A CRITICAL ANALYSIS OF INTEGRATION OF NON-FUNCTIONAL REQUIREMENTS INTO DESIGN PATTERNS FOR SAFETY-CRITICAL COMPUTER SYSTEMS

Dr.Gandi Satyanarayana[1]
Professor, Department of Computer Science & Engineering, Avanthi Institute of Engineering and Technology
Makavarapalem, Visakhapatnam, Andhra Pradesh, India

Dr Kadupukotla Satish kumar[2]
Associate Professor
Department of Information Technology
St Peters Engineering College, Hyderabad-500010, India

Abstract

Resilient Design Patterns is of considered opinion of the experts in the field of software engineering for safety. Thus the need for robust design Patterns towards concise and compact solutions for compromising safety aspects of design issues both in software and hardware designs. Therefore it is necessary to account for functional and non-functional requirements in the design patterns of safety-critical computer systems.

This work focuses on integration of non-functional requirements & implications to the prevailing design pattern concept. In order, this paper proposes a new pattern for safety-critical computer application through rigorous study of design methods by including fields for the implications, complications and unforeseen peripheral issues of the traditional design patterns based on non-functional requirements of the entire systems. The paper also considers requirements including designs for safety, reliability, modifiability, cost, and execution time. The overall study indicated and showed improved results that the proposed new design pattern can be of better use in safety-critical computer systems.

Keywords: Design Pattern, Embedded Systems, Non-Functional Requirements, Safety-Critical Computer Systems

Notations:

R_{AC}: The reliability of the actuation channel.

(R_{old}= R_{AC})

R_{SC}: The reliability of the fail-safe processing channel.

R_{SE}: The reliability of the safety executive component.

C: The coverage factor which is defined as: the probability that a fault in an actuation channel will be identified by the safety executive and the fail-safe processing channel will be activated.

R_{new} = The reliability after using this patteren

R_{old} = The reliability of the basic system

I. INTRODUCTION

Design pattern, originally proposed in (Acharyulu, 2015) by the architect Christopher Alexander, is a universal approach to describe common solutions to widely recurring design problems. Ever since, this concept has been applied in several domains of hardware design (electronics) and also became popular in the software domain after the success of the book Design Patterns: Element of Reusable object oriented Software by Gama et al. [2].

As the concept of design pattern aims at supporting designers and system architects in their choice of suitable solutions for commonly recurring design problems, this concept might also be useful to support the design of safety-critical embedded systems. The design of these systems is considered to be a complex process, as hard- ware and software components have to be considered during the design as well as potential interactions between hardware and/or software components. Moreover, not only functional requirements[1] have to be fulfilled by these systems. Failures in safety-critical systems could result in critical situations that may lead to serious injury or loss of life or unacceptable damage to the environment. Therefore, also the non-functional requirement safety has to be

considered in these systems to assure that the risk of hazards is acceptable low in the considered system. To support the design of safe devices, safety measures are given by international safety standards as the IEC61508 [3]. Beside life cycle and process requirements, also different measures for the design of software and hardware components are recommended. These safety measures have typically an impact on the cost, the reliability, the real time behavior of the system, and on the modifiability of the resulting system. Depending on the application domain of the later embedded system, these non-functional requirements are of great importance. For this reason, nonfunctional requirements should be considered during the design of any safety-critical system.

While current concepts of design pattern exist for many different application domains, they typically lack a consideration of potential side effects on nonfunctional requirements. In order to integrate these side effects into the pattern concept, we propose an extended template for an effective design pattern representation for safety-critical applications.

This pattern representation includes the traditional pattern concept in combination with an extension describing the implications and side effects with respect to the non-functional requirements. While this concept has been described briefly in [4] before, this work focuses on the application of our approach. Thus, two example patterns are included to illustrate the proposed representation of design patterns for software and hardware components in safety-critical applications.

