XIII. Non-Functional Requirements (or, Quality Factors)

What are Non-Functional Requirements (NFRs)?

Classification of NFRs

Portability, Reliability, Performance

Example NFR for an Automated Money Machine

Non-Functional Requirements (NFRs)

- Define global constraints on a software system, such as development costs, operational costs, performance, reliability, maintainability, portability, robustness etc.
- Should not be confused with functional requirements, which impose requirements on the function of a system.
- Are generally stated informally, often contradictory, difficult to enforce during development and to evaluate for the customer prior to delivery.

How do we measure them?

How do we take them into account during development?

Types of NFRs

- Interface requirements -- describe how the information system is to interface with its environment, users and other systems; include user interfaces and their qualities (e.g., "user-friendliness")
- Performance requirements -- describe performance constraints involving
  - time/space bounds, such as workloads, response time, throughput and available storage space, e.g., "system must handle 1,000 transactions per second")
  - 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 per three months"
  - security, such as permissible information flows
  - survivability, such as system will survive fire, natural catastrophes
- Operating requirements -- include physical constraints (size, weight), personnel availability, skill level considerations, accessibility for maintenance, environmental conditions...

Classification of NFRs

<table>
<thead>
<tr>
<th>Acquisition Concern</th>
<th>User Concern</th>
<th>Quality Factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>Resource utilization security, confidence, performance under adversity, ease-of-use</td>
<td>efficiency, integrity, reliability, survivability, usability</td>
</tr>
<tr>
<td>Design</td>
<td>Conform to reqs? easy to repair?... verified performance?</td>
<td>maintainability, maintainability, maintainability, maintainability</td>
</tr>
<tr>
<td>Adaptation</td>
<td>Easy to expand? upgrade function or performance?... change?... interface with another system?... port?... use in another application?</td>
<td>expansibility, flexibility, interchangeability, portability, reusability</td>
</tr>
</tbody>
</table>

Factors and Criteria

- Factors are customer-related concerns, such as efficiency, integrity, reliability, correctness, survivability, usability...
- Criteria -- technical (development-oriented) concerns such as anomaly management, completeness, consistency, traceability, visibility...
- Each factor depends on a number of associated criteria, e.g., correctness depends on completeness, consistency, traceability...
- Verifiability depends on modularity, self-descriptiveness and simplicity
Quality Metrics

<table>
<thead>
<tr>
<th>Quality Metric</th>
<th>Quality Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>transactions/sec, response time, screen refresh time</td>
</tr>
<tr>
<td>Size</td>
<td>number of RAM chips</td>
</tr>
<tr>
<td>Ease of Use</td>
<td>training time, number of help frames</td>
</tr>
<tr>
<td>Reliability</td>
<td>mean-time-to-failure, probability of unavailability</td>
</tr>
<tr>
<td>Robustness</td>
<td>time to restart after failure, percentage of events causing failure</td>
</tr>
<tr>
<td>Portability</td>
<td>percentage of target-dependent statements, number of target systems</td>
</tr>
</tbody>
</table>

Factors vs Criteria

<table>
<thead>
<tr>
<th>Factors</th>
<th>Design</th>
<th>Performance</th>
<th>Robustness</th>
<th>Reliability</th>
<th>Ease of Use</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portability</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Robustness</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ease of Use</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>System accessibility</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Design vs Criteria</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Completeness</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Availability</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Traceability</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Visibility</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Portability

Portability is the degree to which software running on one platform can easily be converted to run on another.

- Portability is hard to quantify, because it is hard to predict on what other platforms the software will be required to run.
- Portability for a given software system can be enhanced by using languages, operating systems, and tools that are universally available and standardized, such as FORTRAN, COBOL, or C (for languages), or Unix, Windows or OS/2 (operating systems).
- Portability requirements should be given priority for systems that may have to run on different platforms in the near future.

Reliability

Reliability of a software system is defined as the ability of the system to be expected consistently in a user-acceptable manner when operating within the environment for which it was intended.

Reliability can be defined in terms of a percentage (say, 99.999%) may have different meaning for different system:

- for a telephone, it might mean that the telephone should break down, on average, 1hr per year
- for a patient monitoring system, it may mean that the system may fail for up to 1hr/year, but in those cases doctors/nurses should be alerted of the failure.

Reliability: Adopting Techniques from Hardware

Theory and practice of hardware reliability are well established, and some try to adopt them for software.

