Exploring Problems and Solutions in estimating Testing Effort for Non Functional Requirement.

Pratima Singh
Deptt. Of Computer Engineering,
I.I.T.(BHU).
Varanasi, U.P.

Anil Kumar Tripathi.
Deptt. Of Computer Engineering,
I.I.T.(BHU).
Varanasi, U.P.

ABSTRACT

Importance of testing has been realized in literature now, although late, but has been realized. Testing of Non Functional Requirement still remains unattended by the Software engineering Community. It is still being given second hand treatment from its specification, design to Testing. We try to analyze the contributory factors of testability of NFR, so that some metrification for the purpose of Effort Estimation due to NFR can be estimated. For the purpose of same we have tried to identify certain, difficulty related to NFR and indicators of its testability.

Keywords

Non Functional Requirement, Testing, Metrics, Testability,

1. INTRODUCTION

The increasing visibility of software as a system element and the attendant "costs" associated with a software failure are motivating forces for well-planned, thorough testing. It is not unusual for a software development organization to expend between 30 and 40 percent of total project effort on testing. In the extreme, testing of human-rated software (e.g., flight control, nuclear reactor monitoring) can cost three to five times as much as all other software engineering steps combined![18] Due to the enormous pressure towards deploying software as fast as possible, functional requirements have been the main focus of software development process at the expense of implementing non-functional requirements (NFRs) such as performance and security. Thus, in practice, NFRs have been observed to be frequently neglected or forgotten in the software development process. However, NFRs is an important concept in requirements engineering which plays an essential role in the success or the failure of systems. NFRs introduce quality characteristics, but they also represent constraints under which the system must operate. Due to enormous [1]The importance of NFR becomes more crucial for mission critical software. NFR are handles informally, confusingly, intermingled with Other Functional requirement in SRS. Identification & Isolation of NFRs are a problem & it becomes multifold important for a Business or mission critical system.[2]

There are several Non Formal Methods using Natural language & a variety of graphical notations. Although careful application of analysis and design methods, coupled with thorough review can and does lead to high-quality software, sloppiness in the application of these methods can create a variety of problems. A system specification can contain contradictions, ambiguities, vagueness, incomplete statements, and mixed levels of abstraction. [18]

Testability can be broadly defined as: Some define testability even very broadly: as anything that makes software easier to test improves its testability, whether by making it easier to design and test more efficiently. According to[3] testability is composed of the following.

• Control. The better we can control it, the more the testing can be automated and optimized.

• Visibility. What we see is what we test.

• Operability. The better it works, the more efficiently it can be tested.

• Simplicity. The less there is to test, the more quickly we can test it.

• Understandability. The more information we have, the smarter we test.

• Suitability. The more we know about the intended use of the software, the better we

can organize our testing to find important bugs.

• Stability. The fewer the changes, the fewer the disruptions to testing.

This broader perspective is useful when you need to estimate the effort required for testing or justify your estimates to others.

According to ISO 9126 S/W quality attributes comprises of six main attributes(called characteristics)[19]are 1)Functionality: The capability to provide functions which meets stated & implied needs when the s/w is used.

2) Reliability: The capability to maintain a specified level of performance.

3) Usability: The capability to be understood, learned & used

4) Efficiency: The capability to provide appropriate performance relative to the amount of resources used.

5) Maintainability: The capability to be modified for the purpose of making corrections, improvements or adaptations.

6) Portability: The capability to be adapted for different specified environments without applying actions or means other than those provided for this purpose in the product.

Usability: has the characteristic of understandability, learn ability, operability.

