CHAPTER 2

LITERATURE SURVEY

This chapter provides the related work that has been done about the software performance requirements which includes the sub sections like requirements engineering, functional requirements, nonfunctional requirements and performance requirements.

2.1 Requirements Engineering

Requirements for a software system are defined as the descriptions of the services that are to be provided by the system and its operational constraints. Generally, requirements reflect the customer needs. Requirements engineering is defined as, the process of identifying, analyzing, specifying and documenting the services and their constraints [41]. Creating and maintaining system requirements document is the main goal of the requirements engineering process. The requirements engineering process includes four activities. These are

- Feasibility Study
- Requirements Elicitation and Analysis
- Requirements Specification
- Requirements Validation

Feasibility Study: In requirements engineering process, feasibility study is the first activity that happens in developing a new system. It is the study of possibilities to
complete a project successfully taking into account, technological, legal, scheduling economic and other factors [42]. The inputs for the feasibility study are the purpose of the system to support the business process, preliminary requirements set and an outline of system description. Based on the result of the feasibility report one can say that whether the project is worth investing in or not.

**Requirements Elicitation and Analysis:** Elicitation is the process of seeking, uncovering, acquiring, and elaborating requirements for software systems [43]. Requirement analysis is used to analyze and model those requirements that are captured in requirements elicitation process [44].

In an organization different people take part in requirements elicitation and analysis of a software system. These people are referred as stakeholders. The main goal of requirements elicitation and analysis is to answer the question “have we got the right requirement”? [45]. The research work carried out about requirements elicitation and analysis can be found in [46,47,48,49,50,51,52,53,54, 55,56,57,58,59,60,61,62,63]. This process includes different activities that are mentioned below.

*Requirements discovery:* It is the process used by requirements analysts to identify or extract problems of software systems and solution requirements from the different users [64]. In this process requirements analyst may interact with different stakeholders to collect the requirements because each stakeholder may have different views about the system.

*Requirements classification and organization:* This is the process of organizing the requirements into different groups that are gathered from different stakeholders. Requirements can be classified based on number of dimensions [41, 65, 66].


Requirements prioritization and negotiation: As requirements gathering involves multiple stakeholders hence requirement conflicts may occur during requirements elicitation and analysis. In this process priorities are assigned to the requirements and the requirements conflicts can be resolved by negotiation with the stakeholders [67, 68, 69, 70, 71].

Requirements documentation: This is an essential and integral part of software requirements process [72]. The requirements gathered are documented and will be given as an input to the further phases of the development cycle. The requirements document obtained is either formal document or informal document [73].

Requirements Specification: A software systems’ requirements specification specifies all the necessary requirements for software system. This task explains the software system behavior which is supposed to be developed. It contains use cases, which describes the interactions among the users and the software system. A Software Requirements Specification (SRS) is a document that describes all external observable behaviors and characteristics expected of a software system. Along with functional requirements the SRS should include nonfunctional requirements. This SRS refers to the constraints imposed on the software system at all the stages such as design, implementation and testing, like design constraints, quality standards or performance engineering requirements. This SRS document provides necessary information about all the requirements which are needed for the development of a software system [74, 75, 76].

Requirements Validation: Requirements validation is the process of confirming whether the requirements gathered actually defines the software system according to the
user needs or not [77, 78]. This process plays an important role because if there are any irrelevant or improper requirements specified in the SRS document, they may lead to the wastage of time and effort in making extensive rework during the development of a system or after the system is implemented, when they are identified [79, 80].

This is the brief information about requirements engineering and requirements engineering process activities, which explains the importance of the requirements engineering process. There are two types of requirements one is functional requirements and other one is nonfunctional requirements.

2.2. Functional Requirements

The requirements can be classified into two types. One is functional requirements and other one is nonfunctional requirements [41]. Functional requirement is defined as a requirement that specifies a function that a system or system component performs [81]. The functional requirements describe what the system supposed to do and what the system should not do explicitly in some cases. The requirements depend on the type of software being developed, users who are expected to use the software and the organizational general approach taken while generating the requirements. The functional requirements are used to explain the functionality of the software system in detail such as its exceptions, inputs and outputs.

