Abstract— In this paper we give the details on Non-Functional Requirements and its importance in various fields. We also investigate the effect of working out for the Non-functional Requirements which leads to the discovery of new Functional Requirements. After the coding phase in the Software Development Life Cycle (SDLC), during the process of articulating the Non-Functional Requirements the software analysts come up with the new unexpected Functional Requirements and which further creates a vicious circle. An effort is made to present the recent research in the field of finding, specifying and rectifying the Non-Functional attributes. This paper gives the details of past and current research, thus giving the ideas to researchers for future work.

Keywords — requirements, functional requirements, non-functional requirements, software development life cycle (SDLC)

I. INTRODUCTION

The term ‘non-functional requirement’ emerged way back in nineteen sixties and lot of research has been done but still, till date we have no standards of specifying, measuring and validating the non-functional requirements, often termed as NFRs are these days considered to play a pivotal role in a successful development of a software product. Although a debate is there to consider few Non-functional Requirements as Functional Requirements [31] (termed as FRs). Apart from all these problems existing in NFRs filed the developers have started making efforts in finding and addressing the NFRs and have found that while dealing with one NFR they give rise to another NFR or FR. This paper makes an attempt to discuss this inter conflict among NFRs and the intra conflicts of the NFRs and FRs. In context to the requirements engineering [21] and the Software Requirements [27] as well, one can find a lot more literature on the FRs [21], [22], [25] but rare matter on the NFRs [23], [24]. Even the IEEE Standard Glossary of Software Engineering Terminology [25] says that the term non-functional requirement is not defined and the standard distinguishes design requirements, implementation requirements, interface requirements, performance requirements, and physical requirements. Jacobson, Booch and Rumbaugh [26] defined the NFR as a requirement that specifies system properties, such as environmental and implementation constraints, performance, platform dependencies, maintainability, extensibility and reliability.

For important software applications, the Non-Functional requirements are as critical as functional requirements. The current competitive market demands software product which is not functionally fit but also implements the non-functional aspects such as: cost, readability, security, maintainability, portability accuracy etc. Dealing with the functional requirements is not enough to fulfill the customer’s demand for Quality Software product. There is a need of formally incorporating NFRs into the software development life cycle and guaranteeing their satisfaction. Developers do understand the importance of the NFRs in the complex and large systems but they also neglect due to lack of time and lack of proper documents on NFRs as well and also much time is consumed in dealing with Functional requirements often termed as FRs only. Some commonly neglected NFRs are security [11] [18] [19], fault – tolerance [12], reliability [13], maintainability [14], usability [15], accuracy and correctness [16]. This list of NFRs is not limited to but the items such as compatibility, integrity, capacity, redundancy, backups, interoperability, robustness, extensibility etc. are the topics for another discussion. A framework known as Non-Functional Decomposition (NFD) was developed by Poort that provides a model to transform conflicting requirements in a system [32].

Meeting the functional requirements in building the software application is not only sufficient. The Functionality answers to the question “what the system does” while the Non-Functionality answers to “how the same system performs”. Although the software development process is initiated with the specification of application requirements, which consists of FRs and NFRs. The FRs are addressed in various testing phases but the NFRs are very rarely taken into consideration. The interdependency between FRs and NFRs requires consideration of both FRs and NFRs throughout the software development process but in practice the NFRs are considered rarely in the late phase of the development. In the past also we found that many systems were subject to "non-functional" requirements. Also, neglecting the NFRs resulted into some serious incidents [5] previously, such as the London Ambulance tragedy [3] and Therac 25 [4] where the software calculated radiation dosage based on the order in which data was entered, sometimes delivering a double dose of radiation and it took life of ten patients. The AT & T lines went dead due to the single line of buggy code in a complex software upgrade implemented to speed up calling caused a ripple effect that shut down the network. Another incident [28] in which Intel’s highly-promoted Pentium chip occasionally made mistakes when dividing floating-point numbers within a
specific range. In one of the incident the system failed because of performance - scalability problems in the New Jersey Department of Motor Vehicles Licensing System [33]. The [5] gives a detailed elaboration of how the very complex and expensive system failed because of either neglecting the FRs and NFRs or improper management in bridging the gap and finding the balance between these FRs and NFRs. It is found that the software products developed without considering the NFRs come up with a failure range of 60% or higher.

