

Extreme Requirements (XR)

Julio Cesar Sampaio do Prado Leite

Pontifícia Universidade Católica do Rio de Janeiro — PUC-Rio
Rua Marquês de São Vicente 255, 22451-041 Gávea-RJ
Rio de Janeiro, Brasil
www.inf.puc-rio.br/~julio

Abstract: This paper builds upon the work of Kent Beck on Extreme Programming. Here we advance some ideas on how requirements engineering research can improve Extreme Programming and how the managerial common sense exposed by Beck can improve requirements management. The ideas expressed in the paper are anchored on Beck's book and on my long experience in Software Engineering. The main argument is that a family of processes, called **XR**, can provide more quality to Extreme Programming projects.

1. Introduction

In the summer of 2000 I had the opportunity of listening to Kent Beck's keynote speech at the International Conference on Software Reuse in Vienna. Just two months before the conference I had bought his book about extreme programming (XP), curious about the name and about the comments I was hearing, both from academia as well as from industry. I had mixed feelings about the talk. My first impression was that there was nothing new and that the proposal was essentially a management strategy. However, I was impressed with acceptance of the ideas and how well they were put together. I also understood it as a very rigid discipline, disguised as a process with not much discipline. Roles were well defined, working hours were defined, working environment was defined as well. So, it seemed to me strange that Beck had criticized Frederick Taylor, father of scientific management, for his rigidity.

Several months had gone by since I finishing reading the book [1]. After that, I did not change much of my first opinion, but I got to better value Beck's words. He managed to put together several very good ideas on software management and more importantly he managed to propose an overall process that is very attractive to "real programmers". He apparently cuts the fat of some well-established software engineering methods, focusing on the final document of the process, that is, source code. I say apparently, since his rules of engagement for the software process, just shift the documentation load to the code itself and to test cases. He does not address verification in an explicitly manner and addresses quality assurance by constant validation.

My main curiosity was how could XP work, if not much attention was given to requirements engineering? After all, years of academic experience as well as industry experience, just confirm what is common knowledge: “you must understand what you will model”. So, how does Extreme Programming deal with requirements definition?

On the other hand my research on software evolution [2] has convinced me that change is intrinsic to the task of software construction, so that we need better processes to address change. This was the other reason to get to know more about Extreme Programming since one of its claims to strength is its dealing with constant change.

My proposal of **XR** as a family of requirements processes is offered as a complement to the Extreme Programming book [1] for XP projects.

2. The Role of Requirements on XP

Nowhere in the XP process there is a mention of a requirements document, but the term “requirements” is used several times in the book. The use of the term requirements is usually in the context of non XP use, as exemplified by quotations from different parts of the book [1].

“The customers can’t tell us what they want. When we give them what they say they want, they don’t like it”. This is an absolute truth of software development. The requirements are never clear at first. Customers can never tell you exactly what they want.

Every project I’ve worked on that had fixed price and scope ended with both parties saying, “The requirements were ’nt clear” ...

In this scenario, Business always specifies too much. Some of the items on the list of requirements are absolutely essential. But some are not. And if Development doesn’t have any power, they can’t object; they can’t force Business to choose which is which.

...

It seems to be in the nature of the less important requirements that they entail the greatest risk. They are typically the poorest understood, so there is great risk that the requirements will change all during development. ...

You will have to convert your existing requirements information to story cards. ...

XP divides the world between Business and Development. Business is the client and Development is the organization that will deliver the product to the client. The following summarizes business responsibility.

- Choosing the scope or timing of releases.
- Choosing the relative priorities of proposed features.
- Choosing the exact scope of proposed features.

To carry out these responsibilities, the XP team has a role named *customer*. This role is to be played by a person who will be devoted full time to the project (*on-site customer* is one of the 12 XP practices).

Defining scope, releases and priorities is done by the customer. This task is mainly performed at the *planning game* practice¹. The goal of *the planning game* is to maximize the value of software produced by the team; the players are Business and Development. The cards of the game are called story cards. These cards are forms in which the customer states what is to be done, by enumerating the activities (functions) that the software should support. There are three moves in the game: exploration, commitment and steer. In the exploration phase a story is written, it is estimated and it could be divided. In the commitment phase, the scope and date of release are set for each. Steering is performed every three weeks based on iteration. Iteration works with sub-divisions of stories, called tasks, which are also written in cards. Iteration is based on estimation, division of labor between developers and on testing.

