Chapter 4

Analysis & Mapping of Non-functional Requirements: Stakeholders' Perspective
4.1 Introduction

Difficulties in developing software are finding the right hierarchy of sub-systems based on the modular approach and adapting the sub-systems to the visualized system or vice versa depending upon Top-Down or Bottom-Up approach. Solutions of these problems are provided in an ad-hoc manner. Furthermore, sub-systems are often viewed as separate entities by having cohesive force of requirements. Software architecture groups functions with similar requirements in the same sub-system. The problem becomes more serious by the fact that most requirements can be met only by the system as a whole and not by individual sub-systems. Therefore, system's structural decisions become driving force for the realization of effective and efficient software. System's structure must be designed to address various aspects such as security, reliability, maintainability, and scalability. The visualized architecture of software has to be considered very important in the development process. As a consequence, software architecture decisions must be built around sub-systems. In practice, Sub systems description focuses only on functional requirements but it has to put emphasis on formalizing non-functional requirements also. Generally, various functional requirements such as audit trailing of system execution, services to manage application in distributed environment etc. are accommodated by architectural strategies. In order to satisfy NFRs, architectural strategies are applied to system design such as n-tier architecture, design patterns, level of granularity etc. but it is difficult to find the combination of architectural
strategies that build a system with the best fit to functional and non-functional requirements both. Bosch (2000) first obtained functionality based architecture and then applied architectural transforms to satisfy non-functional requirements. Paech et al. (2002) discussed the need for an integrated approach for functional requirements, non-functional requirements and architecture. But it is very important to consider internal and external environmental factors also while providing an integrated solution for requirements management. Therefore, in the current study, a framework is proposed, which provides architectural solution and also highlights the relationship between functional requirements, non-functional requirements, internal constraints, external constraints and architecture collectively by analyzing stakeholders’ perspective.

4.2 Mapping Non-functional Requirements using Strategy versus Requirement Matrix

To map non-functional requirements, Strategy versus Requirement matrix is proposed which considers requirements as a base for grouping of architectural entities. This matrix identifies conflicts between requirements and analyzes them using an integrated analysis of functional requirements, non-functional requirements, internal constraints and external constraints. Conventional requirement analysis methods focus on the development of a consistent requirement specification. These techniques tend to suppress conflicts with the consideration of conflict resolution and development as two separate tasks. If the conflicts are resolved separately, it could not be done at appropriate time and could lie outside the software development framework leading to an untraceable resolution. Suppressed conflicts will add communication problems in remainder
of software development process and create misunderstanding during designing and implementation of system. Proposed matrix begins with the descriptions of visualized system, as presented in Figure 4.1, to understand structural properties and architectural requirements of the system.

This requires consideration of various factors such as development process, design strategies, decomposition strategies, implementation strategies and system functionality. Architectural requirements are derived from the relevant application's properties such as concurrency, synchronization, budget, time and memory requirements. Such properties have an impact on software architecture and are not handled by single sub-system. The matrix is generated keeping in mind the fact that various constraints such as meeting time deadline, budget criteria and market conditions have a large impact on system architecture. The matrix visualizes two dimensions intersections. One dimension is the Strategy and other dimension is Requirements as shown in Figure 4.2.

Figure 4.1: Architecture Driven Requirements
The cell of matrix contains strategies from each of three strategy dimensions satisfying each of four types of requirements as shown in Figure 4.3.

Requirements are categorized into Functional Requirements, Non-Functional Requirements, Internal Requirements and External Requirements.

1. **Functional Requirements**: These requirements are targeted at the goal of the system and describe functionality of the system. Identification of these requirements requires understanding business process of an organization. e.g. all transactions should be logged in every module; system should provide audit trail of system execution, system must provide services to manage
distributed applications and availability of messaging system during working hours etc.

