CHAPTER 3

QUALITY PROCESS

3.1 Introduction
The software process is a critical factor for delivering quality software systems, as it aims to manage and transform the user needs into a software product that meets this need. In this context, software process means the set of activities required to produce a software system, executed by a group of people organized according to a given organizational structure and counting on the support of techno-conceptual tools. Software process modeling describes the creation of software development process models. A software process model is an abstract representation of the architecture, design or definition of the software process [1]. Each representation describes, at different detail levels, an organization of the elements of a finished, ongoing or proposed process and it provides a definition of the process to be used for evaluation and improvement. A process model can be analyzed, validated and simulated, if executable. The process models are used mainly for software process control (evaluation and improvement) in an organization, but they can be also used for experimenting with software process theory and to ease process
automation. Perhaps the most complex activity during application development is the transformation of a requirement specification into application architecture. The other phases also are challenging activities, but they are better understood and more methodological and technological support is available. The process of architectural design is less formalized and often more like intuitive craftsmanship than rational engineering. The domain of software architecture has received considerable attention during recent years. Some Quality Requirements are conflicting, thus making it necessary to find an architecture that provides an appropriate compromise. Architectural design is a typical multiple objective design activity where the software engineer has to balance the various requirements during architectural design. Although there are methods for analyzing specific quality attributes, these analyses have typically been done in isolation [2][3][4].

### 3.1.1 Software Process

*Software process* is a partially ordered set of activities undertaken to manage, develop and maintain software systems, that is, the software process centers on the construction process rather than on the product(s) output. The definition of a software process usually specifies the actors executing the activities, their roles and the artifacts produced. An organization can define its own way to produce software. However, some activities are common to all software processes.
3.1.2 Software Process Modelling

A software process model is an abstract representation of the software process. The two key international standards that prescribe processes for developing and maintaining software are IEEE 1974-1991 [5] and ISO/IEC 12207 [6]. Both standards determine a set of essential, albeit unordered activities, which have to be completed to obtain a software product. They do not prescribe a specific life cycle. Each organization that uses the standard must instantiate the prescribed process as a specific process. ISO/IEC 12207 presents a process of adaptation that defines the activities required to tailor the standard to an individual software project. A variety of software process models have been designed to structure, describe and prescribe the software system construction process [7][8].

The modeling of the software process refers to the definition of the processes as models, plus any optional automated support available for modeling and for executing the models during the software process. Finkelstein define a process model as the description of a process expressed in a suitable process modeling language [26]. There are other possible uses of software process models that will not be considered, such as the introduction of a new process in an organization and personnel training/motivation.

3.1.2.1 Goals and benefits of Modelling

Curtis present some of the specific goals and benefits of modeling the software process [9]:

• **Ease of understanding and communication:** requiring a process model containing enough information for its representation. It formalizes the process, thus providing a basis for training.

• **Process management support and control:** requiring a project-specific software process and monitoring, management and co-ordination.

• **Provision for automated orientations for process performance:** requiring an effective software development environment, providing user orientations, instructions and reference material.

• **Provision for automated execution support:** requiring automated process parts, co-operative work support, a compilation of metrics and process integrity assurance.

• **Process improvement support:** requiring the reuse of well defined and effective software processes, the comparison of alternative processes and process development support.

### 3.1.2.2 Software Process Models

Different process models can define different points of view. For example, one model may define the agents involved in each activity, while another may centre on the relationship between activities. There is a model that addresses the organizational culture and focuses on the behavioral capabilities or duties of the roles involved in the software
process [10][11]. This means that each model observes, focuses on or gives priority to particular points of such a complex world as software construction [12].

A model is always an abstraction of the reality and, as such, represents a partial and simplified description of the reality; that is, not all the parts or aspects of the process are taken into account in the model. Generally, a process model can be divided into several sub-models expressing different viewpoints or perspectives.

3.2 Software Process Design

According to Jacobson a design is needed in order to: Understand the system – all those involved in developing the system must understand its structure; organize the development – by breaking the system down into subsystems and defining the interfaces and their relationships; foster reuse – if components with specific functionality are specified, they can be used together in order to: make the system evolve – changes in requirements can be implemented fairly effortlessly [13]. Bass believe software design is important because it facilitates communication among stakeholders and helps in decision-making on design issues by defining restrictions involving implementation, identifying quality attributes, handling changes and using transferable and reusable models [14].