The customer writes stories and tasks, makes decisions, and writes functional tests. The customer must also answer questions from the developers, when the developers are engaged in the *listening* activity. The customer is central to XP and is involved directly in 3 of the 12 practices: *the planning game*, *testing*, and *on-site customer*.

As such, the requirements definition is performed continuously by means of stories, tests, new stories tests and so on. By doing this XP is a learning environment where the requirements are developed as the project goes along. The discipline of writing stories, writing tests and continuously scoping makes it possible that the *on-site customer* steer the development and as such achieve the desired result, which can be a moving target.

XP does not say much about documentation, other than having *coding standards* as one of the 12 practices. It is clear through the book that code is the ultimate document. However, there are two figures in the book, Figure 6 and 7, which show the story card and the task card. These cards, besides having place for natural language text, have pre-defined fields to identify the card, to prioritize it, to have estimates for it and to keep

¹The planning game is another of the twelve XP practices.

history. As such, I suppose that these cards, contrary to the CRC cards or other design annotations, are not thrown away (see page 112-113 of [1]). I assume that those cards are the requirements documents in the XP process. I also assume that the *tracker*, one of the XP roles, will be responsible for maintaining those records.

3. XR (Extreme Requirements) for XP projects

In this Section, I list problems I see in the XP process regarding requirements, and will give a receipt for dealing with them. I will, of course, try to maintain the extreme spirit and use the same principles, practices and values of XP.

Dealing with requirements the way XP does (Section 2) have several problems. I will deal with the five of them.

- 1. The assumption that, in the planning game, the business could be represented by just one customer.**
- 2. The lack of consideration of non-functional requirements from the standpoint of the business.**
- 3. The lack of explicit links between stories and tasks cards to the code**
- 4. The lack of a process for producing functional tests.**
- 5. The lack of a process for producing stories and tasks.**

Problem 1:

One of the major problems when dealing with Business is that very seldom can you find somebody who can be a real representative of several different interests or perceptions of the Business². The problem of dealing with different viewpoints of stakeholders is a well recognized problem [3]. It is not a simple problem and not an easy one to solve. By relying on a representative of the Business, *the on-site customer*, is a way of saying that it is the responsibility of the Business to solve the business's "internal" inconsistencies. Well, that may work, but it poses a high risk for the project.

² In XP terms Business represent the demand side, where Developers are the supply side. The practice of the on-site customer is to have one representative from the demand side (those which are the clients for the software) participating in the XP team for the whole project.

My proposal is to use a process and a representation for writing the stories and tasks cards. The representation is scenarios, and the process is an inspection process.

A scenario is a description technique that enhances understandability of task-related descriptions and communicability among stakeholders. Jarke, Bui and Carroll [4] point out that the effectiveness of the use of scenarios in several disciplines is fundamentally due to their capability of stimulating thinking. Scenario provides a situated task vision together with an effective way of communication among the actors involved in the subject of study.

We have been using scenarios for some time now [5] with very positive results [6]. I believe that they are a perfect addition to the XP story/task cards. Our scenario [5] description language is very easy to learn and easy to use. People of several different backgrounds having been using the language with success.

Figure 1 and 2 show examples from [6].

<p>TITLE: Organize the Meeting GOAL: Assure an efficient development of the <u>meeting</u>. CONTEXT: The <u>meeting</u> has been previously scheduled. RESOURCES: <u>equipment</u>, <u>physical space</u>. ACTORS: <u>requester</u>, <u>secretary</u>.</p> <p>EPISODES: The <u>requester</u> instructs the <u>secretary</u> about the <u>meeting call</u>. CALL TO THE <u>MEETING</u>. # NOTIFY ASSISTANCE. NOTIFY ABSENCE. [ASK FOR <u>EQUIPMENT</u>.] IF the convoking date was made with anticipation THEN REMIND THE <u>MEETING</u>. [The <u>secretary</u> assures that the <u>equipment</u> is available for the <u>meeting date</u>.]</p>
--