2. **Non-Functional Requirements**: These requirements describe how the system is to interact with users, its environment and other systems. These are also known as quality attributes such as reliability, maintainability, scalability and availability. e.g. the requirement “System must handle 1000 transactions per second” is non-functional. Another requirement “System must have less than 30 minutes downtime in every two months” is also non-functional.

3. **Internal Constraints**: Internal constraints are those, which cannot be measured by system assessment, in fact it incorporates managerial issues. For example, projected lifetime, budget limitations, time to market, technical skills availability etc. These requirements are as important to system design as functional or non-functional requirements are.

4. **External Constraints**: These constraints can be related to hardware or market related issues such as hardware availability, network resources availability, outsource options availability, natural catastrophes etc. For example, depending upon the availability of bandwidth network, mobile devices may vary in performances. These requirements affect the software development process. External environment factors may affect software development strategies as some countries have different legislations regarding security issues.

Strategy to satisfy requirements is based on three criteria: *Choice of development process*, *Design* and *Implementation*. Categorization of both dimensions, *Requirements* and *Strategy*, of the matrix is shown in Figure 4.4.
Different strategies to satisfy requirements are explained below:

1. *First* way to satisfy requirements is by making choices in the *development process*. Various models like Capability Maturity Model, use of CASE tools, Agile data Models, Object Oriented Software Process, XP, Test driven development methods etc. can be adopted to satisfy requirements of software.

2. *Second* way to satisfy requirements is by making choices in *designing* of software process. Some of the examples include decisions about logical grouping of components in the form of 2-tier or 3-tier architecture, granularity level, choice of design patterns, choice between Relational database Management System (RDBMS) and Object Oriented Database Management System (OODBMS) etc. These design strategies will make the system with more layers and of higher granularity to achieve higher performance but that will lead to conflict with maintainability.

3. Third way to satisfy requirements is by making choices in *implementation* strategy about how to provide functionality of the system. e.g. use of long encryption key or multilevel encryption to achieve security or use of menu-based interface to achieve
usability or caching functionality to achieve performance. All these types of strategies may affect certain quality requirements such as affordability and reliability. Similarly, use of Multi-level encryption will affect usability and security.

Thus conflict between various requirements emerges when strategies are applied to a single sub-system to achieve functionality of the system. Then sub system is decomposed into low-coupled parts and different strategies are applied to the respective parts that can resolve these conflicts. The decomposition of a system into low-coupled parts, as presented in Figure 4.5, should not lead to conflict between modifiability and efficiency. Therefore, a system should be decomposed into parts for modifiability and then each part should be made efficient.

![Figure 4.5: System Decomposition](image)

After decomposition, different alternative strategies are applied to the sub-system in order to achieve Functional, Non-Functional, Internal and External Requirements. These solutions are described below:

**4.2.1 Strategies to Achieve Functional Requirements**

Functional requirements directly aim at the objective of the system. Different strategy dimensions can be applied to achieve functional requirements.

1. Traditional *development* methods like waterfall model, prototype model, iterative enhancement model and spiral model can be applied to achieve goal of the system.
2. Choice of proper design strategy such as choice of programming language, normalization of database etc. affects the functionality of the system. All these decision will make the system more structured.

3. A suitable implementation strategy should be opted to satisfy user’s concern. If user prefers the system to be easy to use, menu based or GUI based interface can be the choice. To support this concern, an online help system can also be provided. If user is concerned about the privacy of data, password functionality can be provided to achieve security.

4.2.2 Strategies to Achieve Non-functional Requirements

1. Non-functional requirements can be achieved by making the choices in software development process like Capability Maturity Model, Agile Data Models, Xtreme Programming, Object-oriented Software Process, Test-driven Development Methods etc.

2. Various design options can be selected to achieve non-functional requirements. These options include choice of RDBMS or OODBMS, choice of platform, choice of Object Request Broker or Message queue, decision about logical grouping of components, break up of system into different behavioural entities to maintain level of granularity etc. All these specify the structural aspects of the system.

