# Pattern-Based Architectural Design Process Model

N. Lévy, F. Losavio

***Abstract:*** *The identification of quality requirements is crucial to develop modern software systems, especially when their underlying context is changing or depending over time. Much work has been done on functional requirements specification, however nonfunctional requirements have been only superficially considered. Moreover, methods for architectural design focus on quality requirements; however their specification and traceability with respect to the system's functionality is not clear. The goal of this work is to outline a pattern-based architectural design process model focused on quality requirements engineering.*

***Key words:*** *process model, quality requirements engineering, architectural design, architectural patterns*

## 1 INTRODUCTION

Software systems are requested by organizations, customers or clients to solve some of their specific problem. A document or "cahier des charges", describing the problem contains the business requirements and it is given by the client to the developers of the system. From this informal document, the software system requirements must be elicitated. Many techniques, such as the development of the Use case model [14], have been employed to identify users' requirements, which are generally called functional requirements. They are evaluated by the users when the system is executing in its final operation environment by effectiveness, safety, productivity and satisfaction issues [6]. However, at this stage it can be very expansive to correct any deficiency of the system. The fulfilment of the main final system's goals is based on the accomplishment of internal and external quality properties, which are derived from the system's nonfunctional requirements, constraining the execution of the global system's functionality. The specification of nonfunctional requirements has only recently drawn the attention of the software community, since modern applications are based on the quality of the services offered. Terms like requirements engineering and quality requirements engineering are now being employed. However, the capture or elicitation, specification and measure of requirements are still open problems. On the other hand, architectural design identifies the key strategies for the large-scale organization of the system under development [8]. An architectural style [12] is a starting point for further refinement or transformation. Architectures are the baseline on which a software system is articulated. They are solutions to particular problems described in detail, without ambiguity, and organized so that they can be understood and reused [5]. The transition from one abstraction level to another is not straightforward. The MDA (Model Driven Architecture) approach is an attempt to fill in this gap. Methods have been developed for architectural design [2, 4] and this step is now included in general software development frameworks [8]. On the other hand, all these methods consider that nonfunctional requirements are crucial for architectural design, especially when the application must respond to critical issues and to a changing environment. However, the specification and usage of requirements that drive the architectural decisions is not clear. The problem of identifying nonfunctional requirements and the related quality properties (quality requirements) needed to guarantee the accomplishment of the overall system's functionality is in general not addressed. Important issues such as the problem specification derived from the "cahier des charges", the requirements (functional and nonfunctional) elicitation, specification and measurement, the notation used to express the problem and the solutions (architectural patterns) in terms of the problem's requirements, the requirements completeness and consistency [10], must be considered by a sound software development process, to guarantee software engineering best practices.

The main goal of this work is to outline the modeling elements concerned with the architectural design process, where the engineering of quality requirements, the problem

definition and the reuse of existing solutions are crucial issues. We propose an approach to architectural design focusing on the problem to be solved [9]. The problem is described in terms of its functional and nonfunctional requirements, elicitated from the "cahier de charges" of the customer. A quality model [6] is used to specify the quality properties related with the problem's domain. At each step of the process, an architectural pattern is chosen on the basis of a quality property addressed by the pattern. A first solution is obtained by application of the selected pattern as a response to this quality property. The architecture is modeled in UML 2.0 [14]; functional and nonfunctional requirements are expressed using components interfaces and tags, providing a sound standard documentation. Consistency and correctness of the requirements can be checked using formal techniques.

The structure of this paper is the following: Section 2 describes the definition of the architectural patterns which are used in the process. The pattern-based architectural design process model is described in Section 3. The conclusion presents final remarks.

## 2 ARCHITECTURAL PATTERNS

Several patterns libraries are available [3, 5, 13]. These libraries describe the patterns focusing on the solutions proposed. But the problem description is only informally described [7]. As a consequence, it is really difficult to choose the adequate pattern to solve a current problem. We aim at providing helps to guide and document the application of architectural patterns. To do so, we add to the actual descriptions the precise definition of the problem part with both functional and nonfunctional requirements [9].

The pattern structure usually contains several clauses concerning both the *Problem part* and the *Solution part*. The problem is described within several clauses:

- Specific design problems are informally described in the *Intent* clause. We explicitly include the *Problem functionality*. A scenario may be given in the *Motivation* clause.
- *Participants* are classes or object already existing that can be used as parameters of the pattern. They are partially described or defined in the *Structure*.
- The *Applicability* clause contains a list of situations in which the pattern can be applied.