Figure 1

<p>TITLE: Cancel a meeting GOAL: Free the <u>agenda</u> from a <u>meeting</u> to be cancelled. CONTEXT: The <u>meeting</u> has already been scheduled. RESOURCES: <u>agenda</u>, <u>schedule of meetings</u>, <u>physical space</u>, <u>equipment</u>, <u>list of attendees</u>, communication media... ACTORS: <u>requester</u>, <u>secretary</u>, <u>attendees</u>.</p> <p>EPISODES: The <u>requester</u> or the <u>secretary</u> records the <u>cancellation of meeting</u> in the <u>agenda</u>. <i>Constraint:</i> it must be done previous to the <u>meeting date</u>. The <u>requester</u> or the <u>secretary</u> records the <u>cancellation of meeting</u> in the <u>schedule of meetings</u>. <i>Constraint:</i> it must be done previous to the <u>meeting date</u>. #IF <u>convocation</u> has been done THEN the <u>secretary</u> informs the <u>cancellation of the meeting</u> to every attendee by any communication media using the <u>list of attendees</u>. IF <u>convocation</u> has been done THEN the <u>secretary</u> records the <u>cancellation of meeting</u> in the <u>list of attendees</u>. The <u>secretary</u> cancels then reservation of the <u>physical space</u>. [The <u>secretary</u> cancels then reservation of the <u>equipment</u>.]#</p>
--

Figure 2

Of course, integrating our description language to the XP story/task cards requires some minor adaptations. One of them would be to focus the scenario not on the actual situation of the Universe of Discourse, but on the desired situation with the presence of the software.

Once the scenario or the set of scenarios is written by the *on-site customer*, it can be inspected by other representatives of the Business, not directly involved in the XP team, in order to assure that it is not a particular view of the Business. The inspection process is a simplification of the scenario inspection process [7] and is mostly conducted in a single meeting or a single reading cycle with other representatives of the Business. These representatives have to be carefully picked to adequately represent the Business.

In Figure 1, we can see that some episodes are in upper case. This is an indication that there are sub-scenarios to represent those episodes. We see this as a way of representing the story and the task and maintaining the link between them.

Problem 2:

The fact of postponing the dealing with non-functional aspects may lead to an increase in cost as well as to problems in the quality of the deliverables [8]. The answer to this problem is to try to bring non-functional aspects to light as early as possible [9]. Non-functional aspects may be general or specific. General non-functional aspects are cross domain and deal with things like: usability, performance and safety. Specific non-functional aspects are the constraints that should be attached to functional aspects.

In Figure 2, there are two episodes that have a special field, *Constraint*. This special feature of the scenario language makes it possible that non-functional aspects be stated at the same time as functional aspects.

Processes to deal with specific non-functional aspects are not difficult to teach. Customers writing stories can be asked to think about the restrictions that may be applied to parts of the story. The fact that the scenario language has a placeholder for this type of information certainly helps to avoid leaving those important restrictions out of the story description.

Problem 3:

The task of *tracking* is the responsibility of the *tracker*. Tracking is not an easy task. It implies that traceability support be available so relationships can be recorded. As Beck

says, the *tracker* is the historian of the XP project. In **XR**, there is a built-in aspect that deals exactly with the aspect of traceability to make it possible to link stories to code in a very simple manner.

The **XR** process that deals with tracking is the name spacing process. Name spacing is based on the language extended lexicon [10], a technique that maps the vocabulary of the Business. Name spacing aims to guarantee that the names in the lexicon are preserved in the stories and will be the same names used in the code. That is, the *coding standards* must have an explicit rule about naming the operators and operands with the names present in the lexicon.

The process of building scenarios [6] enforces the rule of naming according to a lexicon. See in Figure 3 an example of the lexicon. The symbol **requester** is present in the scenarios of Figures 1 and 2. In the scenarios, the reader should note that the word **requester** is underlined, meaning that an explicit link exists between the underlined word and the lexicon. This is the type of tracing that **XR** implements in XP projects.

The lexicon is a vocabulary that uses two definitions. One is the denotation (notion) of the symbol, what it is. The other is the connotation (behavioral aspect) of the symbol, an extra meaning of the symbol in the context at hand. An entry of the lexicon follows two principles: closure and minimal vocabulary. The minimal vocabulary says that the words used to describe an entry should be drawn from the minimal subset of most used words in the natural language of choice. The closure principle tries to maximize the use of symbols in the description of other symbols. These rules force the lexicon to be self-contained and highly connected hypertext. Using the lexicon as a link to scenarios and to code enables the application of a simple, but effective traceability policy.

Requester

Notion:

- person who invites attendees to a meeting.
- may be a participant.