3. If RDBMS is selected as design mechanism, database can be implemented using Oracle or Ingres. Similarly, Orbix or VisiBroker can be opted if Object Request Broker is decided to provide communication between components.

4.2.3 Strategies to Satisfy Internal Constraints

1. Time Boxing approach can be opted to develop software if user demands the software to be delivered within in certain defined
time period. If there is need of early realization of the working of software then prototyping or Incremental Development approach can be the alternatives.

2. Choice of Object Request Broker or Message Queue design mechanism depends upon the designers' capability.

3. Depending upon the budget to develop software, Make or Buy Decision or Reuse of Software Components can be selected as implementation strategy. Implementation strategy may become specific if any technology platform is mandate for the user.

### 4.2.4 Strategies to Satisfy External Constraints

1. A proper Software development strategy can be decided to release software in different versions in order to accommodate change in the market.

2. A specific design strategy such as Object Oriented Database Management System (OODBMS) can be selected by looking at the competitor's move in the market.

3. Choice of implementation strategy of the software depends upon market conditions such as network resources availability, hardware availability etc.

Hence Choice of Development Process Strategy results in the best choice to achieve system requirements and satisfy environment constraints. Design Strategy gives structural solutions for system requirements. The combination of various Implementation Strategies provides overall system functionality. All these strategies are exhibited in Figure 4.6 to reflect relationship between requirements and architecture through Strategy versus Requirement matrix.
The highlight of the matrix is that it considers Functional, Non-functional, Internal Constraints, and External Constraints right from the beginning while deciding system structure. It integrates all requirements, development process and designing of system to handle tradeoffs between architectural and project decisions. In order to map system requirements on architecture, all architectural entities are decomposed on the basis of requirements.

### 4.3 Proposed Framework to Analyze Stakeholders’ Perspective

During the requirement elicitation process, there are many different interpretations of stakeholders’ perspectives and their inter-relationships, which lead to conflict among requirements. A methodology is needed to maintain multiple prospective and knowledge about their inter-relationships and conflicts simultaneously. Therefore, trade-offs between various FRs and NFRs are also considered in the proposed

---

**Figure 4.6: Strategy versus Requirement Matrix (Requirements and Architecture)**

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Solutions</th>
<th>Design</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Functional Requirements</strong></td>
<td>Choice of Development Process</td>
<td>Choice of programming language, Normalization of database etc.</td>
<td>Menu based interface, GUI based interface, Online help system</td>
</tr>
<tr>
<td></td>
<td>Traditional Development Methods (Waterfall Model, Iterative Enhancement model)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Non-functional Requirements</strong></td>
<td>Capability Maturity Model, Agile Data Models, Xtreme Programming, Object-oriented Software Process, Test-driven Development Methods etc.</td>
<td>Choice of RDBMS or OODBMS, Choice of platform, Choice of Object Request Broker or Message queue, Decision about logical grouping of components</td>
<td>RDBMS (Oracle or Ingres), Object Request Broker (Orbix or VisiBroker)</td>
</tr>
<tr>
<td><strong>Internal Environment Constraints</strong></td>
<td>Time Boxing, Rapid Enhancement model, Prototyping, Incremental Development approach</td>
<td>Object Request Broker or Message Queue</td>
<td>Make or Buy decision Reuse of software components</td>
</tr>
<tr>
<td><strong>External Environment Constraints</strong></td>
<td>Software release in different versions</td>
<td>Competitor's design strategies</td>
<td>Network resources availability, hardware availability</td>
</tr>
</tbody>
</table>

The matrix considers Functional, Non-functional, Internal Constraints, and External Constraints from the beginning while deciding system structure. It integrates all requirements, development process, and designing of the system to handle tradeoffs between architectural and project decisions. In order to map system requirements on architecture, all architectural entities are decomposed on the basis of requirements.