In the literature several methods have been proposed with various sophisticated tools, which supports the process of requirements analysis. The widely used methods are Goal Based Analysis (GBA) [82], Coupling Goal with Scenario [83], Use Case driven
Analysis (UCA) [84], Scenario Based Analysis (SBA) [85] and multi view approach using goal and scenario.

In use case driven analysis, the use case represents the functional requirement which is the basic unit. This approach provides the useful relationships such as dependency, association and generalization, which are important for software developers in designing and implementing the software systems. With this use case driven analysis the software developers can easily understand the structure of the system as well as its functionalities. The limitation of this approach is that these notations emphasize the requirements in the developer orientation rather than the user orientation.

Moreover, the use case analysis cannot support the process of the requirements identification. In the scenario based analysis, the scenarios are used to describe the existing system environment by providing enough context information. Based on this information the requirements can be identified and validated, and also describes agents behavior. The actual experiences with a system captured from users are the instances of scenarios. Scenarios familiar to users are analyzed as the requirements. The scenario based models are created in order to explain the software system environment, which can identify the dependencies and connections between the system and its environment. Scenario modeling may appear to be as same as the system modeling. In scenario based models it is found that the environment model or the system content depends on where the intended system boundary exists. During requirements investigation, boundaries of the system will change frequently which causes the single model of the intended system environment upon which a boundary will be imposed. Further, the alternative technical
specifications may be described within the model. The interactions among the agents in some particular context will generate a scenario. SBA does not provide any mechanism to obtain the relationships between the scenarios, therefore entire software system requirements can’t be identified at a higher level [86]. The requirements which are high-level abstractions of objectives of the system are represented and analyzed as goals in the GBA and in the coupling approach. Extraction of goals can be from various methods of gathering information which includes process descriptions such as entity relationship diagrams or flow charts or unified modeling language diagrams. From the process descriptions, goals can be identified by searching statements which seems to guide design decisions at different levels within an organization of a system. The whole system requirements represent the goal which is very abstract. By using “OR”, “AND” and “Conflict” relations the relationships between goals are expressed. During the refinement of goals these relations are identified typically.

2.3. Nonfunctional Requirements

In requirements engineering, nonfunctional requirements (NFRs) plays an important role in the failure or success of software systems. Nonfunctional requirements are defined in several ways in the literature [87]. The definition for nonfunctional requirements is “a software requirement that describes not what the software will do, but how the software will do it” [88]. One more definition of NFRs is a description of property or characteristic that a software system must have exhibit or a constraint that it must respect other than an observable behavior of the system.
Thirteen NFRs are listed by the ‘IEEE-Std 830 - 1993’, which are supposed to be included in a software requirements specification document [29]. These are

i. Performance requirements

ii. Resource requirements

iii. Documentation requirements

iv. Interface requirements

v. Operational requirements

vi. Safety requirements

vii. Quality requirements

viii. Security requirements

ix. Acceptance requirements

x. Portability requirements

xi. Reliability requirements

xii. Verification requirements

xiii. Maintainability requirements

So far NFRs are not treated as first class requirements by the majority of the researches. In fact, it is very difficult to incorporate nonfunctional requirements into the different life cycle phases of the software development. Many challenges are faced by researchers such as including large range of nonfunctional requirements, formally specifying requirements, subjective nature of nonfunctional requirements. These requirements can be incorporated into models. These models are used for resolving conflicts among NFRs and specifying functional requirements. Usually requirements are not independent but sometimes achieving one nonfunctional requirement may impact
negatively achieving another nonfunctional requirement. For example, while ensuring security using multiple passwords (one nonfunctional requirement), the performance of the system (another nonfunctional requirement) may be affected because when it checks for multiple passwords, it takes more time, which in turn slows down the performance of the system. Most of the times, nonfunctional requirements are verified only after the implementation phase, i.e., from requirements phase to implementation phase, nonfunctional requirements are not considered explicitly and directly.