Most popular metric for hardware reliability is mean-time-to-failure (MTTF). "Bathtub" curve characterizes the failure rate of an artifact during its lifetime:
Reliability: Counting Bugs

- Sometimes reliability requirements take the form: "The software shall have no more than X bugs per thousand lines of code".

**But how do we measure bugs at delivery time?**

- Use debugging: just before testing, a number of seeded bugs are introduced to the software system, then testing is done and bugs are uncovered (seeded or otherwise).
- **Number of bugs = # of seeded bugs - # of detected bugs in system**
- **The theoretical underpinnings of the approach are founded in Monte Carlo statistical analysis techniques for random events.**

**...BUT, not all bugs are equally important!**

Reliability Metrics

Reliability requirements have to be tied to the loss incurred by software system failure, e.g., destruction of mankind, destruction of a city, destruction of some people, injury to some people, major financial loss, major embarrassment, minor financial loss. Different metrics are more appropriate in different situations:

- **Probability of failure on demand.** This measures the likelihood that the system will behave in an unexpected way when some demand is made of it. This is most relevant to safety-critical systems.
- **Rate of Failure Occurrence (ROCOF).** This measures the frequency of unexpected behaviour. For example, ROCOF=2/100 means that 2 failures are likely to occur within every 100 time units.
- **Mean Time to Failure (MTTF).** Discussed earlier.
- **Availability.** Measures the likelihood that the system will be available for use. This is a good measure for applications such as telecommunications, where the repair/restart time is significant and the loss of service is important.

Efficiency

- **Software efficiency** refers to the level at which a software system uses scarce computational resources, such as CPU cycles, memory, disk space, buffers and communications channels.

- **Efficiency** 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? We don't want the system to simply crash! Rather, we may want to stipulate that the system should handle the load, perhaps with degraded performance.

Reliability in Perspective

- **The best that can be done in terms of reliability requirements is something like:**
  - "...No more than X bugs per 10KLOC may be detected during integration and testing; no more than Y bugs per 10KLOC may remain in the system after delivery, as calculated by the Monte Carlo seeding technique of appendix Z; the system must be 100% operational 99.9% of the calendar year during its first year of operation..."

- **Reliability requirements have to be tied to the loss incurred by software system failure, e.g., destruction of mankind, destruction of a city, destruction of some people, injury to some people, major financial loss, major embarrassment, minor financial loss.

Efficiency: Timing Requirements

- Let **stimulus** refer to an action performed by the user/environment, **response** is a system-generated action.

- **Four types of timing requirements** [Dasarathy85]:
  - **Stimulus-response** - e.g., "...the system will generate a dial tone within 2secs from the time the phone is picked up...", or "...the system will arm the alarm no sooner than 1min after it is completed..."
  - **Response-stimulus** - e.g., "...the system will generate a dial tone within 2secs from the time the phone is picked up...", or "...the user must dial her password within 15secs from typing her login name...
  - **Degradation of service** - what happens when a system with capacity X widgets per time-unit receives X+1 widgets? We don't want the system to simply crash! Rather, we may want to stipulate that the system should handle the load, perhaps with degraded performance.
Safety

- Safety is a critical requirement for certain types of software systems, e.g., nuclear plants, airplanes, X-ray machines, ... where failure may result in loss of human life.
- Analysis of safety requirements often entails hazard analysis and fault trees; these are techniques adopted from engineering disciplines.
- A hazard is a condition which may cause human death or injury (a "mishap").
- Severity of a hazard measures the worst possible damage caused by a hazard. Risk measures the probability of damage to humans.

Safety Hazards

Here are some hazards for an insulin delivery machine which is attached to a patient and automatically delivers prescribed insulin doses to a patient:

<table>
<thead>
<tr>
<th>Hazard</th>
<th>Probability</th>
<th>Severity</th>
<th>Estimated risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insulin overdose</td>
<td>medium</td>
<td>high</td>
<td>high</td>
</tr>
<tr>
<td>Insulin underdose</td>
<td>high</td>
<td>low</td>
<td>low</td>
</tr>
<tr>
<td>Power failure</td>
<td>high</td>
<td>low</td>
<td>low</td>
</tr>
<tr>
<td>Machine breaks off</td>
<td>low</td>
<td>high</td>
<td>medium</td>
</tr>
<tr>
<td>in patient</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Infection</td>
<td>medium</td>
<td>medium</td>
<td>medium</td>
</tr>
<tr>
<td>Allergic reaction</td>
<td>low</td>
<td>low</td>
<td>low</td>
</tr>
</tbody>
</table>

Types of Hazard Analysis

- Forward -- takes an initial event and traces it forward e.g., pipe breaks, pressure drops, pump breaks...
- Backward -- starts with a final outcome and determines the events that lead to it; e.g., insulin overdose <-> bad calculation <-> defective sugar-level sensor
- Problems with hazard analysis:
  - Unrealistic assumptions, such as system built according to specs, operators are experienced and trained, testing is perfect, maintenance is perfect, key events are random and independent;
  - Accident model doesn’t match reality;
  - Model oversimplifies reality.