This paper is organized as follows; Section II gives a brief introduction of various possible and important non-functional requirements which a system is frequently subject to. Section III gives the details on various works done to specify these non – functional requirements. Concluding remarks and avenues for future work are made in section IV.

II. NON-FUNCTIONAL REQUIREMENTS

It is a fact that non – functional requirements [31] are those requirements which are not expressed procedurally till date. The IEEE glossary on Software Engineering does contain the term functional requirement but not the non- functional requirement. If worked upon the below mentioned NFRs properly and in time; would result into a quality software product. Some of the most common NFRs which the software developers encounter are:

Performance deals with response time, which means the time taken by the system to load, reload, screen open and refresh times etc. The processing time which is due to functions, calculations, imports, exports. Query and reporting time is also accountable for system’s performance. The performance of a system is most common and breathe taking NFR for software developers.

Reliability of a system refers to the mean time between failures, means what can be the maximum down time? For example, two hours in six months etc. It also deals with the time taken by the system to recover if failure occurs; this is also termed as mean time to recovery.

Availability tells the operating hours of the system and when it is available.

Compatibility of a system deals with what ease the system is able to operate with shared applications. These shared applications can be the 3rd part applications as well. This also covers the system’s compatibility on different-platforms. The platforms can be either hardware, software or both.

Usability of a system refers to the standards of “look and feel”. Usability gives the measure of how much user friendly the system is. This also includes the Internationalization/localization requirements such as, languages, spellings etc.

Maintainability of a system is the level of ease with which a system can be modified or some changes may be brought into. This change or modification can be due to the addition of some new functionality to the system or for the sake of bug fixing.

Interoperability refers to the ability of a diverse system to work together with other product or systems without any restricted access or implementation.

Recovery is the ability of the software system to recover after some damage. The time taken by the system to recover back to its original shape is the recovery time.

Robustness is the ability of a computer system to cope with errors during execution or the ability of an algorithm to continue to operate despite abnormalities in input, calculations, etc.

Resilience is the ability to provide and maintain an acceptable level of service when some faults occurs and performs normally.

Further investigation also shows that neglecting NFRs in developing a software system is strongly influenced by NFR’s characteristics. It is because of the fact that the NFRs are subjective, relative, interacting, abstract and not uniform in nature.

III. HOW TO SPECIFY NON-FUNCTIONAL REQUIREMENTS

The current IT industry with the increase and sophisticated demands of clients need a mechanism to capture and specify non-functional requirements in such a way that they can drive architectural decisions and be used to validate the software architecture. The requirements engineering should be as complex and well thought out as the design and programming, yet its insufficiencies have led to many projects with poor requirements and blamed as the major reason for many software failures. Therefore, requirements engineering is now moving to the forefront gaining increased significance in software engineering for services oriented web applications. Recent years have seen an increased level of research in the field of Non-functional Requirements (NFRs) but still there is lack of proper standards to specify and measure these kinds of requirements. The Non-functional requirements (NFRs) are rarely taken into consideration in the industry in most of the software development. The reasons for neglecting NFRs are: no support from any language or tools, very rarely the NFRs are stated and that too informally. An effort made by [1]
shows the way of defining the NFRs during the software
development. This language was able to avoid the inter NFRs
conflict but was not able to solve the issue of addressing the
NFRs giving birth to new FRs. The [2] uses a Process
Oriented Approach to resolve the conflicts among the NFRs
only by using the quality characteristics of the execution
domain and there approach was limited to theories only and
not numerically. In [6] the classpects are used for integrating
the NFRs and FRs. For integrating these, the NFR Framework
is used for eliciting and analyzing the NFRs. In this approach
the synthesis of classpects is done. After this synthesis the
classes are discovered which are later integrated for the sake
of integrating the NFRs and FRs. Integrating the NFRs in to
data modeling was also done in [7]. Usually it is found that
the NFRs play no role and are not even found in the complete
software development life cycle, but in past there were some
special efforts made where architectural tactics were used to
embody the NFRs into the Software Architecture [8]. This
approach could not meet the need of solving the Quality
requirement Conflicts explained in [9] which ultimately shows
the non meeting of the FRs generation because of addressing
the NFRs. Not only in the software systems but also in the
Embedded Systems it is found that the NFRs give rise to the
new FRs. The component model for embedded system used in
[10] shows that it is only possible to verify the NFRs. The
commendable work done in [17] is only helpful in using and
representing NFRs by the process oriented approach while
how to deal with NFRs is still missing. Software applications
requirements have new characteristics causing them to change
more rapidly. This makes traditional requirements modeling
and validation methods insufficient to provide adequate
support for web applications.