4.3 Proposed Framework to Analyze Stakeholders’ Perspective

During the requirement elicitation process, there are many different interpretations of stakeholders’ perspectives and their inter-relationships, which lead to conflict among requirements. A methodology is needed to maintain multiple prospective and knowledge about their inter-relationships and conflicts simultaneously. Therefore, trade-offs between various FRs and NFRs are also considered in the proposed
framework. During review of the literature, it was found that some of the approaches did not consider the nature of conflicts and some approaches have not clearly mentioned conflict-detection scheme. The choice of conflict detection approach is equally important to understand the nature of conflicts. In order to resolve conflicts, one has to recognize that a conflict does exist. Proposed framework, shown in Figure 4.7, emphasizes conflict detection at high level NFR based on the relationship among FRs, NFRs and constraints. It considers various quality attributes, environmental factors, internal constraints and functional aspects of the organization while transforming the high level NFR. When a conflict is detected at a high level non-functional requirement, high level NFR is then refined into low-level NFRs and again conflict is detected using proposed Strategy versus Requirement Matrix. This process is repeated until high-level conflicts are analyzed and transformed into low-level conflicts.

![Diagram](image.png)

**Figure 4.7: Proposed Framework for Understanding Semantics of Conflicts through Integrated Analysis**
These conflicts are represented in the proposed conflict tree to understand the hierarchy of non-functional requirements with conflicts which works as a rationale for further re-examination of requirements. Conflict tree helps to analyze the undesired state of a system where conflicts arise among quality attributes. It attempts to model and analyze trade-offs between requirements.

### 4.3.1 Representation of Hierarchy of NFRs with Conflicts

Conventional requirement analysis methods focus on the development of a consistent requirement specification. These techniques tend to suppress conflicts with the consideration of conflict resolution and development as two separate tasks. If conflicts are resolved separately, it could not be done at appropriate time and could lie outside the software development framework leading to an untraceable resolution. Various symbols are suggested to represent non-functional requirement hierarchy with conflicts. These symbols are represented as:

1. A high level Non-functional requirement
2. A low level non-functional requirement which cannot be further refined into another level.
3. Factors responsible for achieving non-functional requirement means which directly belong to solution domain.

Conflict tree represents high level conflicts using symbol \(\triangledown\) and low-level conflicts using symbol \(\bigcirc\). Solutions to a conflict which can be implemented is shown using symbol \(\blacksquare\).

### 4.3.2 Multifunctional Printer System: A Case Study

The case study is about the Japanese company Ricoh which provides various digital office equipments such as copiers, cut sheet printers, continuous feed printers, personal computers, network equipments
Analysis & Mapping of Non-functional Requirements: Stakeholders’ Perspective

and multi-functional products. Ricoh is an industrial partner of the competence center “Virtual Office of the Future”, founded by Fraunhofer IESE and the German Research Centre for Artificial Intelligence (DFKI) in Kaiserslautern, Germany. This case study was conducted as part of this collaboration (Denger et al. 2004). Multifunctional Printer System (MFP) is integrated office equipment consisting of printer, scanner and facsimile functionalities. As usual in the domain of embedded systems, MFP software & hardware are highly interconnected in order to achieve the required performance. In addition, due to its wide range of functionalities, requirement analysis for MFP products is complicated. Since top-level attribute is “Quality”, therefore, in this case study, quality aspect of MFP systems is reported and various requirements conflicts during manufacturing of MFPs. In this process, quality attribute hierarchy of ISO 9126 framework is referred and it is represented in the Figure 4.8.

![Figure 4.8: ISO 9126-Quality Attribute Hierarchy (ISO, 2001)](image)

Ricoh aims at monitoring of the working of multifunctional printer systems and controlling the manufacturing process of MFPs. Manufacturing process has to be Reliable and Efficient in order to produce MFPs of good quality. In order to achieve top level “Quality” attribute, two low-level NFRs Reliability and Efficiency are identified. Reliability of MFP refers to its capability to maintain a specified level of performance when used under specified conditions. Efficiency refers to
the capability of MFP to provide appropriate performance, relative to the amount of resources used, under stated conditions. Reliability requirement demands failure free operation without concentrating on the usage of amount of resources whereas efficiency concentrates on both performance and amount of resources. This gives rise to a conflict C1. Hierarchy of Efficiency and Reliability NFRs conflicts occurred in the manufacturing of multifunctional printer system is shown below in Figure 4.9.