Maintainability: implies changeability, testability, stability

Portability: adaptability, instability

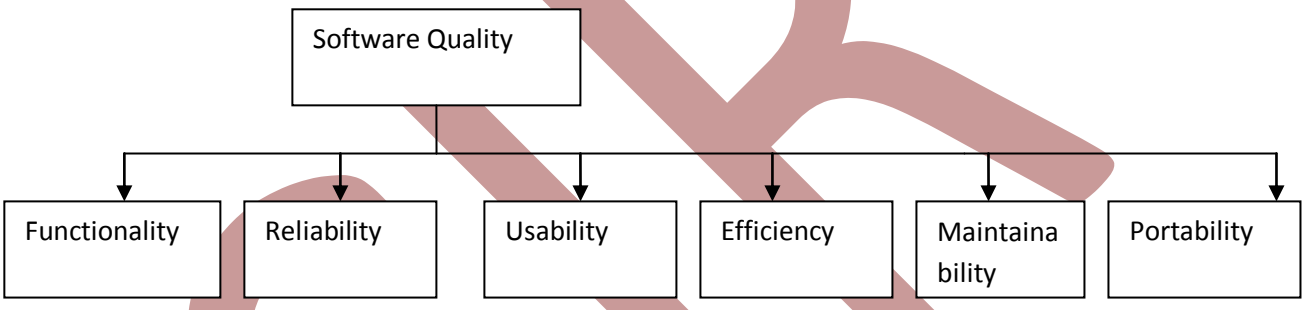
There are two important consequence of having multiple dimensions to quality

First software quality cannot be reduced to a single no(single parameter)

Second the concept of Quality is project specific [19]

Essentially a software system's utility is determined by both its functionality and its non-functional characteristics, such as usability, flexibility, performance, interoperability and security. Nonetheless, there has been a lop-sided emphasis in the functionality of the software, even though the functionality is not useful or usable without the necessary non-functional characteristics[4]

Non-Functional requirement - in software system engineering, is a software requirement that describes not what the software will do ,but how the software will do it, for example, performance requirements, software design constraints, software external interface requirements and software quality attributes. Nonfunctional requirements are difficult to test; therefore, they are usually evaluated subjectively. [4]Just with almost everything else the concept of quality is also fundamental to software engineering . Both functional & non functional characteristics may be taken into consideration in the development of quality software. NFR can be explained as several "abilities" as extra-functional requirements, quality factors,(not dysfunctional requirements)"-ilities": 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



The following Tree explains the various Quality Factors for Product , Process & External considerations.[19]

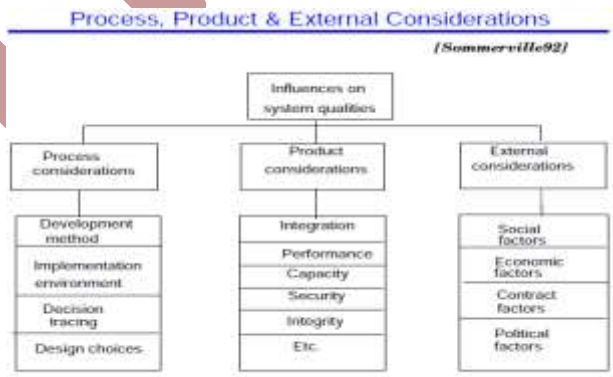
trainability, transferability, usability, variability, versatility, ...-ities": additivity, distributivity, diversity, modularity, plasticity, safety, security, similarity, simplicity, ...

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

[4]NFRs introduce quality characteristics, but they also represent constraints under which the system must operate. So, the chances of success for the software system are maximized when NFRs are modeled since the initial phases of the development process.

Several definitions of NFR exist in the literature.

IEEE defines Non-Functional Requirements as "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, design constraints, and software quality attributes".[1]



[20]

2. Major challenges in handling NFR:

A survey from a small sample of organizations, of the state of the practice in terms of nonfunctional requirements has shown that[ref:paper8] nonfunctional are often overlooked,. Questioning users is insufficient .Methods do not help the elicitation of nonfunctional requirements, and there is a lack of consensus about the meaning and utility of nonfunctional requirements

1)Identification /Isolation & Incorporation of NFR in Architect.

2)Conflicting Identification & resolution among NFR is major concern. Conflict Resolution of several intermingled requirements is difficult. optimizing one at the cost of other is multi optimization problem

3)Great Diversity of NFR.(Types of NFR & various attributes of various NFR

Great diversity of NFR makes it difficult to be identified and dealt with.[1].All NFR can't be treated a likely, for example security & usability requirements can not be treated in the same way. It is difficult to follow a single way or method of dealing with different NFR at the same time.[6]

\4)Cross Cutting concerns penetrating across several modules. Different viewpoints of the same software

5).Informal treatment of NFR [1]. NFR has been treated as second class requirement after FR.Explictty dealing with NFR and specifying NFR in concert to FR is still a future research area[ref50].Explicit dealing of NFR is missing[6].