Jan Bosch method considers the design of software design taking account of the quality requirements from the early stages of development. The architectural design process, seen as an optimization problem, is viewed as a function taking as input the functional requirements specification and generating as output the architectural design [15]. In the first step, a first
version of the architecture is produced, not accounting for the quality requirements. Then, this design is evaluated with respect to the quality requirements. Each quality attribute is given an estimated value. These values are compared with the values of the quality requirement specification. If all the values are as good as or better than required, the architectural design process is finished. Otherwise, a second step transforms the initial architecture, during which, quality values for some attribute improve. This design is again evaluated and the same process is repeated, if necessary, until all quality requirements are fulfilled or until the software engineer decides that there is no feasible solution. In this case the software architect needs to renegotiate the requirements with the customer. Each transformation, generally improves one or some quality attributes, affecting others negatively.

An architectural design is concerned with structural, functional, and behavioral issues of a system by modeling the problem and the outline solution. Structural issues that influence design decisions include organization and control structures; communication, synchronization, and data access protocols; physical distribution of functions of design elements; future developments, etc. Critical architectural design decisions are made in early stages of the software life cycle. These decisions have great repercussions in the evolution of a system. In addition, when changes are necessary it often happens that the structural decisions adopted do not accommodate the change. Software architecture defines the major computational elements of a system as a means to address these architectural-level decisions.

Garlan and Shaw consider software architecture to consist of components, connectors, and configurations [16]. In this definition components are computational elements, connectors are descriptions of
the interactions between these components, and configurations are the resulting structure of topology of the architecture which can reflect the attributes of a particular architectural style. Garlan and Shaw denote architectural style as defining a family (class) of architectures that share a common vocabulary of components, connectors, and which meet a set of constraints on the topology of that style. Many organizations that engage in the development of large-scale complex software have become aware of the need for a design method and associated analysis tools that will allow modeling of systems at the architectural level of detail. Software process design should not be considered as an independent activity, but a step further in the development and evolutionary process of software products [17].
3.3.1 Activities involved in Design Process

<table>
<thead>
<tr>
<th>Activities</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Define the problem domain or context</td>
<td>Define the characteristics of the domain according to the experience of the designer.</td>
</tr>
<tr>
<td>1.2 Identify the design types</td>
<td>Identification of the main components of the system based on the functional requirements.</td>
</tr>
<tr>
<td>2.1 Identify the structure</td>
<td>Give the topology of the initial architecture.</td>
</tr>
<tr>
<td>3.1 Design the scenarios profile for a scenario based evaluation</td>
<td>Design the set of scenarios for the quality attributes</td>
</tr>
<tr>
<td>3.2 Establish the result of evaluation and analyze results</td>
<td>Determine the potential of the candidate architecture to satisfy its quality requirements</td>
</tr>
<tr>
<td>3.2 Define design decisions and feedback</td>
<td>Establish rules and constraints</td>
</tr>
<tr>
<td>4.1 Evaluate and store design decisions</td>
<td>Store the results</td>
</tr>
</tbody>
</table>

Table 3.1: Activities involved in the process design

The activities for the development process are specified in the above table. The step of the process is used as the first number to identify the corresponding activity or deliverable or for each step. Notice that an activity can be a simple one or a complex one,
requiring the application of a specific method or technique such as, for example the design of the scenarios.

3.3.2 Description of Sub-activities

Actually, each activity could be decomposed into sub-activities.

- The first activity is to define the problem domain or the context. This activity involves the description of various attributes of the domain and the relationships defined in the context to explicitly or formally describe the problem that can help the developers to achieve design as required by the intended users. After identifying the functional requirements, various components of the system are described.

- Next step in the design process is to identify the structure of the initial design. This structure will help the designer to understand how to move forward in the design process.