Due to this problem, there is a chance of getting requirements error at the later stages of software development. Some of the well-known problems in the development of a software because of not considering some nonfunctional requirements such as dissatisfaction of system users, system discontinuation and overruns of schedule and cost, are discussed in [89]. Nonfunctional requirements will have the impact on all phases of the software life cycle. The identification of NFRs should be complete, accurate and must be done as early as possible.

Generally, in industry, functional requirements are specified at the early stage of the software life cycle and nonfunctional requirements are stated at design level or at the implementation level [90, 91]. If the nonfunctional requirements are not specified in analysis phase and specification phase, the product may get failed [91]. To overcome this problem NFRs must be considered in all the phases of the software development from the inception as per international standards. Various approaches are proposed like informal text, unstructured text and formal mathematical approaches [91]. The resources that are available and project goals are two factors to be considered in selecting an approach.
2.3.1 Semiformal and Informal Approaches

Unlike other approaches, semiformal and informal approaches do not need highly expertise persons which make these approaches at ease of use. These approaches cannot guarantee the completeness and ambiguity of the specifications obtained from system analysis. As the formal approaches are expensive and difficult to use, most of the people are used to prefer semiformal and informal approaches to specify nonfunctional requirements [92, 93].

The most popular work about nonfunctional requirements is the NFRs framework [88]. In the development process, NFRs framework considers nonfunctional requirements as soft goals which are supposed to be addressed. Goal graphs represent NFRs relations and their design decisions. In this graph the nodes represent design decisions or NFRs. Goals presented in NFRs framework can be elaborated into detailed concrete goals. This framework makes explicit relationships between intended decisions and NFRs. Each design decision may affect one or more NFRs negatively or positively which is better understood by the explicit relationship. The main purpose of this framework is that other models can reuse this framework to deal nonfunctional requirements. In [92] an approach is proposed which presents a semiformal model which is systematic and precise in nature. This model enhances the taxonomy of the nonfunctional requirements framework by combining the concept of the hard goals of NFRs in the requirements engineering process. This model provides the necessary information for early consideration of identifying, specifying, and separating functional and nonfunctional requirements.
A new informal approach is proposed to discover requirements from stakeholder point of view [93]. Requirements of one stakeholder may conflict with requirements of other stakeholder, as many stakeholders may take part in system development. The pattern communicated by stakeholders is required to coordinate the requirements among the stakeholders. Nonfunctional requirements take a vital role as it influences most of the stakeholders concerns [93]. This approach is used to reduce the dissatisfaction of stakeholder and new possibilities can be found to satisfy other stakeholders. This procedure results improved specifications which are written in sequence diagram and the evaluation tables which contains the evaluation information of stakeholders. In the evaluation table the rows represent requirements types and columns represent the kinds of stakeholders. In this table a cell represents the type of requirement by the evaluation of stakeholders and three attributes are used to label each cell, these are reference of a refined specification, a score, and the content of the evaluation. This procedure is useful to validate the specifications completeness at the instance. This procedure does not work well when the requirement engineers do not interact with stakeholders in discovering requirements.

In [94], a semi-formal approach (CMU SEI’s research result) is proposed which deals the impact of quality attributes on software architecture. The main focus of this approach is the terminology in taxonomy that can be used as a vocabulary to provide nonfunctional requirements and this vocabulary plays vital role in architectural design. The taxonomy is categorized into security, safety, performance and dependability which are considered as various areas. Methods, concerns, and factors are the dimensions to analyze the quality attributes. Firstly, methods specify how to address the requirements. Secondly, the
attributes of a system are presented by the requirements or concerns. Thirdly, system and its environment properties are considered as factors which might have effect/cause relationships and may impact requirements.

In [95], a semiformal approach is proposed for elicitation of nonfunctional requirements. This approach is also used for documenting the efficiency requirements. Quality models and quality attributes are used to capture the knowledge about nonfunctional requirements, some of which are provided in a template. This approach uses a quality model and generalized meta-model to capture the knowledge about other nonfunctional requirements.