6) Incorporating NFR's into different phase of software life cycle is difficult.

7) NFR are not mapped directly and explicitly from requirements engineering to implementation . This Non mapping of NFR from Req to implementation phase results in NFR Omission there by Overrun in Cost & schedule.[1]

8)NFR are Subjective in Nature[6]

9) Conventional Testing Methodology do not handle NFR Properly[7]

10) There is shortage of mature design methodology for any form of NFR .[1,7]

11)First software quality cannot be reduced to a single no(single parameter)[19] So NFR are difficult to identify ,handle and be paramatized. [1]

12) most of the work on NFR uses Product oriented approach only. which is concerned with measuring how often a software system is in harmony with set of NFR that it should satisfy.[5,14] . Very few , process oriented approach for NR is there in literature. POMSA(Process Oriented Metrics for software Architecture Adaptability) achieves the needed tracing by adopting the NFR Framework.[5]

13) Existing requirement specification are difficult to extend. Solution are being searched in formal methods which are all the more uncommon because of its dificcult understanding & implementation. A survey of the literature found that most people use informal or semi-formal approaches to specify NFRs because the formal ones are still perceived as more difficult and expensive. [7]

14) There are no metrics for ranking nonfunctional requirements . For Example Describing software reliability via hardware reliability metrics such as "mean time between failure" is nonsensical. Ways of expressing the "importance" or "criticality" of components are also lacking.[7]

15)Programming language constructs for directly expressing non-functional requirements are weak (even when the new Ada proposals are considered)[7]

16) A survey of the literature found that current programming languages were not designed with NFRs.[1]

3.The principal challenges for testing and debugging non-functional requirements are as follows.[7]

1)It is difficult to test embedded systems under realistic conditions. Simulation environments, particularly for hardware components, are themselves error-prone. A related problem

is the difficulty of constructing "harnesses" for testing software components in isolation.

2) Debugging distributed systems is an unsolved problem in itself, even before non-functional requirements are considered

3) Introducing debugging code into a program may alter its non-functional characteristics, especially timing

4)The huge amount of data generated by fast embedded systems is difficult to capture and present to the programmer in a comprehensible format.

4.Approaches to Handle NFR related problems are:

) Two extremes of Requirement specification are Natural Language & Formal Methods. Extending and relaxing formal methods in order to support the majority of NFRs is one of the solution.[9]

2)Modeling and analyzing functional requirements and NFRs that should be considered separately. Providing methodologies guidance through the whole development process.[1]

3) The most mature theoretical work on NFR is on "timed" Petri nets. Abstract specification language (Formal Methods)such as Z & VDM have been used for timing specifications .[1]

4) Researches are proposing several models & approaches to tightly integrate FR ,NFR to Architectural Decisions.[10,11]

5)NFR refers to orthogonal properties, conditions and restrictions that are spread out over the entire system. pure OO Approach do not handle these cross cutting concerns successfully so new approach like Aspect Oriented Programming is applied to fill this gap[12]

6)The **NFR Framework** is the most popular work in this topic.It promotes goal orientation with major emphasis on NFR.[1] It treats NFRs as soft goals (goals that are hard to express) to be addressed during the development process. NFRs, design decisions and their relations are captured in a goal graph where the nodes are either NFRs or design decisions. Goals in NFR Framework can be refined into detailed concrete goals. NFR Framework makes the relationships between NFRs and intended decisions explicit. This helps better understand the impact of every design decision; i.e. typically one design decision may impact multiple NFRs positively or negatively. The main interest of this framework is that it can be reused by other models to handle NFRs. NFR .

The i* family,Tropos and GRL(Goal Requirement Language) inherited the concept of softgoal from the NFR Framework aiming at dealing with softgoals, or non-functionality related attributes as a first class modeling concepts.[4] Goal oriented methods such as the NFR,KAOS and i* family are the few process that consider Non Functionality as a first class concepts.

7) There has been an effort to link UML to NFR Framework by extending UML for NFR .