Fault Tree Example

![Fault Tree Diagram]

Fault Tree Evaluation

Pros

- Forces system-level analysis
- Offers an intuitive way of displaying relations between events
- Facilitates the detection of omissions

Cons

- Requires detailed understanding of the system that is being analyzed
- Some automation of the analysis is possible, but only for hardware
- Does not work for large complex systems because the fanout is huge e.g., can’t do a fault tree with root node “airplane crashed”...

Event Tree Analysis

This is a forward search hazard analysis method It starts with a description of event chains

![Event Tree Diagram]
Non-Functional Requirements

Human factors -- building a user-friendly system requires expertise that most of us do not have; [Mantei88] describes cost-benefit tradeoffs of human factors.

Testability and Modifiability -- Usually defined in terms of cohesion and coupling.

Reliability Requirements
The AMM System shall exhibit a system Mean Time Between Failure (MTBF) of not less than 96 hours. MTBF is defined as the quotient of the total number of operating hours divided by the total number of failures.

Availability Requirements
The AMM System shall exhibit an availability of not less than 95 percent.

Maintainability Requirements
The AMM System shall maintain a Mean Time To Repair (MTTR) of not more than 2 hours. The MTTR is defined as the sum of the time required for fault isolation, correction, and restoration to service for each failure divided by the number of failures.

Expandability Requirements
The AMM System shall be designed in such a manner as to allow for future addition of 4 user buttons and 4 additional banking services.

Security Requirements
Access to account transactions shall be restricted to holders of valid banking cards and personal identification numbers.

Platform Requirements
The AMM System shall operate on a 80386 processor or better.

Example NFRs for the AMM

Maintainability Requirements
The AMM System shall exhibit a Mean Time To Repair (MTTR) of not more than 2 hours. The MTTR is defined as the sum of the time required for fault isolation, correction, and restoration to service for each failure divided by the number of failures.

Availability Requirements
The AMM System shall exhibit an availability of not less than 95 percent.

Reliability Requirements
The AMM System shall exhibit a system Mean Time Between Failure (MTBF) of not less than 96 hours. MTBF is defined as the quotient of the total number of operating hours divided by the total number of failures.

Expandability Requirements
The AMM System shall be designed in such a manner as to allow for future addition of 4 user buttons and 4 additional banking services.

More Examples

Restart Requirements
The AMM System shall perform an automatic restart in the event of a fatal software error, to be completed within 5 minutes.

Backup Requirements
The AMM System has no backup requirements as the banker account information is stored on the People’s Bank Computer System.

Fallback Requirements
The AMM System shall terminate the current transaction and shutdown in the event of a fatal device error, repeatable fatal software error, or network failure. The AMM System will not be operational again until the maintenance crew has investigated the failure.

Security Requirements
Access to account transactions shall be restricted to holders of valid banking cards and personal identification numbers.

Platform Requirements
The AMM System shall operate with not more than 4 MB RAM. 1 MB RAM shall be reserved for local data structures. 3 MB RAM shall be reserved for the operating system.

The AMM System shall operate with not more than 80 MB hard disk space. 3MB hard disk space is reserved for banking service files and configuration files.

The AMM System will operate on a 80386 processor or better.
Performance Requirements

The AMM System will be allocated 1.0 MB main memory to accommodate local data structures.
The AMM System will be allocated 3 MB hard disk space to accommodate any AMM banking files or configuration files.
The AMM System will respond to all banker requests in less than 10 seconds. This time shall be allocated as follows:

Banking Applications Subsystem: 0.5 seconds
Network Manager Subsystem: 0.5 seconds
People’s Bank Computer System / Network: 9 seconds

Timing analysis will be performed throughout the design and implementation of the subsystem to ensure that timing allocations are not being exceeded.

Additional Readings