Section II discusses about the related work in discipline, Section III describes about the outline of Design Patterns, Section IV details various implications of non-functional requirements while integrating into design patterns, Section V illustrates about some of the example patterns, Section VI deals with the integration with implications in integration of non-functional requirements into design patterns, and concluding remarks are given in Section VII

II. RELATED WORK

The field of design pattern is large and still rapidly growing. Many researchers have focused on the use of design pattern in the software domain, but further research is still needed in the domain of safety-critical systems to integrate the non-functional requirements in design patterns. In his books [10] and [11], Bruce Douglass proposed several design patterns for safety-critical systems based on well known fault tolerant design methods and by integrating some modification to increase the safety level on these patterns. Gross and Yu [12] discuss the relationship between non-functional requirements and design patterns, and propose a systematic approach for evaluating design patterns with respect to nonfunctional requirements.

They propose the use of design patterns for establishing traces between non-functional goals in a goal tree such as a soft goal interdependency graph (SIG) and the system design. Cleland-Huang *et al*. enhance the patterns defined by Gross and Yu [12] through defining a model for establishing traceability between certain types of non-functional requirements and design and code artifacts, through the use of design patterns as intermediary objects. Xu [15] classified the dependability needs into three types of requirements and proposed an architectural pattern that allows requirements engineers and architects to map dependability requirements into three corresponding types of architectural components. Konord [16, 17] describe a research of how the principle of design pattern can be applied to requirements specifications, which they term requirements patterns for embedded systems. They include a constraints field in the pattern template to show the functional and non-functional restrictions that are applied to the system.

In comparison to our work, none of the aforementioned approaches show clearly the implications on the non-functional requirements as part of the pattern. These patterns and the other

developed patterns focus on the traditional structure of the pattern that includes: context, problem and solution. The use of non-functional requirements in these approaches is restricted to the requirement analysis phase of the design process. In these approaches, neither a relative measure nor an indication for the implications of the patterns on the non-functional requirements, were given. To improve these approaches, we propose a new template representation in Section 4 to show the implications of the represented patterns on the non-functional requirements.

III. DESIGN PATTERN OUTLINE

In this section, the template pattern we propose for the representation of design patterns for safety critical embedded applications is described. As depicted in Figure 1, the upper part of the template includes the traditional representation of a design pattern while a listing of the pattern implications on the non-functional requirements is given in the Implication section. Moreover, further support is given by stating implementation issues, summarizing the consequences and side effects as well as a listing of related patterns.