Attempts have been made to integrate NFR into class, sequence & collaboration diagram. Use case & scenario can be adapted to deal with NFR[1,5].UML has been extended towards Aspect J where by tools have been developed to semi automatically or automatically generate AspectJ code from UML[13,14]

8) In an attempt to formalize system requirement System requirement are captured by aspect –oriented use case diagrams & are formalizes as Aspect oriented state Chart. A formal approach with aspect-oriented statecharts is used[15]

9) A survey of the literature found that current programming languages were not designed with NFRs. new programming style into the existing object-oriented language Java has been introduced. This new method is called "Constraint and Object Oriented" programming style. The last revision of Ada language is another example of new

programming style which takes account different NFRs. Exception Mechanism is also another new style programming which is supported by many current programming languages like C++ and Java. This mechanism separates the normal control flow from the exceptional control flow under error conditions. This separation of concerns and centralized exception handling reduce the complexity of programming. Thus, exception mechanism can be viewed as a special form of policy because it provides a mechanism to specify the policies about how to handle faults. They are also another type of solution proposed by many works that combines rule based techniques and object oriented programming. For example, a new language, called R++ rules, which is an extension to C++. R++ rules , which trigger automatically upon relevant data change. One important contribution of R++ is that it introduced rule as member of class. R++ Rules are introduced as a natural extension to object-oriented classes, they support inheritance, overriding, and visibility rules.[14]

10) A strong traceability between NFRs and functional requirements can address the problems of NFR [1]

11) Early NFR Identification, isolation i.e. at requirement engineering phase. [7]

12) Handling the Aspects of different stake holder right from the Requirement Engg. phase to analysis, design & implementation phase so that the Testing Efforts can be estimated or predicted early

5.Metrics of NFR [16]

Non-Functional Requirements - Checklist

Security

• Login requirements - access levels, CRUD levels

• Password requirements - length, special characters, expiry, recycling policies

• Inactivity timeouts – durations, actions

Audit

• Audited elements – what business elements will be audited?

• Audited fields – which data fields will be audited?

• Audit file characteristics - before image, after image, user and time stamp, etc

Performance

• Response times - application loading, screen open and refresh times, etc

• Processing times – functions, calculations, imports, exports

• Query and Reporting times – initial loads and subsequent loads

Capacity

• Throughput – how many transactions per hour does the system need to be able to handle?

• Storage – how much data does the system need to be able to store?

• Year-on-year growth requirements

Availability

• Hours of operation – when is it available? Consider weekends, holidays, maintenance times, etc

• Locations of operation – where should it be available from, what are the connection requirements?

Reliability

• Mean Time Between Failures – What is the acceptable threshold for down-time? e.g. one a year, 4,000 hours

• Mean Time To Recovery – if broken, how much time is available to get the system back up again?

Integrity

• Fault trapping (I/O) – how to handle electronic interface failures, etc

• Bad data trapping - data imports, flag-and-continue or stop the import policies, etc

• Data integrity – referential integrity in database tables and interfaces

• Image compression and decompression standards

Recovery

• Recovery process – how do recoveries work, what is the process?

• Recovery time scales – how quickly should a recovery take to perform?

• Backup frequencies – how often is the transaction data, set-up data, and system (code) backed-up?

• Backup generations - what are the requirements for restoring to previous instance(s)?

Compatibility

• Compatibility with shared applications – What other systems does it need to talk to?

• Compatibility with 3rd party applications – What other systems does it have to live with amicably?

• Compatibility on different operating systems – What does it have to be able to run on?

• Compatibility on different platforms – What are the hardware platforms it needs to work on?

Maintainability

• Conformance to architecture standards – What are the standards it needs to conform to or have exclusions from?

• Conformance to design standards – What design standards must be adhered to or exclusions created?

• Conformance to coding standards – What coding standards must be adhered to or exclusions created?

Usability

• Look and feel standards - screen element density, layout and flow, colours, UI metaphors, keyboard shortcuts

• Internationalization / localization requirements – languages, spellings, keyboards, paper sizes, etc

Documentation

• Required documentation items and audiences for each item

[9]

- Time
  - Transactions / sec
  - Response time
  - Time to complete an operation
- Space
  - Main memory
  - Auxiliary memory
  - (Cache)
- Usability
  - Training time
  - Number of choices
  - Mouse clicks
- Reliability
  - Mean time to failure
  - Downtime probability
  - Failure rate
  - Availability
- Robustness
  - Time to recovery
  - % of incidents leading to catastrophic failures
  - Data corruption probability after a failure
- Portability
  - % of non-portable code
  - Number of systems where software can run

Portability

1)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)

2) the degree to which software running on one platform

can easily be converted to run on another.