In agile development [96], there are no explicit approaches for elicitation of nonfunctional requirements. In [97], an agile approach addresses the importance of performance requirements specification and testing which proposes a model called PREM (Performance Requirements Evolution Model). This model is used for identification and specification of performance requirements by development team. Initially these requirements are casual descriptions, and thereafter specifications are refined to desired level of detailed descriptions.

In [98, 99, 100], some approaches called misuse cases are proposed to deal with security requirements based on Unified Modeling Language (UML). Misuse cases are the descriptions of the set of sequence of actions which are not allowed by the system. In [98], a method is proposed based on misuse cases, which is used for deriving the nonfunctional requirements and functional requirements. Nonfunctional requirements analysis by means of threats, assets, counter measures and misusers is used to
complement project and software requirements. In [100] an approach is proposed for differentiating both misuse cases and security use cases. These security use cases and misuse cases are used to specify and analyze the security requirements and security threats respectively. Both of these include the properties like the semantic inconsistencies of UML, inherit popularity and simplicity.

In [101, 102] a semiformal approach is proposed, which is called Knowledge Acquisition in autOmated Specification (KAOS). It is used for elicitation of functional requirements and also used to analyze and model these requirements. This model is the combination of various sub models which are connected by means of consistency rules among these sub models with first-order temporal logic.

In [103] KAOS has been extended by Lam which handles security requirements. In this approach, for elicitation of security goals like integrity, confidentiality, availability, authentication, non-repudiation and privacy, few generic specification patterns are proposed. This approach contains two concurrent models which are incrementally built and specified to elaborate KAOS security requirements. First one is an intentional model of the system-to-be and the second one is an intentional anti-model. This anti-model yields capabilities and vulnerabilities, which are used to achieve anti-goals of major security goals which are from the original model. This method constructs threat trees incrementally by means of refinements of anti-goals till the leaf nodes are reached. The leaf nodes are either implementable anti-requirements or observable attack vulnerabilities by the attacker.
2.3.2 Formal Approaches

Elaboration method of formal nonfunctional requirements is written in a formal language is popularly used. To specify requirements using formal methods is difficult and cost effective. As it is cost effective, formal methods can be applied only to critical nonfunctional requirements like security requirements. If the nonfunctional requirements are not specified properly, it leads to a huge loss with respect to money, data and time.

In [104] Formal Design Analysis Framework (FDAF) is proposed. This framework supports the systematic design of the software system which meets nonfunctional requirements like security, performance etc. To describe software architectures various notations have been used like formal methods and UML, which supports the design analysis [105]. Requirements engineers, formal methodologists and designers, are the stakeholders who develop the software system design by using this FDAF framework so that it has to meet nonfunctional and functional requirements. The requirements specification (functional requirements and nonfunctional requirements) and an object oriented design model are the inputs for this framework. This framework used to translate an extended semiformal UML design to a formal design, and also used to select the formal method. It is difficult to study various interactions among nonfunctional requirements using FDAF because it formalizes NFRs using various formal notations. Using this FDAF, it is also difficult to detect conflicts among various types of nonfunctional requirements.

In [106, 107, 108] some formal specification languages are proposed for component based software systems to specify non-functional properties. In [107] formal
specifications of timeliness properties are proposed to describe the system by making use of temporal logic of actions in which states are represented as values which are assigned to state variables. In [106] a specification language called Component Quality Modeling Language (CQML) is proposed for Quality of Service properties of component based system.

In [109] a method for nonfunctional requirements formalization is proposed and a language called NoFun is presented. This language contains three parts, first one is the definition of software quality characteristics and attributes, in second one quality attributes of components are assigned with values, in third one both context-dependent and context-free quality requirements are specified over components. This language consists of type definition elements, structuring mechanisms. This language can also support to define non trivial quality models by other constructs. In [109], mapping of these concepts to Unified Modeling Language is studied by using some extension mechanisms. In [110] a method is proposed which describes the specification of formal requirements so that it is intended to increase the reliability and safety of the railway system by means of Z notation.