In addition, we add the following information:

- In the *Context* clause, the nonfunctional requirements.
- The new *Quality* clause contains a quality model [6, 11] related to the problem context. It is used to associate quality characteristics to functional and nonfunctional requirements. Goals may be assigned to each characteristic as a ranking: high, medium, low. They may guide the choice of a solution according to nonfunctional requirements priorities.

### Example of an architectural pattern description

**Pattern Name**: **Repository** (based on Shared Memory [12]).

**Problem definition.**

- *Intent:* Several components of a software system need to communicate directly or indirectly, that is exchange (share) potentially large and evolving data in order to meet system requirements.
- *Functional requirements*: data sharing (Figure 1).

**Structure of the problem** :

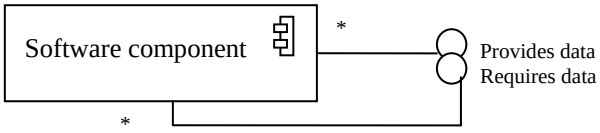


**Figure 1.** Problem: Data sharing

**Context: Communication in a distributed environment**

- Components are executed on different processors, notifications mechanisms are generally implemented to notify the concurrent components of any change of the shared repository.
- Additional mechanisms need to be provided to implement the control part. They depend on the application characteristics and the execution platform.

- *Nonfunctional requirements* and *quality model* (Table 1):
  - Data must be completely and correctly transmitted.
    Quality characteristic: reliability, maturity (robustness)
  - Limited transmission time.
    Quality characteristic: efficiency, time behavior (performance)
  - Communication must be flexible to meet changing requirements, since relationships between components can evolve statically and dynamically.
    Quality characteristic: maintainability, changeability (flexibility)
  - Components can be changed or replaced over time:
    Quality characteristic: reliability, consistency

| **Nonfunctional requirements** | **Quality Characteristics** | | |
|---|---|---|---|
| | *Reliability* | *Efficiency* | *Maintainability* |
| *Data must be completely and correctly transmitted* | - *maturity*: mechanisms to avoid failures should be introduced to assure robustness<br>- Attribute: presence of a mechanism<br>- Metrics: Boolean<br>- Goal: High | | |
| *Limited transmission time* | | - *performance* with respect to time behavior<br>- Attribute: latency<br>- Metrics: percentage [0..1] | |
| *Communication must meet changing requirements* | | | - *changeability* flexibility of the components relationships<br>- Attribute: size<br>- Metrics: measure of complexity |
| *Components can be changed or replaced over time* | - *consistency*: a mechanism (e.g. to replace the interface of the component) must be provided<br>- Attribute: presence of a mechanism<br>- Metrics: Boolean<br>- Goal: High | | |

**Table 1:** Quality model for data sharing in a distributed context

The UML architectural description is decorated with tags denoting the characteristics from the Quality Model.

**Structure of the problem (Figure 2)**:

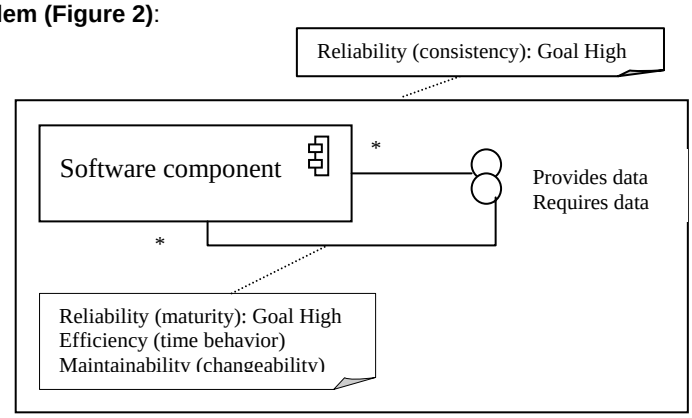


**Figure 2.** Data sharing in a distributed context

**Structure of the solution in a distributed environment (Figure 3)**:

- A set of software components, containing the knowledge of the domain, communicate to each other to meet system requirements. They do not know each other (indirect communication); they are only defined by their needs to perform the computations (their inputs) and the results they can provide (their outputs). When a component produces some information that is of interest for other components, it stores it in the shared repository. The other components will retrieve it if needed.
- A repository that is accessible by every component (read and write accesses). This repository can store all the data that need to be exchanged by components during system execution.