Behavioral Response:

- defines the objective of the meeting, the subjects to be discussed, the attendees, the material to exhibit and the material to distribute.
- records the objective and the attendees in the agenda.
- performs the scheduling of meetings.
- organizes the meeting.
- records the substitute in the agenda.
- decides the cancellation of the meeting.
- decides changes in the meeting requirements.
- decides the moving of date.

Problem 4:

It is not easy to teach a customer how to write functional tests. On the other hand, it is easy to ask the customer to give examples of the scenarios that he or she wrote. So the functional tests are derived simply and directly from the scenarios.

Our goal is to find derived situations from the scenario that cause the system to fail. The user has to understand that he will write **cases** that could lead the system to fail. So, he has to understand that different variations of the given situation may exist and that he should try to enumerate those. He also has to understand that testing the limits of each situation provides a good indication of how the software will behave.

The above general heuristics are given to the customer for writing possible **examples**. The examples should be input data that reflect real situations that may happen for a given story/scenario.

First, we need examples for the definition part of the scenario (context, resources and actors). Each example tries to present a situation in which the software may fail. Thus, each constraint and its negation will have to be exercised.

Second, we create examples for each episode, depending on its constraints, its negation or the control structure it uses. Examples for each episode are tried after the definition part is tested.

I will use the scenario of Figure 2 as an example to illustrate how the process should work.

Below we show examples for the non-behavioral parts of the scenario.

- **EXAMPLE 1** Establish the context -- The meeting XYZ was defined on 06/22/99 for happening on the 07/2/99.
- **EXAMPLE 2** Negate the context – The meeting XYZ was not defined.
Expected output: error message
- **EXAMPLE 3** Establish the resources (information/files/fields) – Agenda TT does exist, Schedule of Meetings SST does exist, Physical space does exist, Equipment does exist, ...

- EXAMPLE 4 Negate the context -- Any of the resources fails to exist.
Expect output: error message

Assuming that the examples above were exercised, we can exemplify tests for the first episode in Figure 2.

- EXAMPLE 5 on 6/21/99, an actor (requester or secretary) records cancellation of meeting XYZ in the agenda. Expected output: Error message, meeting not defined
- EXAMPLE 6 on 7/2/99, an actor (requester or secretary) records cancellation of meeting XYZ in the agenda. Expected output: Error message, meeting can not be canceled on the same date
- EXAMPLE 7 on 6/23/99, an actor (requester or secretary) records cancellation of meeting XYZ in the agenda. Expected output: agenda has XYZ as canceled.

Of course that we simplified the presentation a little bit in order to exemplify the principle. For instance, for building the examples for the third episode of Figure 2 we need a list of attendees, that is the name of each attendee, provided as data. An example to test limits would be listing the name of the requester of the meeting as one of the names in the list of attendees, trying to check if a requester was also implemented as a attendee or not.

Some of these heuristics could be automated in a scenario editor environment. The test cases generated by the *on-site-customer* should be inspected as the original scenario was, in order to see if further tests are necessary. So the process used to build a shared viewpoint is also used to generate tests.

Problem 5:

The Extreme Programming book does not provide much detail of the processes. I believe that other XP publications will show processes in more detail. However, I do not believe they will detail the requirements-related problems we are addressing.

I and some of my colleagues [6] built a process for producing scenarios, in which both validation and verification were taken into consideration. The overall process can be seen in the SADT diagram below, taken from [6]. I believe that such a process can be customized for **XR**. The use of such process in the context of XP, that is, with scenarios being the description part of the story/task cards, will add the opportunity of describing

the requirements information in an organized way. Of course, imposing such process will add an overhead in terms of time in *the planning game*. However the activities we are adding are not complex and can be kept at almost the same level of detail as the original cards, thus maintaining the overall XP time frame.