![Conflict Tree (Monitoring and Control System)](image)

*Figure 4.9: Conflict Tree (Monitoring and Control System)*

*Reliability* requirement is furthermore refined into Maturity & Recoverability and Efficiency into Time Economy & Resource Economy. Every firm needs to manage its scarce resources to achieve resource economy. On the other hand, equipments needed to manufacture MFPs have to be highly recoverable to repair the physical damage of printer systems. This gives rise to conflicts between Recoverability and Resource...
Economy and is denoted by C1.1. In order to resolve this conflict, Recoverability and Resource Economy requirements are further transformed to other low-level requirements. Resource Economy is further transformed to Workload Distribution and Capacity requirements of various equipments such as CPU, memory, network etc. Recoverability requirement is transformed to Fault Recovery and Fault Prevention requirements. To achieve high productivity, multifunctional printer systems in the firm must give its output in short span of time but it is not necessary to have all the equipments at same maturity levels. Some equipment may be of good maturity and reliably returns output in comparison to other equipments. Thus Quality attributes in Time Economy and Maturity Requirements generate conflict C1.2. In order to resolve this conflict, Time Economy and Maturity Requirements are further transformed to other low-level requirements. Time Economy is further transformed to Workload and Response Time requirements of various equipments such as CPU, memory, network etc. Maturity of equipments can be measured by Failure Probability and Availability of equipments. If there is any fault during manufacturing of printers, Response Time depends upon Recovery Process, which gives rise to another level conflict C1.2.1. One of the Fault Prevention techniques is removal of service with fault, which may delay the completion of task assigned to equipment. That is another conflict C1.2.2. Response time depends upon the Complexity of Task and the Start up Time of task. Here we got the transformation of conflict C1.1.1 between other functional requirements such as Start Time and Task complexity. Similarly, Conflict C1.2.1 is transformed into Redundancy and Rollback functionalities. But these transformations produce another level of conflicts denoted by C1.2.1.1 and C1.2.1.2. This process continues until a list of conflicts is generated which cannot be further resolved by transforming NFRs to the next level. All conflicts are represented using conflict tree, which shows abstract and low-level conflicts. Figure 4.10 represents a conflict tree
obtained on detection and analysis of conflicts using Strategy versus Requirement Matrix.

![Conflict Tree](image)

Figure 4.10: Conflict Tree (Hierarchy of Conflicts)

At the lowest level, all non-functional requirements are compared again and detected whether there is another conflict. Through this analysis, we may find whether conflicts are mutually exclusive or partially conflicting. This tree can be used as a ground for the conflict resolution and can be recorded and verified at the later stages, if necessary.

4.4 Summary

The proposed framework not only focuses on the identification of conflicts among NFRs, also focus on the analyzing of conflicts by transforming high-level conflicting quality attributes to low level NFRs. It uses proposed Strategy versus Requirement Matrix as a technique to bring more clarity and structure in the mapping of requirements onto system architecture. The idea behind technique is to categorize requirements into Functional Requirements, Non-functional Requirements, Internal Constraints and External Constraints. The technique proposed
choice of Development Process, Design and Implementation strategies to optimize the sub-system individually to resolve conflicts among requirements due to multiple stakeholders' perspective instead of optimizing the system as a whole system. It highlighted that the best way to deal with requirement conflicts is to transform them on system structure. After detection of conflicts, high-level NFRs are transformed to low-level NFRs until a list of specific conflicts is obtained and further decomposition is not possible. After getting a list of conflicts, conflicts are represented in conflict tree, which works as a rationale for further re-examination of requirements.