Figure-1. The design pattern outline

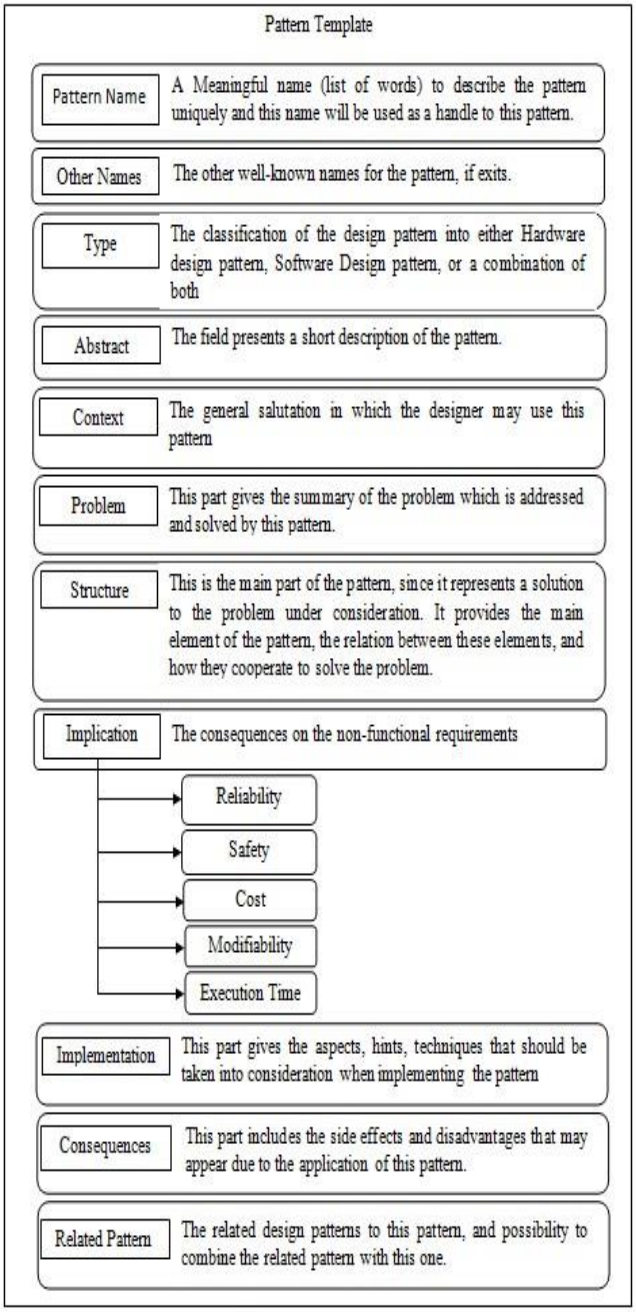

Figure 1. The design pattern template

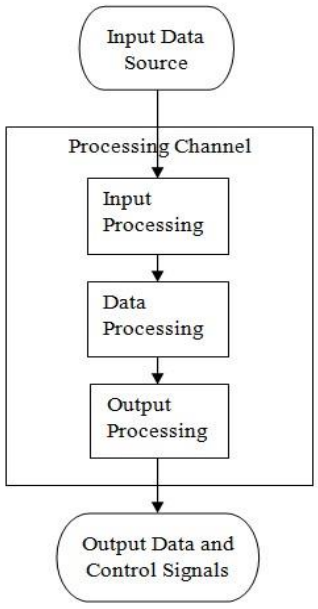

Figure 2. Basic System without specific safety requirements

The proposed design outline includes a part for pattern implications on the non-functional requirements reliability, safety, cost, modifiability and execution time. To allow a suitable description of these implications, the changes/improvements of using the corresponding design pattern are represented relative to a basic simple system (Figure 2). This basic system has a given reliability (R_{old}), a given cost, a given modifiability and is resulting in a given executing time. Moreover, this basic system has no specific safety measurements.

IV. THE IMPLICATION ON NON-FUNCTIONAL REQUIREMENTS

While the main part of the design pattern proposed does not differ from well known approaches [18, 19, 20, 21], the part for the implications on the non-functional requirements is described in this section. As mentioned above, the implications are stated relative to the basic system without any specific safety method. In the following, the determination of the five implications on nonfunctional requirements is described:

Reliability: In this context, reliability is defined as the probability that of a system or component to perform its required functions correctly under stated conditions for a specified period of time. This part of implications describes the relative improvement in the system's reliability relative to the maximum possible improvement in reliability, which is defined in the following equation:

$$\text{Reliability Improvement} = \frac{R_{new} - R_{old}}{1 - R_{old}} * 100\%$$

R_{new} = The reliability after using this patteren

R_{old} = The reliability of the basic system

Safety: The safety of a system is usually determined by the residual risk of operating this system (see e.g. [3]).Therefore; the notion of risk can be used as a measure for the assessment of safety-critical systems. The problem concerning design patterns is that they describe an abstract solution to a commonly recurring design problem. As it is not related to a specific application or to a specific case, it is difficult to determine an actual value for the possible residual risk without considering a concrete application. To allow an indication of the safety that can be achieved by the application of a specific design pattern, existing recommendations given in safety standards are used.

In detail, it is stated to which Safety Integrity Level (SIL) the pattern is recommended in a given safety standard. The safety integrity levels used here include the levels SIL1 to SIL4 as they are defined in the standard IEC61508 [3]. Additionally, the notation SIL0 is used in this template to describe a system without specific safety requirements. If measures are described in design patterns that are not included in current safety standards, these measures have to be assessed in an appropriate manner, e.g. by comparing them to measures with known recommendations.