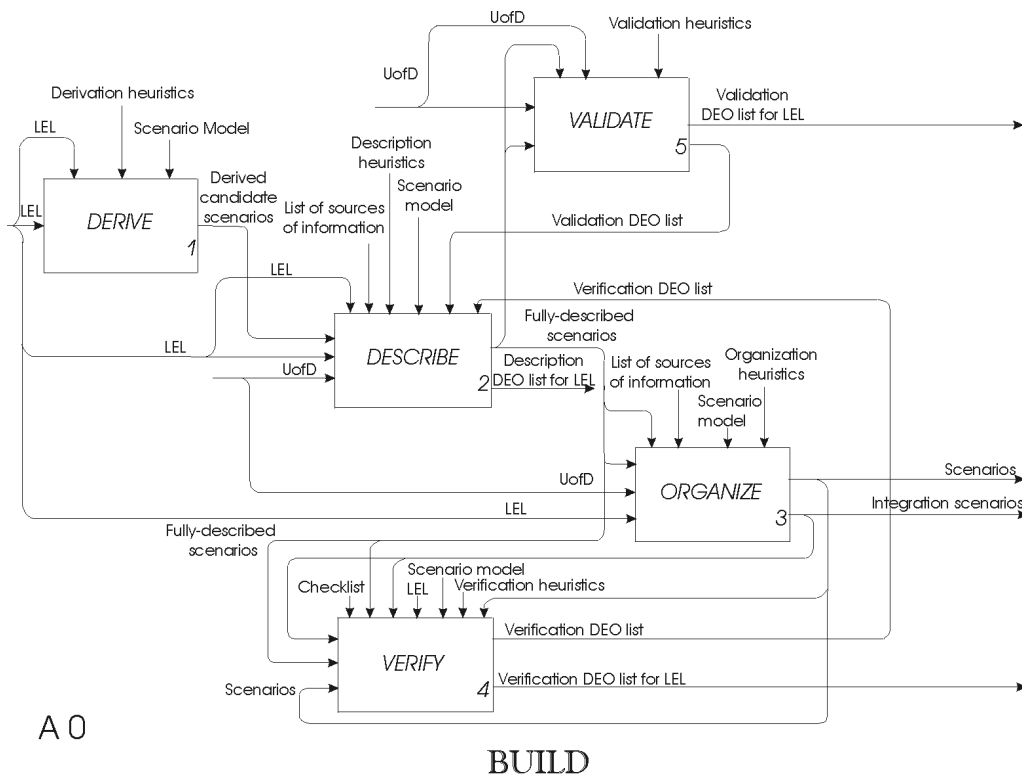


Figure 4 - SADT of the Scenario building process [6]

The scenario building of Figure 4 has 5 sub-processes. DERIVE is concerned with eliciting requirements and is performed by Business. DESCRIBE details the scenarios as well as the functional test cases (examples). ORGANIZE is used to maintain a relationship among the different cards (scenarios) being produced. VALIDATE and VERIFY are quality checkpoints for the process. VALIDATE is performed by Business through the process of sharing viewpoints or viewpoint resolution [11]. VERIFY is performed by the XP team during the activities of *testing* and *listening*. Note that the VALIDATE sub-process is not the XP practice *testing*. Here, we are just

making sure that the cards and the functional tests express the overall viewpoint of Business.

Looking closer at Figure 4, observe that having established the lexicon of Business is a pre-condition to the process proposed in [6]. I also believe that it should be as such in the **XR**, however we do not need a complete lexicon to start with, but just a list of the important symbols, since definitions will be added during the continual feedback of the XP process. Accordingly, the heuristics to derive the scenarios are not the same as used in [6]. In **XR**, the focus is on getting a very first cut of the cards, so that they can be a reasonable basis for producing the functional tests and guiding the XP design process.

4. Making XR work

I believe that the solutions proposed above are easy to be implemented and are not too expensive. We estimate that a 4-hour introductory course on scenarios is sufficient as a quick start. It is important to stress that the scenarios are being used as a replacement for the description part of the story/task card. So, there is not much hassle in preparing to using scenarios. However, it is recommended that a consultant be available to help tune the customer's expertise on using scenarios.

Functional testing should be introduced in a similar way. Again, a 4-hour introductory course, after the scenario course, is sufficient for the customer to gain initial speed on producing test cases. Also the help of a consultant to tune the process is recommended.

With respect to the VALIDATE process, there is a need for more investment. An introductory course on viewpoints should require 8 hours, and the customers should already be familiar with scenarios and test cases. Here the focus is on collaboration work and how to implement such a process on the Business side of the organization. Of course, the task is not trivial, since it will require commitment from the organization on **XR** process. However, having more confidence on what the story cards say is definitely a plus. Remember that problem 1, mentioned in Section 3 above can put the whole XP project at risk.

XR maintains the same actors and roles as in XP. The major difference is that **XR** is within the Business side by trying to achieve viewpoint resolution and validating the scenarios and test cases. In a way, we bring the spirit of XP towards the Business side, since viewpoint resolution is, of course, an extreme collaborative process.