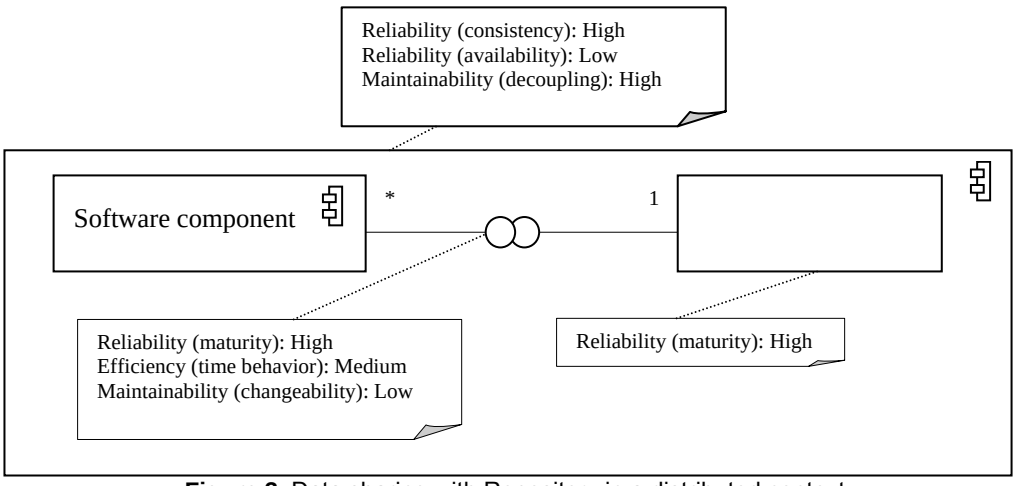


**Figure 3.** Data sharing with Repository in a distributed context

## 3 THE ARCHITECTURAL DESIGN PROCESS MODEL

The SPEM (Software Process Engineering Metamodel Specification) notation [14] in Figure 4 is used to represent our architectural design process model (process package). The model elements involved are also specified.