These are some approaches that are available for the analysis of nonfunctional requirements in the literature. These methods include formal and semiformal approaches. Informal and semiformal approaches are simple to use but these approaches supports less for system analysis, that means there is no guarantee that the specifications obtained are unambiguous, whereas formal approaches are difficult and expensive to use.
2.4. Performance Requirements

Performance requirements are one of the thirteen nonfunctional requirements listed by ‘IEEE-Std 830-1993’ and most important to consider in software development. A performance requirement is a statement that describes how well a function is to be executed or accomplished, or how well it is to be achieved. Performance requirements generally expressed in terms of timeliness, rate, quantity, degree and quality. Performance requirements have to be identified and specified completely in system requirements specification to avoid the problems that may occur at the end of the development of the product with respect to performance. If the product is developed with improper performance requirements specification definitely it leads to poor performance of the product. In these situations, the performance requirements have to be refined for next version of the product. So identification and refinement of performance requirements is an important task in software development. This thesis focuses on performance requirements identification and refinement.

2.4.1 Performance Requirements Identification

Performance requirements identification is the process of identifying the necessary performance requirements for the software system according to customer needs. This is an important activity in requirements analysis particularly in nonfunctional requirements analysis. The related work about performance requirements can be found in [8, 9, 10, 11, 12, 13, 14,15, 16, 17,18, 19, 20, 21, 22, 23]. In [8] Edward presents the research work about the technology which can be used for specification of performance requirements of data processing. The initial research results indicate that a technology
based on petri nets, formal logic, and simulation can be used to describe and analyze some important aspects of data processing performance requirements. The work presented in [8] is only about the technology which supports the specification of performance requirements for a specific application called ballistic missile defense system software. The intent of the approach proposed in [8] is to avoid the introduction and propagation error at design of a complicated software system not at requirements level. The work presented in [9] describes the requirements exclusively for Ada compilers and their associated tools which are used for development of real time embedded weapons system. The approach proposed in [9] is all about the technical and performance requirements of Ada compilers and linkers. It is particularly about compilers and linkers specification not about complete system performance requirements specification which is one limitation of the proposed approach.

In [10, 11, 12, 13, 14, 15, 16] Brain A. Nixon proposed a framework for managing the performance requirements for information systems. The Performance Requirements Framework (PeRF) integrates and catalogues various types of knowledge of performance and information systems. This framework includes the concepts and principles of performance. These can be used to build the performance into information systems. The framework proposed for information systems to manage performance requirements is developer oriented approach rather than user oriented. This is the one drawback because user is the most important stakeholder for any system.

In [17] an approach is proposed for analyzing the software requirements specifications for performance. In this approach the scenarios of the use cases are executed with data operations and responsibilities. Use case maps are used in specification and then analyzed
for performance. The model presented in [17] is validated only indirectly. As the model is intended to for insight into potential performance fit falls it should be validate for structure, completeness and parameter accuracy. These are the drawbacks from the proposed approach. In [18] an approach is proposed. This is named as “PeGU”. This approach is used for modeling the performance by using Goal oriented Requirement Language (GRL) and Use Case Map (UCM). This is used to evaluate the architectures qualitatively with each technology at early stages. Performance requirements are treated as soft goals and goals of GRL by PeGU. Use case maps are used for representation of scenarios.

The work presented in [18] uses goal and scenario oriented approach for addressing and refining the performance requirements. The performance requirements addressed are so abstract. They are not detailed descriptions. For example the performance requirements identified by using the approach presented in [18] are response time and space, which are represented as high level performance requirements. This may cause for difficulty in understanding the performance requirements. This approach cannot effectively support the performance requirements identification process. The work presented in [19] describes about expressing the soft bounds of performance for software systems using stochastic probes. This approach uses regular expressions to specify stochastic probes. The work presented in [19] uses stochastic probes as means of measuring soft performance characteristics over software systems. In this work an analysis is made to identify the performance bounds and reliability bounds. Performance analysis is made on system specific operations. There is no approach proposed particularly which can be used at requirements level.
There is an approach [20] for gathering performance requirements which bases on three views like deployment view, operational workload view and persistent data view. Deployment view is used to acquire service-level agreements, the hardware and software environment constraints for the deployment. Operational workload view is used to acquire several types of workloads of the software system, the intensity and pattern of requests during the various workloads, the growth projections, intervals and expected responsiveness. Persistent data view is used to capture the growth of data over a time period and the current data size. The main attributes from technological and business point can be captured by these three views. But these three views will not present all the performance requirements as many other views are there which still need to be considered. For example different stakeholders’ views have to be considered during the performance requirements identification. Use case view has to be considered to know the number of use cases and corresponding constraints with respect to performance.