Reliability

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

2) theory and practice of hardware reliability are well established; some try to adopt them for software one popular metric for hardware reliability is mean-time-to-failure (MTTF)

3)Sometimes reliability refers to the level at which a software system uses scarce computational resources, such as CPU cycles, memory, disk space, buffers andcommunication channels can be characterized along a number of dimensions:

maximum number of users/terminals/transactions

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: maximum number of users/terminals/transactions what happens when a system with capacity

e.g., "the system will generate a dial tone within 10 secs from the time the phone is picked up"

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"

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.

Metrics For product oriented Approaches are:

Product-oriented approaches

Quality 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 Of a Quality Factor is:

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

6. Heuristics of Software Testability [17]

The better we can control it, the more the testing can be automatedand optimized.

• A scriptable interface or test harness is available.

• Software and hardware states and variables can be controlled directly by the test engineer.

• Software modules, objects, or functional layers can be tested independently.

What you see is what can be tested.

• Past system states and variables are visible or queriable (e.g., transaction logs).

• Distinct output is generated for each input.

• System states and variables are visible or queriable during execution.

• All factors affecting the output are visible.

• Incorrect output is easily identified.

• Internal errors are automatically detected and reported through self-testing mechanisms.

To test it, we have to get at it.

• The system has few bugs (bugs add analysis and reporting overhead to the test process).

• No bugs block the execution of tests.

• Product evolves in functional stages (allows simultaneous development and testing).

• Source code is accessible.

The simpler it is, the less there is to test.

• The design is self-consistent.

• Functional simplicity (e.g., the feature set is the minimum necessary to meet requirements)

• Structural simplicity (e.g., modules are cohesive and loosely coupled)

• Code simplicity (e.g. the code is not so convoluded that an outside inspector can't effectively

review it)

The fewer the changes, the fewer the disruptions to testing.

• Changes to the software are infrequent.

• Changes to the software are controlled and communicated.

• Changes to the software do not invalidate automated tests.

The more information we have, the smarter we will test.

• The design is similar to other products we already know.

• The technology on which the product is based is well understood.

• Dependencies between internal, external and shared components are well understood.

• The purpose of the software is well understood.

• The users of the software are well understood.

• The environment in which the software will be used is well understood.

• Technical documentation is accessible, accurate, well organized, specific and detailed.

• Software requirements are well understood.

7.Future Research Direction.

1) Requirement conflicts Identification & resolution among NFR. by Traceability matrix or conflict resolution Matrix. Or Formal Method Specification[14,15]

2) Integration of NFR with FR at requirement elicitation stage,(Extending Use case or Formal method description) or at design stage(in architecture) [15]

3)Un ambiguous specification of NFR in SRS, one of them can be by Formal Specification of NFR .

4) Aspect Oriented Programming approach to Isolate, Identify & find testability metrics based on NFR.[12]

8.Conclusions: Trying to Estimate Testability of NFR is as difficult as trying to Stream line the fall of a fast flowing water fall from the Hill top. The informal treatment of NFR's move down from Requirement elicitation to Analysis, Design & Coding Phase which makes it impossible to identify isolate bugs the cause of software failure there by increases testing efforts.[1] Mixed status of NFR with FR results in diluted focus on NFR in a software which becomes all the more dangerous for Critical System because of which the existence of a critical system can be at stake. [2]

The Unambiguous ,clear-cut ,explicit NFR Specification may mitigate the problem of effort estimation at the later level of Testing .

Work has been done to find solution at each level.

At Specification Level: Two extremes of specification can be English like Natural Language specification or Discrete interpretation as in mathematics through Formal method Representation.

At Design level : It can be Extension of UML or Use cases for incorporating NFR in Use cases, class, Collaboration or Sequence Diagram.[1,7] or directly fusing at Architecture level[10,11,1,8]

At Implementation level instead of Using OO Programming Constructs proposals are there to use AOP ie Aspect oriented Programming approach which takes care of Crosscutting Concerns of various Stakeholders scattered or Tangled across the entire Functional Requirements. These cross cutting Concerns Are Quality factors also called Non Functional requirements.