- After specifying the structure, next activity is to define the scenarios. A scenario is an ordered set of interactions between partners, usually between a system and a set of actors external to the system. It may comprise a concrete sequence of interaction steps or a set of possible interaction steps. Scenarios are a natural means for writing partial specifications. Every scenario captures a sequence of user-system interactions representing a system transaction or a system function from a user’s perspective. The particular strength of scenarios lies in the fact that they provide a
decomposition of a system into functions and that each such function can be treated separately.

- The initial architecture is evaluated and analyzed to identify whether it satisfies its quality requirements or not.
- Final design decision and feedback is stored by defining the various rules and constraints of the design.

3.3 Quality Process Design Framework

Only recently, the importance of design of software architecture has grown up considerably for the construction of reliable evolutionary systems. Modern applications including distribution, portability, interportability, component reusability and real time approaches require an early definition of the system architecture in order to fulfill nonfunctional requirements such as maintainability and reliability, which are crucial for the achievement of the overall functional purpose of the software system under construction.
The output of the first phase of development life cycle i.e. Requirement engineering phase is provided as the input to the process design. SRS is the final document which represents the actual requirements which are further classified in the next phase for the definition of problem domain. The activities of the design process are arranged in an iterative pattern in order to achieve high quality.
The process starts by defining problem domain and design type followed by classifying the requirements represented by SRS that leads in defining problem domain and then identifying the structure and components wise description. The next activity is subdivided in further activities. These sub activities are again arranged in a cyclic order to ensure the quality since this is the most important activity. The main activity defined above in process design is to define scenarios and architectural views. A view of the architecture is a simplified description of a system seen from a particular perspective or a point of view, making available particular knowledge and omitting entities that are not relevant from its perspective. Scenarios help in eliciting functional and nonfunctional requirements of a system.

The design strategies based on reusable architectural styles or patterns are finalized to take the process to the next step. After this, the scenarios are realized and the results are analyzed which leads to the definition of the design decisions and feedback. The above defined activity is subdivided and arranged in an iterative order so as to improve the quality of the design decisions. And if any error or fault is found in the scenario or view definition, the analysis is repeated again and classification of requirements is done again. In the last step, the final design decisions are evaluated, defined and then stored for the next phase of the development process.
3.4 Quality parameters for a development process

A stable architecture for guiding the system throughout its future lifetime is obtained iteratively and includes the identification of requirements, design, implementation and testing. Software architecture encompasses the different ways of presenting the system through components from different perspectives or points of view. However, software architecture is influenced not just by structure and behavior, but also by use, functionality, performance, flexibility, reuse, understandability, restrictions, economic and technological commitments and aesthetics [13]. Many of these are intuitive; quality attributes that any architecture must have. Bass believe software design is important because it facilitates communication among stakeholders and helps in decision-making on design issues by defining restrictions involving implementation, identifying quality attributes, handling changes and using transferable and reusable models [14].

3.4.1 Identification of Quality Parameters

Bass describe Functionality as “a system’s capacity to do the work for which it was designed or proposed.” The authors also affirm that Functionality is orthogonal to the structure and is therefore not an architectural characteristic by nature; i.e. any possible number of structures can be conceived in order to implement any functionality. The system could even exist as a monolithic component without any internal structure [14]. Therefore, Functionality does not depend on software architecture, so it must not be taken as one of its quality characteristics.
Although focused on systems in general, Jacobson state the following: “If we can be sure about anything, it is that any sizeable system will evolve. It will even evolve if it is still under development.” [13] This is possible thanks to the architecture, which is why maintainability has to be considered one of the architectural quality characteristics to be taken into account.

As far as system Reliability is concerned, it is defined as the system’s ability to remain operational over time and, like Ortega, they point out that Reliability is related to fault tolerance and the time it takes to recover, both aspects being attainable through the architecture [18]. Thus Reliability must also be considered a quality characteristic or attribute to be propitiated by the architecture.

Reasons for the faults from which the system has to recover, putting Reliability at stake is: there can be many kinds of fault (physical damage of a piece of hardware, electrical current failure, etc.), software failures (unexpected cases, invalid operations, etc.) or faults caused by external effects (request overload, unauthorized access, etc.). Although Security, according to Ortega, is part of the Functionality and the latter is not architectural, Bass consider that Security is, as they say that prevention, detection and response to such effects involve architectural strategies that may require the existence of special components to solve it [18][14].