In [22] a new method was proposed to predict performance requirements of mobile devices’ software tasks using system models describing the hardware and software. With the help of clustering algorithms and linear regression, behavioral models of software tasks are generated automatically. These models are used to project the runtime of representative parts of the software tasks. The runtime of representative execution parts is determined with instruction-accurate simulations which are not feasible for whole executions. The inputs for the projection task are a model of the hardware platform and input data parameters, especially the data size. In this way the method helps to seamlessly integrate the performance analysis process into the development process. The method presented in [24] is an automated method which models the performance requirements
automatically based on the behavior models which are not part of performance analysis. This method was applied for only mobile devices not generic software systems. Moreover this is not a generic approach for identification of performance requirements. The existing approaches discussed are having their own advantages and disadvantages. These are not effective in identifying the performance requirements. The analysis methods of performance requirements should be able to identify and analyze performance requirements from all different perspectives of a software system.

Due to this reason a five layered model is proposed in this thesis for the identification of performance requirements. This model is very useful in identification of performance requirements of any software system.

2.4.2 Performance Requirements Refinement

Sometimes, it may be difficult to specify performance requirements completely at the early stages of the software development process. If the product is developed without having an appropriate performance requirements specification, definitely it leads to performance problems at the end of the development. To produce the product with acceptable performance for next version, the performance requirements have to be refined. For this purpose, a very few methods are proposed in the literature such as performance requirements improvement model [24, 26], improving with field failures [27] and improving performance requirements with the “defect not a problem reports” [28]. In [24] author has proposed a framework called Performance Refinement and Evolution Model (PREM). This can be used to manage performance software development. Workload specifications and quantitative requirements are used to
distinguish the levels in model. Performance properties are specified qualitatively in test
cases and requirements at higher levels. This model is used in performance improvement
in [27, 28].

In [27] a procedure is proposed for improvement of performance requirements in
future product releases. The information obtained from field failure reports is used in this
procedure and requirements specification. This procedure uses the UML Profile for
Schedulability, Performance, and Time (UML-SPT) [111] to extract the information. The
information includes the subject of the requirement, performance measure, computation
resource, and workload. By using the information extracted new performance
requirements can be constructed. PREM is used in specifying the new requirements.

The approach proposed in [27] for performance requirements improvement uses field
failure reports from customers. Some rules are proposed for estimation of performance
requirements in this approach. These rules are so abstract rules rather than detailed.
Individual measures for refinement of performance requirements are not specified in the
rules. The team size used to test the product is 15-20 and the members of the team all are
professional testers. If the team size increases then more number of chances will be there
for uncovering the errors. If the product is tested by the common users then the results
obtained projects more of the user opinion about performance requirements rather than
developers. These are the drawbacks from the proposed approach. The approach “defect
not a problem” proposed in [28] to improve the performance requirements uses defect
reports which are considered as “not a problem” reports of defects. Professional testers
have submitted the maximum number of defect reports. Additional information was
added by the development team, which is gained during the investigation to the defect.
“Defect not a problem” is designed based on the information obtained from four elements. These are workload, subject, environment and measure. These elements are used for only the work described in [28]. In some cases these elements may not sufficient, where other types of requirements has to be considered. This is one aspect that has to be studied. Another problem is, after some releases this not a problem defect report may become actual problem. There should be strong reasons to say the defect found is not a problem.

The approaches which are described in this section are not much effective in refining the performance requirements. Due to this reason a seven layered model is proposed in this thesis for performance requirements refinement. This model is very useful for refining the performance requirements.