Now today when the NFRs are widely recognized as a
very important aspect in effective software development, it is
found that there is hardly any literature on NFRs and hence
these NFRs are often neglected, poorly understood and not
considered adequately in software development. Many of the
studies investigating on the various practices for dealing with
NFRs also tells that the software developers do not pay much
attention to NFRs [29], [30]. In the software development
process, functional requirements are usually incorporated into
the software artifacts step by step. At the end of the process all
functional requirements must have been implemented in such
a way that the software satisfies the requirements defined in
the early stages. The NFRs if addressed are not implemented
in the same way as functional ones. To be more realistic NFRs
are hardly considered when software is built. Neglecting the
NFRs in a static software can be considered but the developers
neglect NFRs in a dynamic software architecture at their own
risk. An early detection of NFRs is useful and commendable
because it enables system level constraints to be considered
and incorporated into early architectural designs as opposed to
being refactored due to the non-meeting of customer’s
requirements and the system’s performance in at a later time.

An interesting analysis is done [34] by Mairiza D., where
the three dimensions of NFRs: (1) definition and terminology;
(2) types; and (3) relevant NFRs in various types of system
and application domains. They have also explored the relevant
NFRs to the various Application Domains in the current
market scenario. Table I gives a brief outline of various NFRs
important in various application domains.

<table>
<thead>
<tr>
<th>Application Domain</th>
<th>Relevant NFRs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Banking and Finance</td>
<td>Accuracy, confidentiality, performance. Security, usability</td>
</tr>
<tr>
<td>Education</td>
<td>Interoperability, performance, reliability, scalability, security, usability</td>
</tr>
<tr>
<td>Energy Resources</td>
<td>Availability, performance, reliability, safety, usability</td>
</tr>
<tr>
<td>Government and Military</td>
<td>Accuracy, confidentiality, performance, privacy, provability, reusability, security, standardizability, usability, verifiability, viability</td>
</tr>
<tr>
<td>Insuarance</td>
<td>Accuracy, confidentiality, integrity, interoperability, security, usability</td>
</tr>
<tr>
<td>Medical/ Health Care</td>
<td>Communicativeness, confidentiality, integrity, performance, privacy, reliability, safety, security, traceability, usability</td>
</tr>
<tr>
<td>Telecommunication Services</td>
<td>Compatibiltiy, conformance, dependability, installability, maintainability, performance, portability, reliability, usability</td>
</tr>
<tr>
<td>Transportation</td>
<td>Accuracy, availability, compatibility, completeness, confidentiality, dependability, integrity, performance, safety, security, verifiability</td>
</tr>
</tbody>
</table>
IV. CONCLUSION AND AVENUES FOR FUTURE WORK

In this paper, we have extensively reviewed the various non-functional requirements and their importance in the current competitive software market.

Summarizing, we believe that, from this survey, still a lot more work is needed to be done in this field of software engineering and researchers can come up with a formal way of defining, specifying and measuring the non-functional requirements. There is a need to come up with the prediction and estimation of the above mentioned non – functional requirements very early in the software development life cycle ultimately resulting into the profit maximization and customer satisfaction. This work would contribute to the software engineering community and to the researcher in improving the understanding the notion of non – functional requirements and also to motivate them to come up with some pre-specified standards on NFRs.

REFERENCES


[28]. http://www.willamette.edu/~mjaneba/pentprob.html