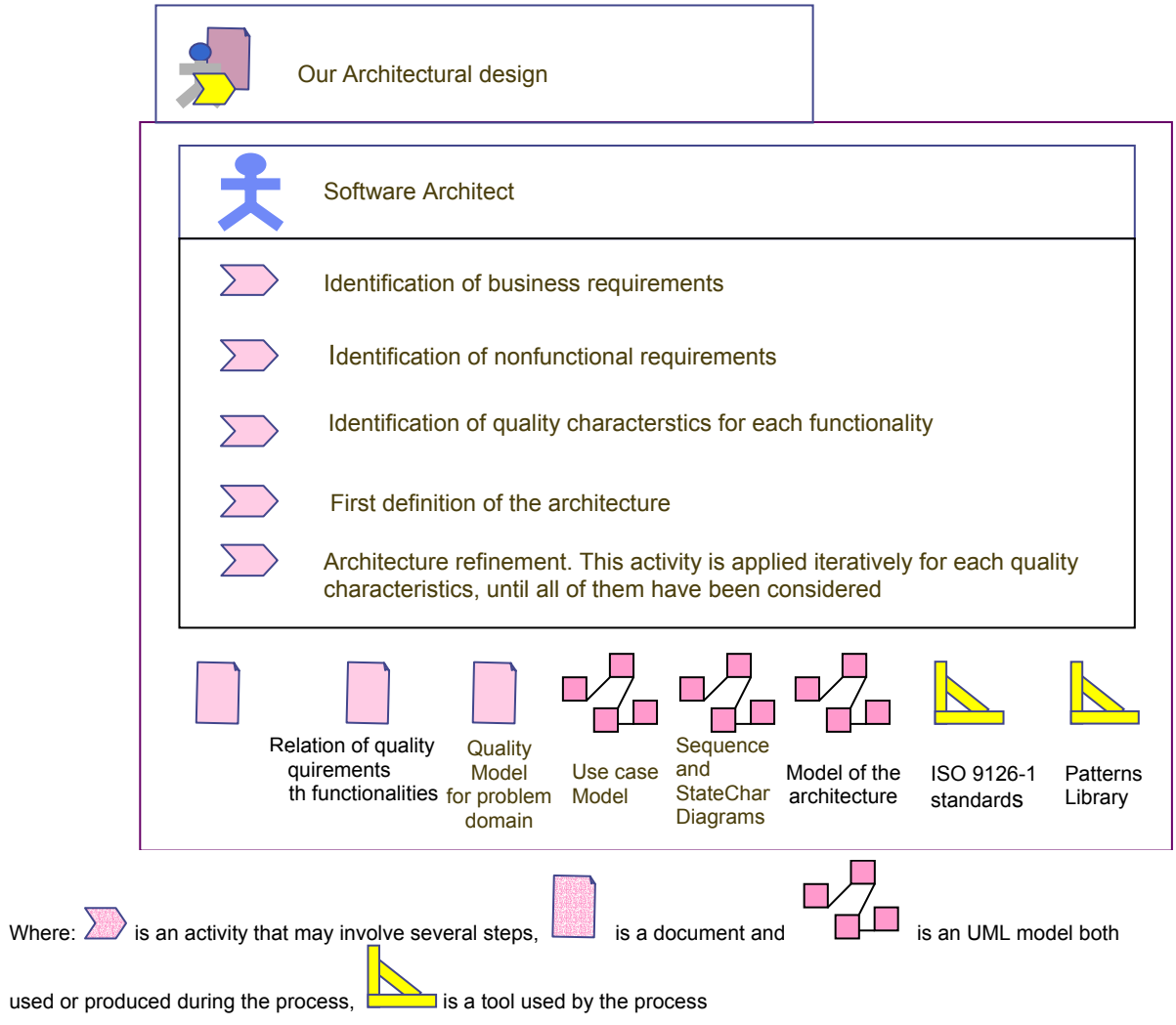


**Figure 4**. Architectural design process model package

Requirements are the base on which the software development process is built. In our approach they can be elicitated using a four step process [10]. A classification of requirements has been added to facilitate the identification of the nonfunctional requirements and derive the corresponding quality properties [11]. This method supports

two phases to perform the early stages of the software life cycle in a systematic way, namely requirements elicitation and formal specification development. It starts with a brainstorming process where the problem domain and the requirements are described in natural language [10]. The basic idea behind the proposed architectural design process is to focus on the problem and not to go straight to the design of its solution [1, 7]. In our case, the problem statement is characterized by both its functional and nonfunctional requirements. The functional requirements are generally derived from the user's needs and the nonfunctional requirements are more related with the problem's environment or context, which can have different views according to the stage of the development, such as the problem's real world and the system's operational environment. Each functionality is associated with a quality goal that must be satisfied to ensure the accomplishment of the functionality in the final software system, running in a specific operational environment. Moreover, when a nonfunctional requirement is formulated, it implies that a new functionality or implicit functionality, such as the handling of transient connections, has to be considered in the applications designed for such an environment.

