Computer Science Program, The University of Texas, Dallas

Non-Functional Requirements

What are NFRs?

Types of NFRs

Product- vs. Process-Oriented Approaches

Process-Oriented Approach

Product-Oriented Approach

Portability, Reliability, Efficiency, Usability

Lawrence Chung

Non-Functional Requirements: What?

- "... Requirements definition is a careful assessment of the needs that a system needs to fulfill.
  It must say why a system is needed, based on current and foreseen conditions, which may be internal operations or an external market.
  It must say what system features will serve and satisfy this context.
  And it must say how the system is to be constructed ..." [Ross77]

- "non-functional requirement - in software system engineering, a software requirement that describes not what the software will do, but how the software will do it,
  for example, software performance requirements, software external interface requirements software design constraints, and software quality attributes.
  Nonfunctional requirements are difficult to test; therefore, they are usually evaluated subjectively." [IEEE]

- desirable system attributes, extra-functional requirements, quality factors, (not dysfunctional requirements)

  "-ilities": affordability, accessibility, adaptability, adjustability, availability, capability, compatibility, composability, comprehensibility, configurability, controllability, customizability, enhanceability, evolvability, expandability, extensibility, flexibility, inter-operability, learnability, maintanability, modifiability, portability, reconfigurability, reliability, repeatability, replaceability, reusability, scalability, standardizability, supportability, survivability, sustainability, testability, traceability trainability, transferability, usability, variability, versatility, ...

  "-ities": additivity, distributivity, diversity, modularity, plasticity, safety, security, similarity, simplicity, ...

  Other: accuracy, completeness, performance, responsiveness, user-friendliness, ...

Types of NFRs

generally informally stated, often contradictory,
difficult to enforce during development and evaluate
for the customer prior to delivery

- [Roman, IEEE Computer85]:

1 Interface requirements: describe how the system is to interface with its environment, users and other systems.
e.g., user interfaces and their qualities (e.g., user-friendliness)

1 Performance requirements: describe performance constraints involving

S time/space bounds, such as workloads, response time, throughput and available storage space, e.g., "system must handle 100 transactions/second"

S reliability involving the availability of components and integrity of information maintained and supplied to the system , e.g., "system must have less than 1hr downtime/3 months".

S security, such as permissible information flows

S survivability, such as system endurance under fire, natural catastrophes.

1 Operating requirements:
include physical constraints (size, weight), personnel availability, skill level considerations, system accessibility for maintenance, etc.

1 Lifecycle requirements: can be classified under two subcategories:

S Quality of the design: measured in terms such as maintainability, enhanceability, portability.

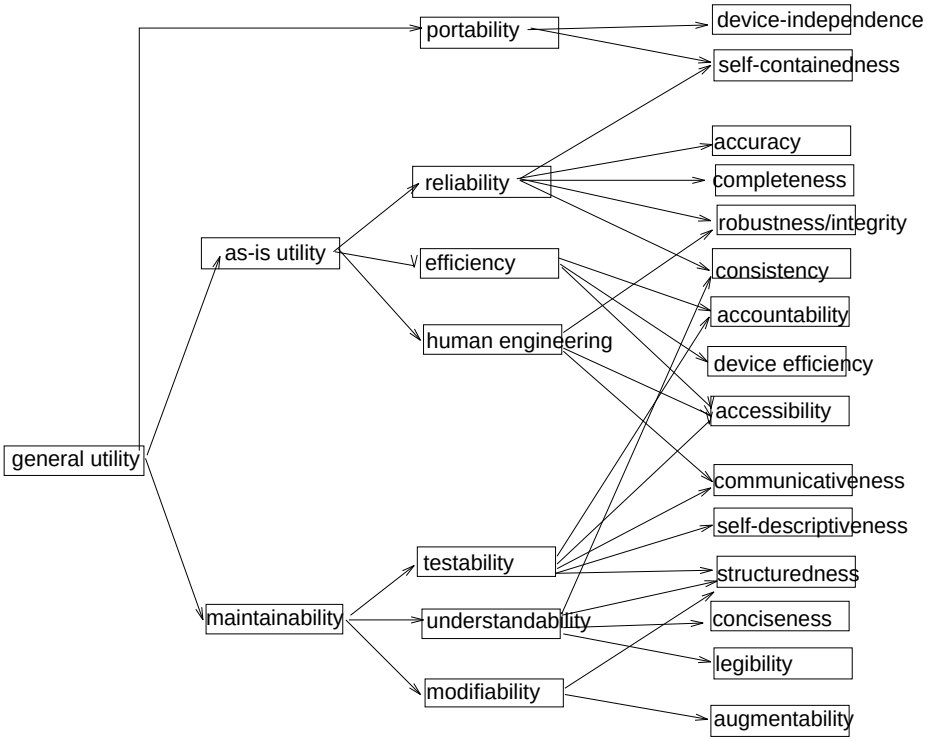
S Limits on development, such as development time limitations, resource availability, methodological standards, etc.