As regards Efficiency, Bass relate it to the time required to respond to a particular stimulus (event) or the number of events processed in the same time interval [14]. These authors also say Efficiency can be measured on the basis of the amount of information and communication between system components, which clearly is an architectural characteristic, since components
can be implemented within the server layer that handles user’s requests efficiently. So, *efficiency must also be taken into account when it comes to guaranteeing product quality through the architecture.*

Thus the most important architectural quality characteristics to be taken into account are: *Maintainability, Reliability and Efficiency.*

### 3.4.2 Quality Parameters

The approach used in this work provides a framework in which global and locally defined quality criteria can be considered. Thus this can be used as a standard of excellence measure for a product quality. Important architectural quality characteristics to be taken into account while developing a process design are maintainability, reliability, security, functionality and efficiency. Quality is a multidimensional construct reflected in the quality model, where each parameter in the model defines a quality dimension.

#### 3.4.2.1 Key Quality Factors

A key to successful quality measurement is to identify those opinions of individuals that are deemed the most important, since these individuals have the most control over the quality of the final product. These opinions are referred to as Essential Views and are determined by examining an individual's expected use of the product, their experiences in developing or using similar products and their overall influence on project decisions. By concentrating on removing conflicts of opinion
between the Essential Views, a consensus can be reached as to what properties constitute quality and how quality should be measured.

In this approach, the properties that constitute the ‘explicit and implicit attributes’ of quality form a set of key quality factors (KQF) and a set of locally defined factors (LDF). The Key Quality Factors represent global quality criteria, i.e. the factors that are required for all products. The locally defined factors represent local quality criteria, i.e. additional factors identified by the essential views. The KQFs represent a common set of criteria that can be used for cross-project comparisons. The KQFs also act as a catalyst for enabling the Essential Views to identify other criteria of interest.

3.4.2.2 The Relationship Chart

Since quality factors depend on many different views, it is necessary to derive methods for determining the Essential Views from the collection of individuals associated with a project. Having identified the Essential Views, any conflicts of quality opinions between these views need to be removed. The first step of the conflict removal mechanism is implemented by use of a Relationship Chart. The relationship chart displays graphically the relationships between quality criteria as a first stage towards measuring the criteria [19].

In the Relationship Chart, each criterion is listed horizontally and vertically. Where one criterion crosses another, the relationship between those criteria is specified.
For KQFs, the relationships are fixed; they are standard. This is because the set of KQF criteria does not change, therefore the relationships between each KQF criterion does not change. The relationship chart shows the relationships between each LDF criteria and KQF criteria. In the Relationship Chart, each criterion is listed horizontally and vertically. Where one criterion crosses another, the relationship between those criteria is specified.

Given two quality criteria, Criterion A and Criterion B, the possible relationships between these two criteria are as follows:

1) Neutral: An improvement to the quality of Criterion A is unlikely to affect the quality of Criterion B.
2) Direct: An improvement to the quality of Criterion A is likely to cause an improvement to the quality of Criterion B.
3) Inverse: An improvement to the quality of Criterion A is likely to cause a degradation to the quality of Criterion B.
3.4.2.3 Polarity Profile

The second step in producing a consensus view of quality is to set the required goals for each criterion, based on the relationships identified in the relationship chart. However, there is a need to ensure that anyone can understand the graphical format chosen quickly and easily, particularly when it is considered that some essential views may belong to individuals with little technical background. There is also a need to illustrate over-engineered criteria, since further improvements in these areas will have little effect on the overall quality of the product. The solution chosen, therefore, is to use a Polarity Profile.
For each criterion, a range of values exists. The required quality of criteria is defined as a single value on a horizontal line. The actual quality achieved is also defined as a single value on the same line. The advantage of using polarity profile is that it is easily understood by anyone. Further, it is easy to determine whether or not a criterion has been over-engineered, since its actual quality value will be further advanced along the line than its required quality value. The allowable required values in the polarity profile are determined using the following rules for a given relationship type specified in the relationship chart:

1) Neutral: no rules.
2) Direct: If criteria A is greater than or equal to the value 2, then criteria B must be greater than or equal to 2.
3) Inverse: neither Criteria A nor B can be set to the value 3. If A is set to the value 4 or value 5, then B cannot be set to the value greater than 2. If A is set to the value 1 or value 2, then B cannot be set to the value smaller than 4. Only the required quality values are constrained by these rules. For each criterion, the Conversion Mechanism will probably be unique to each metric used. Since different organizations may use different metrics, no single Conversion Mechanism will be suitable in all cases.
Each organization will use different metrics and metric approaches to measure different quality attributes. These metrics may be similar, identical or entirely different to those used by other organizations.

