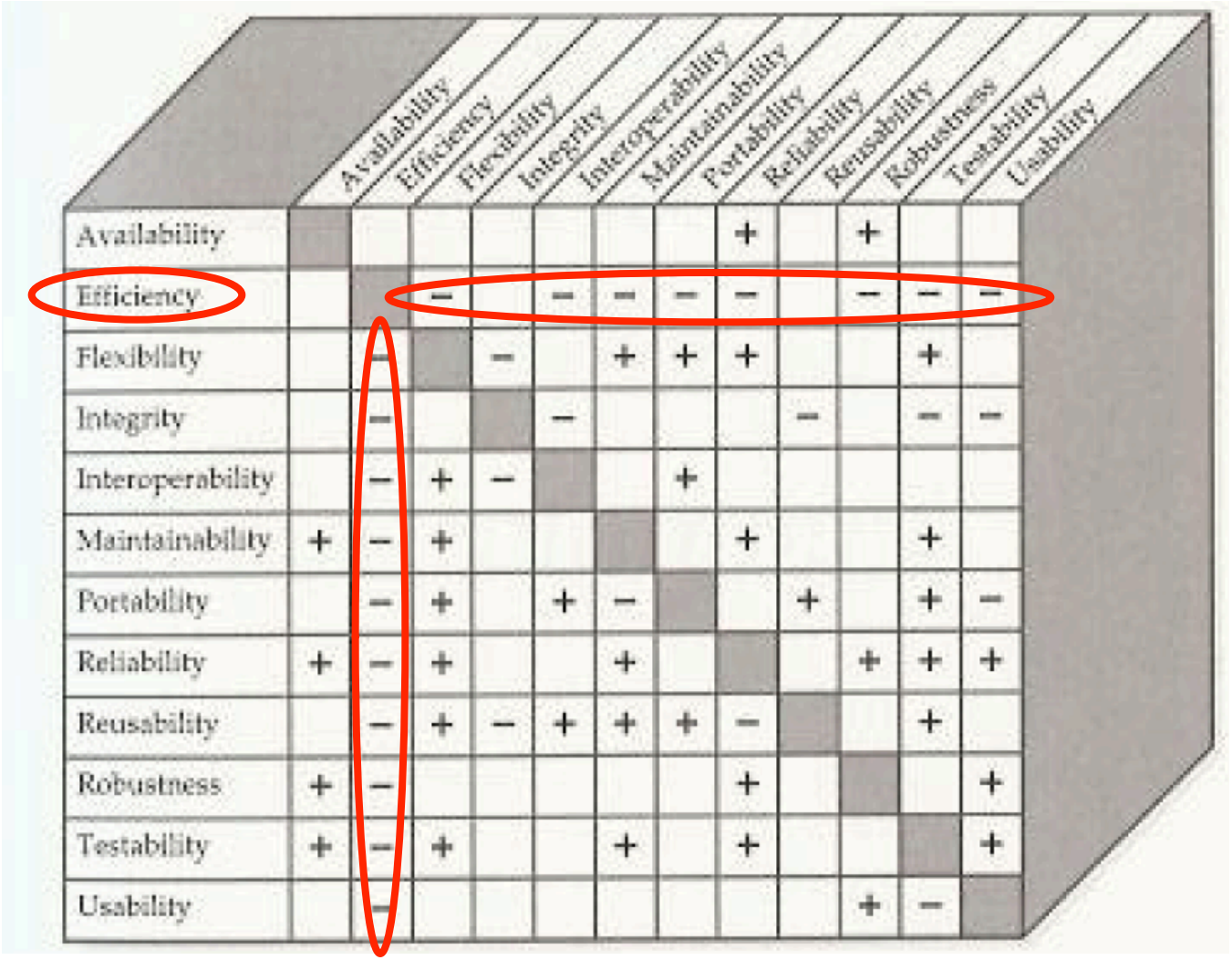
Q \ F	...	WIMP User Interface	...	On-Line Help System	...	Command-Line Interface	...
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
Easy To Use	...	+++	...	++	...	-	
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

Prioritizing NFRs

- Many NFRs conflict with one another:
  - maintainability vs. robustness
    - simple design vs. design that monitors run-time / error recovery
  - performance vs. security
  - performance vs. reuse
  - performance vs. portability
  - robustness vs. testability

- Also, some NFRs can conflict with functional requirements
  - performance vs. particular features (e.g., unlimited undo)

Typical NFR conflicts



	Availability	Efficiency	Flexibility	Integrity	Interoperability	Maintainability	Portability	Reliability	Reusability	Robustness	Testability	Usability
Availability								+		+		
Efficiency			−		−	−	−	−		−	−	−
Flexibility		−		−		+	+	+			+	
Integrity		−			−				−		−	−
Interoperability		−	+	−			+					
Maintainability	+	−	+					+			+	
Portability		−	+		+	−			+		+	−
Reliability	+	−	+			+				+	+	+
Reusability		−	+	−	+	+	+	−			+	
Robustness	+	−						+				+
Testability	+	−	+			+		+				+
Usability		−								+	−	

from *Software Requirements*, by Karl Wiegers (MS Press)

Prioritizing NFRs

- Most stakeholders can't easily questions such as:
  - *How important is inter-operability?*
  - *What mean-time-to-failure rate is acceptable?*

- So instead, we often rank requirements by priority helps make decisions when there are trade-offs.

- Also, different parts of the same system may have different priorities for NFRs, and different stakeholders may have differing answers depending on how they perceive the system

  e.g., aircraft avionics vs. entertainment system

Example quality grid

	Critical	Importance	As usual	Unimportant	Ignore
Product-oriented					
Performance					
Security					
Usability					
Family-oriented					
Portability					
Modifiability					
Reuse					
Process-oriented					
Maintainability					
Readability					
Testability					

NFRs: Summary

- NFRs affect *how* a system accomplishes its functional ("what") goals
- Often NFRs are highly important to user experience
- NFRs can be hard to measure
- NFRs often conflict with each other

CS445 / SE463 / ECE 451 / CS645

Software requirements specification & analysis

8. Non-Functional Requirements (NFRs)

Fall 2010 — Mike Godfrey and Dan Berry