1 Economic requirements: immediate and/or long-term costs

1 Political requirements:

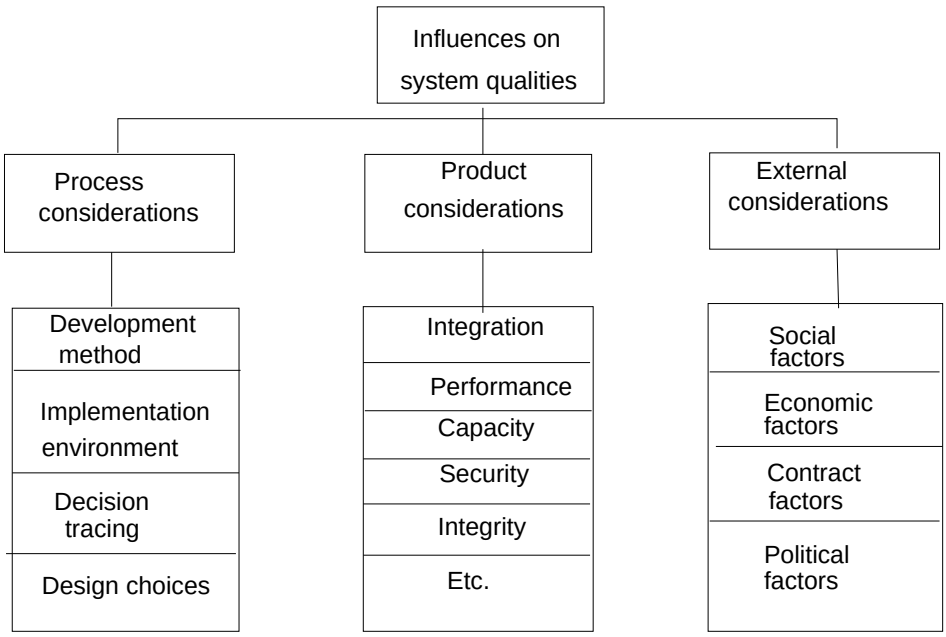
Software Quality Tree

[Boehm76]

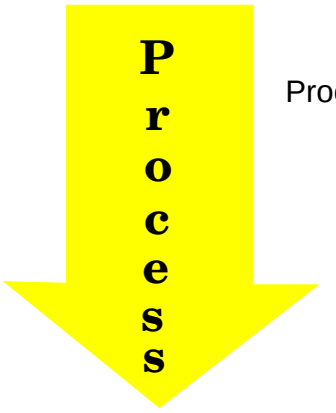


Process, Product & External Considerations

[Sommerville92]