There are several Non Formal Methods using Natural language & a variety of graphical notations. Careful application of analysis and design methods, coupled with thorough review lead to high-quality software, sloppiness in the application of these methods can creates a variety of problems. A system specification can contain contradictions, ambiguities, vagueness, incomplete statements, and mixed levels of abstraction. This becomes more dangerous & crucial for Critical System Specification.

While handling Critical system Usage of Formal Methods for System Specification seems to be the best approach to System

specification. Again application of Formal Methods is not without its D-merits of difficult & heavy cost initial cost incurring approach.

Goal Oriented Model is a midway balance Between the two Extremes of Natural Language specification & Formal Method Specification.[4]

Aspect Oriented Programming may be one of the solution towards handling these cross cutting Concerns or aspects of various Stakeholders but not without its inherent problem of identifying the weaving of concerns after identifying various Joint point or Point cuts in the Core Functionality(Or Functional Requirements).

It will be a good Idea to be able to Estimate testability of a S/w or estimating the testing effort of the software ,from the Specification or the design it self. This will ensure the Concentrated view on NFR right from the requirement elicitation phase so that early predictions of testing cost contributors can be identified.

Formalizing Functional Requirement is in itself an expensive & difficult expectation to fulfill because of Effort to cross over from Natural Human like language(full of ambiguity, inconsistency &multiple inferences) to Formal Method (Representing precise & unambiguous form due to mathematical algebraic solution).Formalizing NFR is all the more difficult & expensive .Informal & casual handling of NFR right from the req elicitation phase to later stage has made NFR difficult to handle.

This late (after thought) for NFR is the vary cause of its difficulty in handling it in Software Engineering process.

References

1. A M. R. e. Laleau, "A Survey of Non-Functional Requirements in Software Development Process," October 2008. TR-LACL-2008-7.
2. Kirsten M Hansen Anders PO Ravn" From Safety Analysis to software Requirements",July 1998.IEEE
3. Bret Pettichord "Design for Testability" Copyright © Bret Pettichord, 2002. All rights reserved.
4. B. A N. Lawrence Chung. Eric Yo, and John Mylopoulos, "NonFunctional Requirements in Software Engineering," Boston:

   Kluwer Academic Publishers, 2000.
5. J. C. S. d. P. L. Luiz Marcio Cysneiros, "Non-Functional Requirements: From Elicitation to Modeling Languages," Proceedings of the 24th International Co11ference on Software Engineering Orlando, Florida IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 30, NO. 5, MAY 2004
6. :Saeed Ullah1, Muzaffar Iq bal2, Aamir Mehmood Khan3" A Survey on Issues in Non-Functional

   Requirements Elicitation" 978-1-61284-941-6/11/$26.00 ©2011 IEEE
7. C.J. Fidge and A.M. Lister" The Challenges of Non-Functional Computing Requirements"
8. Lawrance chung,Nary Subramanian:"Process oriented Metrics for s/w Architecture Adaptability" "
9. formal method:Chumin Yang,Beum Seuk lee" Formal Specification of Non Formal Aspects in Two level grammer"
10. Barbara Paech Allen H.Dutoit" functionalrequirements,non functional requirements, and architecture should not be separated"
11. Xavier Frech Pere Botella" Putting NFR into Architecture""TIC97-1158 CICYT program
12. Marco A. Wehrmeister" An Aspect-Oriented Approach for Dealing with NFR in Model Driven Development of DistributedEmbeded Real time System"ISORC 2007 IEEE
13. Milen Guessi "Extension of uML to Model Aspect Oriented Software System"
14. Dong Kwan Kim" An AOP based performance Evaluation Framework for UML models"
15. Wen"Aformal Approach based on Aspect Oriented Statechart"IEE2010
16. Heuristics of Software Testability by James Bach, Satisfice, Inc.
17. Rogher Pressamb" a Practtsoners approach to software engineering" 6th Edition
18. Pankaj Jalote" An integrated approach to software Engineering" Third Edition.
19. Somerville: P. S. a. S. V. I. Sommerville, "Viewpoints for requirements elicitation: a practical approach," Proceedings of Third International Co11ference on Requirements Engineering 1998.