Of course, those tools can help the process. Dealing with cards by hand is certainly a headache for the *tracker*. A configuration management system is certainly a definitive

requirement to help track the information. Also it would be very productive if we could have a suite of tools that could deal with the lexicon, scenarios and test cases in an integrated way and with links to the code itself. We believe that concentrating the information on code is a great idea, but it would be even better if the requirements information could be linked and managed as an integrated part of the code.

5. Conclusion

Our proposal integrates well with the XP philosophy. Requirements evolution [6] [2] has been a major concern in my research, and I see XP as a possible paradigm to transfer to practice our evolution ideas. As such, I proposed in this paper a first step in this direction, that is how to enhance the XP process with a requirements focus.

I discuss neither the applicability of XP nor its successes and failures. I analyze the requirements part and propose an add-on that I strongly believe has high cohesion with the XP proposal. In addition to seeing XP as an opportunity to spread the word on requirements evolution, my main concern, in this paper, is to show that **XR** is a plausible extension to XP.

Experience on the integration of **XR** is essential. I hope to have the opportunity to implement these ideas and gather data on its use. On the other hand, I see this proposal as also a way of exposing to object-oriented development community the role of requirements in a programming-centered paradigm.

Future work will take a different perspective, that is, how the ideas of XP could influence an **XR** process that is not integrated with XP. In that case, the focus will be on an **XR** process that is geared to other forms of project management.

Acknowledgments

I wish to thank Professor Daniel M. Berry for his comments on an earlier version. Support for this work has been provided by CNPq, Faperj (Cientista do Nosso Estado), MCT-Pronex and by the CYTED project WEST.

References

- [1] Beck, K., *Extreme Programming Explained Embrace Change*, Addison Wesley Longman, Inc., (2000).

Keynote at the *Jornadas de Ingeniería de Requisitos Aplicadas*, Sevilha, June, 2001

- [2] Leite, J.C.S.P., Scenario Evolution. *Dagstuhl-Seminar-Report; 199, Schloss Dagstuhl, Internationales Begegnungs-und Forschungszentrum Fur Informatik*, Bui, Carrol and Jarke (editors), Alemanha, pp.13-14, (1998).
- [3] Vidal, L., Finkelstein, A., Spanoudakis, G., Wolf, A.L. *Joint Proceedings of the SIGSOFT '96 Workshops, International Workshop on Multiple Perspectives in Software Development (Viewpoints '96)*, The Association for Computing Machinery, (1996).
- [4] Jarke, M. Bui, T.X., Carrol, J.M., Scenario Management; An Interdisciplinary Approach, *Requirements Engineering Journal*, Vol. 3, N.4, pp. 155-173, (1998).
- [5] Leite, J.C.S.P, Rossi, G., Maiorana, V., Balaguer, F., Kaplan, G., Hadad, G., Oliverson, A. Enhancing a Requirements Baseline with Scenarios, Proceedings of the *Third International Symposium on Requirements Engineering, IEEE Computer Society Press*, pp. 44-53 (1997).
- [6] Leite, J.C.S.P., Hadad, G., Doorn, J., Kaplan, G, A Scenario Construction Process, *Requirements Engineering Journal*, Vol. 5, N.1, pp. 38-61, (2000).
- [7] Doorn, J., Kaplan, G., Hadad, G., Leite, J.C.S.P. Inspección de Escenarios, WER 98, *Workshop de Engenharia de Requisitos, Departamento de Informática PUC-Rio*, pp. 58-69, (1998) (www.inf.puc-rio.br/~wer98)
- [8] Chung, L., Nixon, B., Yu, E., Mylopoulos, J. "Non-Functional Requirements in Software Engineering" *Kluwer Publishing*, (2000).
- [9] Cysneiros, L.M., Leite, J.C.S.P., Integrating Non-Functional Requirements into Data Modeling, *Proceedings of the Fourth IEEE Symposium on Requirements Engineering*, IEEE Computer Society Press, pp. 162 – 171, (1999).
- [10] Leite, J.C.S.P., Anchoring the Requirements Process on Vocabulary, Requirements Capture, Documentation and Validation, *Dagstuhl Seminar Report – 241*, pp. 13-14, (1999). (www.dagstuhl.de/DATA/Reports/99241)
- [11] Leite, J.C.S.P, Freeman, P. A. "Requirements Validation Through Viewpoint Resolution" *IEEE Transactions on Software Engineering: Vol. 17, N. 1*, pp: 1253 -- 1269, (1991).