Product- vs. Process-oriented Approaches

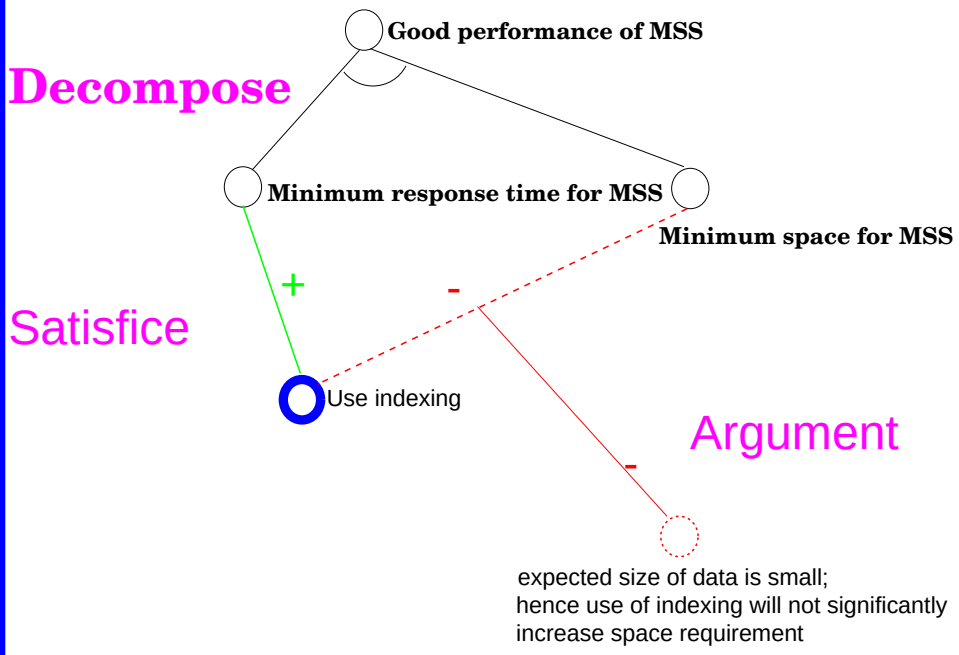


Process-oriented approaches:
integrate the enforcement of NFRs with the development process to build quality in

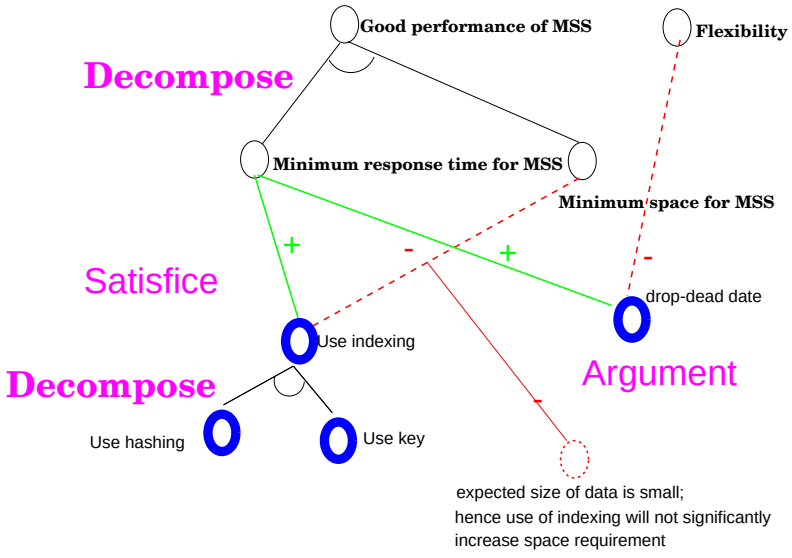
Product

Product-oriented approaches:
measure the degree to which a finished product meets a particular NFR.

Process-oriented approach



Process-oriented approach



- Disambiguate
- Design alternatives
- Tradeoff analysis
- Evaluate
- Prioritize
- Design rationale

Product-oriented approaches

^ Quality Metrics:

Property	Metric
Speed	transactions/sec, response time, screen refresh time
Size	KBytes, LOCs, Function Points, Complexity measures
Ease of use	transactions/sec, response time, screen refresh time

* usual metrification process:
  1. *determine a set of desirable attributes (i.e., ilities)*
  2. *determine relative importance/weight of such attributes*
  3. *evaluate the quality (rating) of each of the attributes*
  4. *compute weighted rating for each*
  5. *sum up all the weighted ratings*

Property	**relative weight**	**rating**	**weighted rating**
Speed	**.3**	**6**	**1.8**
Size	**.6**	**5**	**3.0**
Ease of use	**.1**	**7**	**0.7**
Overall Quality			**5.5/10**

* an inexact science at this point
* however, aids in understanding the factors that affect sw quality
  a first-cut approximation very poor quality factor

Portability

* the degree to which software running on one platform can easily be converted to run on another.
* E.g., number of target statements (e.g., from Unix to PC)
* hard to quantify, because it is hard to predict what a "next generation" platform might be like
* can be enhanced by using languages, OSs and tools that are universally available and standardized,
  e.g., Fortran, Cobol, C (for languages),
  Unix, Windows, OS/2 (OSs)

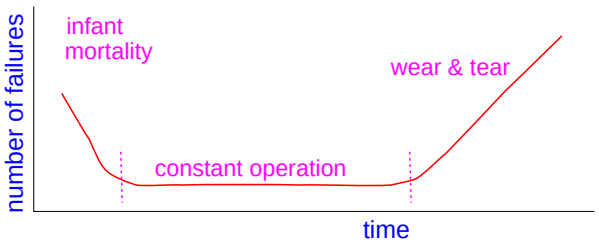
Reliability

* **the ability of the system to behave consistently in a user-acceptable manner when operating within the environment for which the system was intended.**

* **theory and practice of hardware reliability are** well established; some try to adopt them for sofwtare

* one popular metric for hardware reliability is mean-time-to-failure (MTTF)

  "Bathtub" curve characterizes MTTF:



* infant mortality: Given a large population of a particular component, many will fail soon after development due to inaccuracies in the manufacturing process;

* Do 2 different software copies have different characteristics?