***Cost*:** The implications on costs include: The recurring cost per unit, which reflects the additional costs resulting from additional or specific hardware components required by the design pattern and the development cost of applying this pattern.

Modifiability: This implication describes the degree to which the system developed according to this design pat- tern can be modified and changed.

Impact on execution time: With this implication, the effect of the pattern on the total time of execution at run- time is indicated. It shows the execution time overhead that is resulting from the application of this pattern in the worst and the average cases.

The application of the design pattern proposed, especially the use of the implication part introduced briefly in this section, is described in form of two example patterns in the following section.

V. EXAMPLE PATTERNS

Two example patterns are presented in this section to illustrate the application of the proposed approach: The first pattern is a hardware and software pattern that is expected to be suitable for complex and highly safety-critical systems. The second pattern is a hybrid software fault tolerance method intended to increase the reliability of the standard N-version programming approach (Acceptance Voting Pattern).

1. Example 1

In this example pattern, the pattern originally described in [10] is presented in our extended pattern representation including also implications on nonfunctional requirements.

Pattern Name

Safety Executive Pattern (SEP)

Other Names

Safety Kernel Pattern

Type

Hardware and Software

Abstract

The Safety Executive Pattern can be considered as an extension of the Watchdog Pattern[4] targeting the problem that a shutdown of the system by the actuation channel itself might be critical in the presence of faults (shutdown might fail or take too long). This problem occurs especially in those systems in which a complicated series of steps involving several components is necessary to reach a fail-safe state. Therefore, the Safety Executive Pattern uses a watchdog in combination with an additional safety executive component, which is responsible for the shut-down of the system as soon as the watchdog sends a shut down, signal (see also Figure 3. the safety executive pattern). If the system has a safe state, the

actuation channel is shutdown via the safety executive component. Otherwise, the safety executive component has to dele- gate all actuations necessary to an additional fail-safe processing channel.

Context:

The application of this pattern is suitable in the following context:

- The considered actuation channel requires a risk reduction by safety measures.
- The considered system has at least one safe state. If this is not the case, an additional fail-safe processing channel has to be applied to overtake necessary actions.
- A shutdown of the actuation channel is complex. As an example, this is the case if several safetyrelated system actions have to be controlled simultaneously.
- A sufficient determination of failures in the actuation channel can be achieved by a watchdog.

Problem

Provision of a centralized and consistent method for monitoring and controlling the execution of a complex safety measure (shutdown or switch over to redundant unit in case of failures).

Pattern Structure

The Safety Executive Pattern is based on an actuation channel to perform the required functionality and an optional fail-safe processing channel that is dedicated to the execution and control of the fail-safe processing (see also Figure 3). The central part of this pattern is the existence of a centralized safety executive component coordinating all safety-measures required to shut down the system or to switch over to the fail-safe processing channel. The safety executive component can also be used to control multiple actuation channels in the system that each may have multiple channels.

The components of the pattern depicted in Figure 3 are described below:

- **Input Data Source**: This component represents the source of information that is used as input to the system under consideration**.** Typically, this data comes either from the system user or from external sensors that are used to monitor environmental variables such as: temperature, pressure, speed, light, etc...
- **Actuator(s):** Actuators are the physical devices that perform the action of the channel like: motors, switches, heaters, signals, or any other device that performs a specific action. Often, there are more than one actuator in a single channel.
- Actuation Channel: This channel represents a sub- system that performs dedicated tasks in the overall system by taking an input data from the input data source, per- forms some transformation on this data, and then uses the results to generate suitable commands to drive the actuators.
- **Fail-Safe Processing Channel:** This is an optional channel; it is dedicated to the execution and control of the fail-safe processing. In the presence of a fault in the actuation channel, the safety executive turns off the actuation channel, and the fail-safe channel takes over. If the System doesn't have a fail-Safe Channel, then the actuation channels must have at least one reachable safe state.
- **Data Acquisition (Input Processing):** This part of the channel collects the raw data from the input data source and may perform some data formatting or transformations.
- **Data Processing (Transformation):** This part may contain multiple data transformation components; where each one performs a single transformation or processing on the received data to execute the desired algorithm in order to generate the required control signals. The final component of this part sends the computed output to the output processing unit.
- **Output Processing:** This unit takes the computed data from the data transformation unit and generates the final data and the control signals to drive the actuators. It can be considered as a device driver for the actuator.
- **Integrity Check:** This is an optional component that is invoked by the watchdog to run a periodic Built-In Test (BIT) to verify all or a portion of the internal functionality of the actuation channel.
- **Time Base:** This is an independent timing source (timing circuit) that is used to drive the watchdog. This time source has to be separate from the one used to drive the actuation channel**.**
- **Watchdog (WD):** The watchdog receives liveness messages (strokes) from the components of the actuation channel in a predefined timeframe. If a stroke comes too late or out of sequence, the watchdog considers this situation as a fault in the actuation channel and it issues a shutdown signal to the actuation channel or initiates a corrective action through sending a command signal to the optional integrity check. If the system contains multiple actuation channels, then it may contain multiple watchdogs, one per actuation channel.
- **Safety Executive:** This is the main component in this safety executive pattern. It tracks and coordinates all safety monitoring to ensure the

execution of safety action when appropriate. It consists of a safety coordinator that controls safety measures and safety policies. The safety executive component captures the shutdown signal from the watchdog in the case of failure in the actuation channel.

- **Safety Coordinator:** The safety coordinator is used to control and coordinate the safety processing that is managed by the safety measures. It also executes the control algorithms that are specified by the safety policies.

- **Safety Measures**: Include the detailed description of the safety measures. The safety coordinator may control multiple safety measures**.**

- **Safety Policies:** Each safety policy specifies a strategy or control algorithm for the safety coordinator. It involves a complicated sequence of steps that involve multiple safety measures. The reason for the separation of the coordinator from the safety policies is to make the process of changing and adapting a safety policy easier.

Implication

 This section describes the implication of this pattern relative to the basic system without a specific safety method.

Reliability

Let us have the following notations:

R_{AC}: The reliability of the actuation channel. (R_{old} = R_{AC})

R_{SC}: The reliability of the fail-safe processing channel.

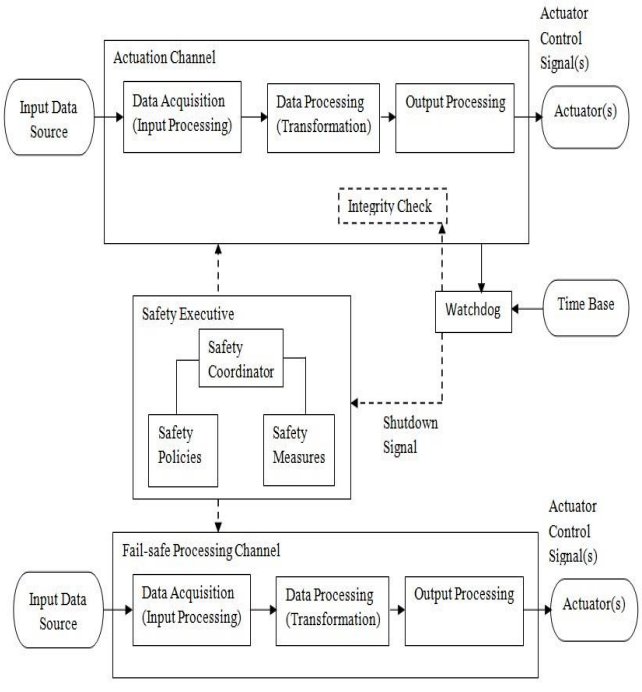

Figure 3. The safety executive pattern

R_{SE}: The reliability of the safety executive component.