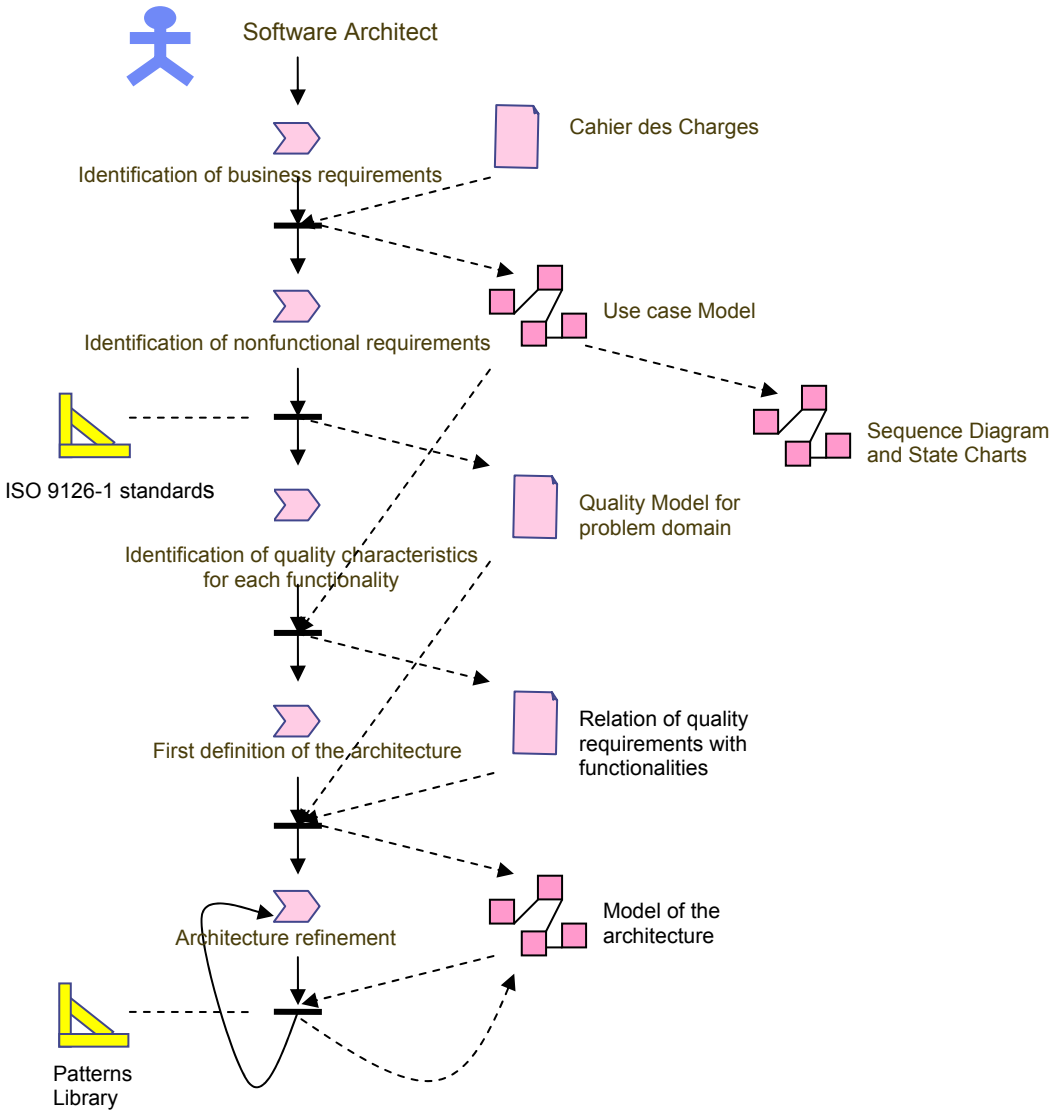


**Figure 5.** Architect activity diagram

The quality properties related to functional and nonfunctional requirements are specified in the quality model associated to the problem domain (see Table 1). The solutions can introduce new quality features that must be added to this quality model. At the end of this process, the architecture has been developed satisfying the nonfunctional

requirements. The complete pattern-based architectural design process is presented in Figure 5 as the activity diagram of the Architect.

## 4 CONCLUSION

In this paper, we have outlined the modeling elements involved in an architectural design process, which uses quality requirements engineering and focuses on the problem definition and the reuse of existing solutions. We observe that standards that are now recommended in software engineering best practices are useful tools, providing in general a rich documentation. However, there is a need for a "mature" literature on standards. The lack of standard patterns library is still a serious drawback to architectural design.

## 5 REFERENCES

[1] Alexander C. The timeless way of building, Oxford University Press, 1979.
[2] Bosch J. Design and Use of Software Architecture, Addison Wesley, Harlow, England, 2000.
[3] Buschmann F., Meunier R., Rhonert H., Sommerlad P., Stal M. Pattern-Oriented Software Architecture. A System of Patterns, John Wiley & Sons, New York, 1996.
[4] Clements, P., Kazman, R. and Klein, M. "Evaluating Software Architecture. Methods and Case Studies". SEI Series in Software Engineering. Addison-Wesley, 2002.
[5] Gamma E., Helm R., Johnson R.. Vlissides J. Design Patterns. Elements of Reusable Object-Oriented Software, Addison Wesley, Reading, Massachusetts, 1995.
[6] ISO/IEC 9126-1 "Software Engineering - Product Quality. Part 1: Quality Model", 2001.
[7] Jackson, M., Problem Frames, Addison Wesley, Harlow, England, 2001.
[8] Krutchen P. The Rational Unified Process, Addison Wesley, Reading, Massachusetts, 1999.
[9] Levy N. Losavio F. Architectural Choices for Dependable Systems, to appear in ICSE-WADS 2004 proceedings, Edinburgh, May 2004.
[10] Levy N., Marcano R,, Souquières J., From requirements to formal specification using UML and B, International Conference on Computer Systems and Technologies – CompSysTech'2002.
[11] Losavio F., Chirinos L. Matteo A., "Identifying Quality-Based Requirements", Information Systems Management (ISYM), Auerbach Publications, Vol. 21, No. 1 (15-21), Winter 2004.
[12] Shaw M., Garlan D., Software Architecture. Perspectives on an Emerging Discipline, Prentice Hall, New Jersey, 1996.
[13] Schmidt D., Stal M., Rhonert H., Buschmann F., "Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects", Vol 2, Wiley, Chichester, 2000.
[14] UML Resource Page, http://www.omg.org/UML

## ABOUT THE AUTHORS

Prof. Nicole Lévy, PRISM, Université de Versailles, Phone: +33 139 25 43 12, E-mail: Nicole.Levy@prism.uvsq.fr

Prof. Francisca Losavio, LaTecS, Universidad Central de Venezuela, Phone: +58 212 753 69 84, E-mail: flosav@cantv.net