  Does software wear & tear by decomposition?

  Does software obey physical laws?

Counting Bugs

- Sometimes reliability requirements take the form:

  "The software shall have no more than X bugs/1K LOC"

  But how do we measure bugs at delivery time?

- ***Begugging Process -*** ***based on a Monte Carlo technique for statistical analysis of random events.***

  1. before testing, a known number of bugs (seeded bugs) are secretly inserted.
  2. estimate the number of bugs in the system
  3. remove (both known and new) bugs.

$$\frac{\#\text{ of detected seeded bugs}}{\#\text{ of seeded bugs}} = \frac{\#\text{ of detected bugs}}{\#\text{ of bugs in the system}}$$

$$\#\text{ of bugs in the system} = \frac{\#\text{ of seeded bugs} \times \#\text{ of detected bugs}}{\#\text{ of detected seeded bugs}}$$

Example:

secretely seed 10 bugs

an independent test team detects 120 bugs (6 for the seeded)

$$\#\text{ of bugs in the system} = \frac{10 \times 120}{6} = 200$$

# of bugs in the system after removal = 200 - 120 - 4 = 76

- But, deadly bugs vs. insignifant ones; not all bugs are equally detectable;

(Suggestion [Musa87]:

"No more than X bugs/1K LOC may be detected during testing"

"No more than X bugs/1K LOC may be remain after delivery, as calculated by the Monte Carlo seeding technique"

Efficiency

- refers to the level at which a software system uses scarce computational resources, such as CPU cycles, memory, disk space, buffers and communication channels
- can be characterized along a number of dimensions:

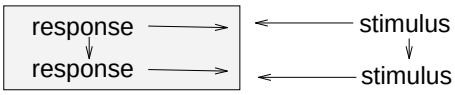
@Capacity: maximum number of users/terminals/transactions ...

@Degradation of service: what happens when a system with capacity X widgets per time unit receives X+1 widgets?

- Let the system crash
- Let the system handle the load, perhaps with degraded performance

@Timing constraints:

Let stimulus refer to an action performed by the user/environment, and response refer to an action generated by the system.



- stimulus-response: e.g., "the system will generate a dial tone within 10 secs from the time the phone is picked up"
- response-response: e.g., "the system will record that the phone is in use no later than 1 micro-second after it had generated a dial tone"
- stimulus-stimulus: e.g., "the user will type her password within 15 secs from typing her login name"
- response-stimulus: e.g., "the user will start dialing the phone number within 1 minute from getting the dial tone"

Usability

^ broadly - quality - fit to use
narrowly - good UI

^ Usability inspection:
finding usability problems in UI design, making recommendations for fixing them, and improving UI design.

Heuristics: a set of criteria against which usability of UI design is evaluated

"9 usability heuristics" [Nielsen90]

Promptness	no undue delay in accepting info items and responding to requests
Tolerance	no hang-ups against errors, delays, unexpected behavior, etc.
guidance	providing guidance for correcting errors, generating reminders, etc.
Coherence	...
...	

"10 usability heuristics" [Molich and Nielsen90]

Simple and natural dialogue; Speak the user's language
Minimize the user's memory; Consistency; Feedback
Clearly makred exits; Shortcuts
Precise and constructive error messages; Prevent errors
Help and documentation

Usability

^ All users will be satisfied with the usability of the product.

^ 95% of all users will be satisfied with the usability of the product.

^ 95% of the users will be able to complete representative tasks without requiring assistance (e.g., modifying exclusion date set)

^ 95% of the users will be able to complete representative tasks by the third attempt without requiring assistance

^ 95% of the users will be able to complete tasks X Y Z by the third attempt without requiring assistance

^ 95% of the users will be able to complete tasks X Y Z in less than 10 minutes without requiring assistance

^ 95% of the users will be able to complete task X in less than 10 minutes

80% of the users will be able to complete task Y in less than 10 minutes

77% of the users will be able to complete task Z in less than 5 minutes

Lawrence Chung

"... The fundamentality of satisficing - the fact that it is the *basic* structure of all real decision making, moral, prudential, economic or even evolutionary - gives birth to a familiar slipperiness of claim that bedevils theory in several quarters ..."

[Dennett, 1995]