C: The coverage factor which is defined as: the probability that a fault in an actuation channel will be identified by the safety executive and the fail-safe processing channel will be activated.

Assume that the watchdogs are carefully designed with reliability=1.

The safety executive pattern will continue to work without system failure as long as one of the following two conditions holds:

- There is no fault in the actuation channel.
- There is a fault in the actuation channel and the watchdog detects this fault and the safety executive initiates a shutdown or activates the fail-safe processing channel.

The new reliability after using this pattern (Rnew) is equal to:

$$R = R_{AC} + CR_{SE}(1 - R_{AC})R_{SC}$$

In this equation, the first term represents the reliability of the actuation channel while the second term represents the reliability of the remaining parts in the case of failure in the actuation channel.

Figure-3. The Safety Executive pattern

The percentage improvement in reliability relative to the maximum possible improvement is equal to:

$$= \frac{R_{AC} + CR_{SE}(1 - R_{AC})R_{SC} - R_{AC}}{1 - R_{AC}} \times 100\%$$

Safety:

The safety executive pattern includes the following four design techniques: program sequence monitoring with a watchdog, test by redundant

In general, we think that the combination of these techniques and the development cost makes the safety executive pattern suitable and highly recommended only for very high critical applications with high safety integrity levels (SIL4 and SIL3) and recommended for lower levels (SIL2 and SIL 1).

Cost:

This pattern is an expensive pattern with very high cost since it consists of different components that involve high recurring and development cost.

 Recurring cost: It includes the cost of the following:

hardware (the watchdog that initiates the integrity check and BITs), safety bag techniques, and graceful degradation. According to the standard IEC 61508 [3], the recommendation for these techniques is shown Table 1.

Techniques	SIL1	SIL2	SIL3	SIL4
Program sequence monitoring	HR	HR	HR	HR
Test by redundant hardware	R	R	R	R
Safety bag techniques	---	R	R	R
Graceful degradation	R	R	HR	HR

- The actuation channel.
- The fail-safe processing channel (if present).
- The safety executive component.
- Watchdogs and their independent timing source.

Development cost: In general, the development cost for this pattern is very high since it includes a development of three different systems (channels) hat including different architectures and different designs.

Modifiability:

There are two types of possible modifications:

1) Actuation Channel: It is very simple to modify this pattern by adding extra functionality to the actuation channel. The only things that should be done: is to know whether the new components need to send stroke messages to the watchdog.

2) Safety policy: One of the main features of this pattern is the centralized safety processing which is performed by the Safety Executive Component. The Safety Executive separates the coordinator from the safety policies to simplify the change and modification of the safety policy and to make it easier. **Impact on execution time:**

The actuation channel and the safety executive have different CPUs and different memories, and they run simultaneously in parallel. Thus, there is no effect for the safety executive component on the actuation channel during the normal operation of the system except the execution of the periodic built in tests.

Implementation

The following points should be taken into consideration during the implementation of this pattern:

- The actuation channel, the safety executive, and the fail-safe processing channel run separately in parallel, therefore each channel will run on its own processing unit and own memory.
- The safety-critical information must be protected against data corruption, e.g. by using CRCs or any other method to detect data errors.
- The watchdog component is simple and often implemented as a separate hardware device. It is capable of detecting a variety of hardware and software fault. However its actual diagnostic coverage depends on the *i*ntegrity check implemented in the actuation channel.
- To provide protection from faults in a common time base, separate timing sources must be used for the watchdog, the safety executive and the actuation channel.

Consequences and Side Effects

The main drawback of this pattern is the high complexity of this pattern for implementation. Therefore it is used for complex and highly safetycritical systems.

Related Patterns

The safety executive pattern is used for complex safety critical applications and it covers a large set of features, provided by of the other patterns, such as sequence monitoring provide by watchdog, switch-tobackup as in the fail-safe channel. For simpler systems with simpler safety requirements, other simpler patterns, such as Watchdog pattern, Sanity Check pattern and Monitor Actuator pattern [11], can be used.