In order to identify the Required Quality for each criterion in the Polarity Profile, the expected properties of that criterion need to be expressed using metrics. The same metrics should be used to identify the Actual Quality for that criterion. There is a need, therefore, for Conversion Mechanisms, which convert the results of metrics used to measure the quality of a criterion. However, for each criterion, the Conversion Mechanism will probably be unique to each metric used. Since different organizations may use different metrics, no single Conversion Mechanism will be

Figure 3.3: Polarity Profile
suitable in all cases. The Conversion Mechanisms used, therefore, should be agreed between the Essential Views [20].

3.4.2.4 Implementation

Having considered both the Relationship Chart and the Polarity Profile it might be useful to produce a single value of quality which may be used to indicate the overall quality of a product in terms of its required versus actual values.

A software tool is used to support this approach which allows users to create and view relationship charts, polarity profiles and overall quality scores for a product [21]. This tool also provides a definition database, allowing storage and criteria definitions. By using this point method, the actual quality of process design can be measured. The single value of quality produced by this method can be used to indicate the overall quality of the product in terms of its required and actual values. This value shows quality in terms of the percentage of quality requirements met.

The advantage of producing single value of quality is that it simplifies quality comparisons between products. But, caution should, of course, always be exercised when making such comparisons; Different Conversion Mechanisms may have been used, different Essential Views may exist and the Relationship Charts and Polarity Profiles will inevitably differ. Different methodologies, processes and resources may also have been used. The results have highlighted and confirmed the impact of the relationships between the various quality factors as well as the need to balance the requirements
and the choices that have to be made. Essential views are considered and evaluated which lead to improvement of both individual software quality factors as well as the overall quality of the project.

Figure 3.4: Quality Score
3.4.2.5 Analysis of Quality Parameters

The main three factors considered in the process design are maintainability, efficiency and reliability. The new introduced framework ensures the maintainability factor of quality. Maintainability is the capability of the software product to be modified. Modifications may include corrections, improvements or adaptations of the software to changes in the environment and in the requirements and functional specifications. The deliverables such as problem domain category, components, set of scenarios for quality attribute, rules, constraints etc. of the above defined framework are required to be of good quality.

By executing the Process Design Framework, any error or fault can be easily rectified or modified without disturbing the other activities. Reliability is the capability of the software product to maintain its level of performance under stated conditions for a stated period of time. Reliability includes maturity which includes average time between failures, operational risks, fault tolerance, recovery and security. Since any change or modification made in the framework will not affect the performance, the final product of the framework will be in accordance to the desired results.

Efficiency is the capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions. Since the main definition and revision of scenarios for quality attributes are evaluated before giving the final definition, any change or error can be easily modified without affecting the whole
process and in a short span of time, thereby increasing the efficiency of the system. This framework helps the system to increase its response time and increases the number of tasks completed in a specified time.

3.5 Conclusion

In this work, after having reviewed the general aspects of several software architecture design methods, a framework is proposed for process design. The proposed framework defines all the activities of the process design so as to improve the quality of the deliverables. It allows the specification of benchmarks against which achieved quality levels can be measured and provides guidance for building quality into software. The feasibility of quality goals is controlled by the use of a Relationship Chart and a Polarity Profile. The main three quality factors i.e. maintainability, reliability and efficiency are taken into account since they guarantee the product quality through the architecture. This framework is a step forward towards the definition of a method for architectural design that could be easily used as a customization of architectural design process or any general process framework. Therefore every process concerning modern applications should be provided with an efficient architectural design method with built in quality issues.
3.6 References