VI. CONCLUSIONS

The design of safety-critical embedded applications requires an integration of the commonly used software and hardware design methods.

Therefore, the use of design pattern is very promising in this application domain, if the specific properties of embedded systems are considered in the pattern representation. In this paper, we proposed an extended pattern representation for the design of safety-critical embedded applications. This representation focuses on the implications and side effects of the represented design method on the non-functional requirements of the safety-critical embedded system including safety, reliability, modifiability, cost and execution time. Two example patterns have been used to show the effectiveness of the proposed pattern representation. We expect that this extended representation will guide

the selection of a suitable design as it allows evaluating alternative patterns with respect to their implications.

VII. FUTURE RESEARCH DIRECTIONS

For a successful application of the proposed representation of design patterns for safety-critical embedded systems, an integration of a higher number of design patterns is desirable. For this reason, we currently construct a pattern catalogue based on the proposed representation by collecting and classifying commonly used hardware and software design methods. Moreover, it is intended to construct the catalogue such that an automatic recommendation of suitable design methods for a given application can be achieved in the future.

References

[1]. Acharyulu, P. (2015). A Framework for Safety Assessment in Software Intensive Critical Systems. *Safety Science* , *77* (01), 113-120.

[2] C. Alexander, "A Pattern Language: Towns, Buildings, Construction," New York: Oxford University Press, 1977.

[3] E. Gama, R. Helm, R. Johnson, and J. Vlissides, "Design patterns: Element of reusable objectoriented software," New York: Addison-Wesley, 1997.

[4] IEC61508 Functional safety for electrical/ electronic / programmable electronic safetyrelated systems, International Electro technical Commission, 1998.

[5] A. Armoush, F. Salewski, and S. Kowalewski, "Effective pattern representation for safety critical embedded systems," International Conference on Computer Science and Software Engineering (CSSE 2008), pp. 91-97, 2008.

[6] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal "Pattern-oriented software architecture: A system of patterns," John Wiley & Sons, Inc., New York, NY, 1996.

[7] P. Coad, "Object-oriented patterns," Communications of the ACM, Vol. 35, pp. 152159, 1992.

[8] K. Beck and W. Cunningham, "Using pattern languages for object-oriented programs," Presented at the OOPSLA- 87 Workshop on Specification and Design for Object- Oriented Programming.

[9] J. Coplien, "Idioms and patterns as architectural literature," IEEE Software, Vol. 14, pp. 36-42, 1997.

[10] B. Appleton."Patterns and software: Essential concept and terminology," available at <http://www.enteract.com/~bradapp/docs/patternsintro.html>.

[11] B. P. Douglass, "Doing hard time: Developing real-time system with UML, objects, frameworks, and pattern," New York: Addison-Wesley, 1999. [12] B. P. Douglass, "Real-time design patterns," New York: Addison-Wesley, 2003. [13] D. Gross and E. Yu, "From non-functional requirements to design through patterns," Requirements Engineering, Vol. 6, No. 1, pp. 1836, 2002.

[14] J. Cleland-Huang and D. Schmelzer, "Dynamically tracing non-functional requirements through design pattern invariants," Workshop on Traceability in Emerging Forms of Software Engineering, in conjunction with IEEE International Conference on Automated Software Engineering, 2003.

[15] J. Fletcher and J. Cleland-Huang, "Softgoal traceability patterns," in Proceedings of the 17th IEEE International Symposium on Software Reliability Engineering (ISSRE 2006), pp. 363374, 2006.

[16] L. Xu, H. Ziv, T. A. Alspaugh, and D. J. Richardson, "An architectural pattern for nonfunctional dependability requirements," Journal of Systems and Software, Vol. 79, No. 10, pp. 13701378, 2006.

[17] S. Konrad and B. Cheng, "Requirements patterns for embedded systems," in Proceedings of the IEEE Joint International Requirements Engineering Conference (RE'02), pp. 127-136, 2002.