Electronic version of an article published in **Eberlein, A. (Hrsg): International Workshop on Time Constrained Requirements Engineering, (TCRE '02). Proceedings, Rio de Janeiro; Papel Virtual; pp. 57-64**

Requirement Documents that Win the Race: Not Overweight or Emaciated but Powerful and in Shape

Kirstin Kohler, Barbara Paech
Fraunhofer IESE
Sauerwiesen 6, 67657 Kaiserslautern, Germany
kohler, paech@iese.fhg.de

Abstract

Time-constrained projects ask for requirements approaches that are agile, i.e. adapted to the project needs and without comprehensive documentation. But how can this be achieved? Our approach provides the steps toward the solution of this question. It supports the identification of the essential content of the requirements document as well as the selection of the appropriate modeling technique. The essential content is determined by conducting a systematic risk analysis, which allows identifying the most important elements of the requirements documentation. For the requirement document to be useful it must be precise and understandable for all project participants. The appropriate modeling technique is selected by taking the identified content and the context of the project into account. This paper reports work in progress. It describes the motivation, related work and first ideas.

1. Introduction

In the meantime it is well known and widely accepted, that a lot of problems in software development are caused by deficiencies in the requirements phase [20]. Never the less a lot of companies lack requirements engineering activities [17]. Especially in projects where „time-to-market“ is critical for the success, there is usually „not enough time“ for investments in the RE-process [16]. In order to overcome this resistance against RE in industrial settings, it is not necessary to invent a new and hopefully better RE method. Instead one should focus on making existing methods more attractive [22]. The goal of our work is to make *documentation of requirements* more attractive for time-to-market projects.

Creating and maintaining requirements documents requires substantial effort [19]. This is why it is often neglected, especially if schedule is tight. Whereas XP puts the “onsite customer” into a project and thereby declines any kind of documentation [5], our experience in industrial project shows that documentation of requirements is crucial for the transfer of knowledge between stakeholders of the actual project as well as to subsequent development projects. Thus, it cannot be completely disregarded. With our approach, we want to balance the diverse goals of creating good documentation and keeping a tight schedule. Our slogan is: “keep the documentation as small as possible but as substantial and useful as needed for the project”. Our approach gives advice how to put this slogan into practice. It supports the following two steps: (1) identify the *essential* part of the documentation and (2) choose the *appropriate* modeling technique to document them. We consider the project context[1] in both of these steps, especially in the definition of “*essential*” and “*appropriate*”. With our work, we do not invent a new RE-method but provide guidance on how to use existing methods efficiently. So far we limited the framework according to two dimensions: (1) it supports only documentation of requirements and (2) it considers only GUI-intensive systems.

The paper shows “work in progress”. Section 2 explains the two basic steps of our approach. By explaining them in more detail in section 3 and 4 we refer to related research work and show how we extend or plan to extend it. Finally, section 5 summarizes our approach and elaborates on future work.

2. Basic steps of the approach

In order to document requirements for time to market projects it is important to make a trade off between quality and schedule. Gaps or faults in the requirements phase as a consequence of missing documentation lead to dissatisfied customers due to quality problems in the end product. In contrast, “high quality” software development including comprehensive documentation takes longer and therefore customers choose competitors products. The product is of less value due to an unsatisfying market penetration. Narrowing these conflicting dependencies to the scope of requirements documentation means making the tradeoff between no documentation leading to bad quality and complete documentation leading to delayed product delivery. The solution is to keep the documentation as *small* as

[1] With context we mean important factors influencing the software project. They are listed in more detail in section 4.

possible but as *substantial* and *useful* as needed for the project. We accomplish this by supporting the two steps "Identify essential content" (ensures *small* and *substantial)* and "Choose appropriate modeling technique" (ensures *useful)*. Figure 1 shows the two basic steps of our framework in relationship to the basic ingredients: the project risk, conceptual information model, the project context and the modeling technique. How these ingredients are integrated in our approach is described in the following subsections, where the steps are elaborated in more detail:

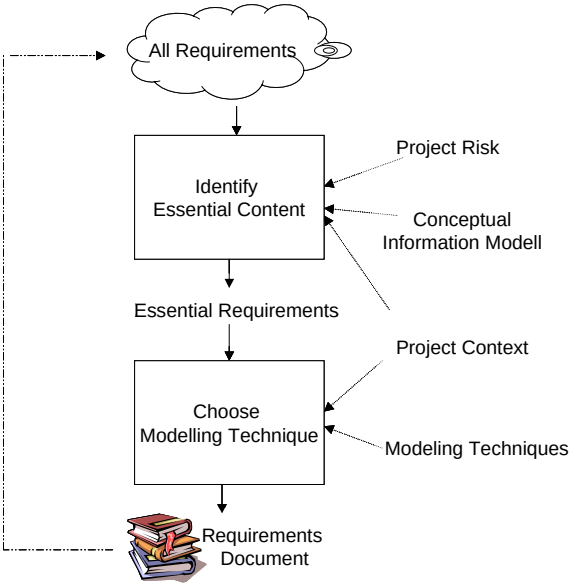

Figure 1: Steps and concepts of the approach

(1) Identify essential content: The basic idea is not to document the complete requirements, but instead concentrate on the most important or essential parts. Finding and separating these essential parts poses a classical filtering problem: Separating the requirements that are "worth" being documented from those that are uncritical and therefore need not be documented. In general a filter separates material or waves according to specific characteristics like grain size or frequency. The materials in our case are the requirements and we use a conceptual information model to classify the requirements. The filter in our approach separates requirements according to risk. It separates those requirements that are accompanied with a high risk from those that are accompanied with a low risk. The size of the risk (comparable to the size of grains passing the filter) is influenced by the project context. We provide an approach for how to assess the risk by considering the risk of different types of requirements in section 3. The next step is to document the requirements and therefore one has to choose a representation technique.

(2) **Choose the appropriate modeling technique**: Requirement documents are a medium of communication and knowledge transfer. In order to gain the most benefit out of them, they have to be precise and understandable for project participants and involved stakeholders (besides other qualities like correctness, consistency and so on). Thus, choosing the appropriate modeling technique is essential. The modeling technique has to fit to the content that is documented as well as to the people, which read the document. For example the navigation of dialogues can be documented by using Constantine's abstract prototypes [9], whereas the interaction between system and users in terms of function calls might be documented by use cases [7]. Depending on the project members, who might include e.g. graphic designers, specification languages like UML might not be suitable. This means when deciding about the appropriate modeling language the content as well as the project context have to be considered in order to make the documentation valuable. Section 4 will elaborate in more detail how the project context guides the selection of the modeling technique.

The dashed line in figure 1 indicates that our approach is not limited to classical waterfall projects, where all requirements are known at the beginning. It should be applied in iterative projects. It can be applied independent of the process model of the project and fits at that point where one has roughly understood the requirements (or a subset of them) and has to document them in more detail. Our approach helps to decide which part of these requirements to document and how to document them.

3. Identify the essential documentation content

We elaborate on the step of identifying the essential documentation content by explaining the three concepts we built on: the conceptual information model, the project risk and the project context.

Before filtering the essential requirements, we have to make explicit what we are filtering. What is the totality of requirements we are choosing from? To improve the understanding of a complex subject, in physics or chemistry scientists usually introduce models to represent these subject. We needed similar a model that allows us to think and argue about requirements especially in the context of the filtering process. But unfortunately there is not a standard model for GUI intensive applications (at least we are not aware of one after an extensive literature search), comparable to the Parnas' model for embedded systems [19]. Thus, we developed a conceptual information model for GUI intensive applications.

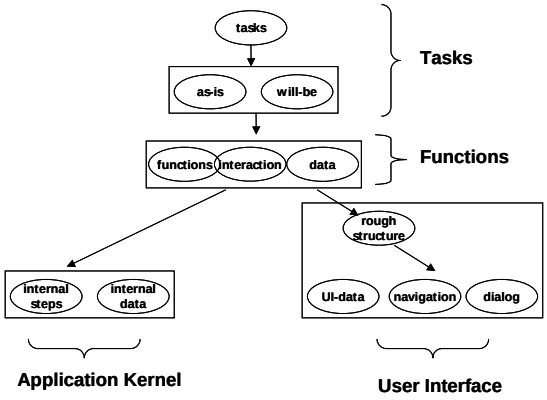

Figure 2: Conceptual information model consisting of 12 types of design decisions

(1) The **conceptual information model** describes the various elements and abstraction levels for requirements of interactive applications (excluding non-functional requirements). The model is based on Kovitz' [13] understanding that requirements activities lead to design decisions. During the requirements phase decisions about the effect generated by the software to be developed are made. For interactive applications we identified 12 types of these "design decisions" (see figure 2).

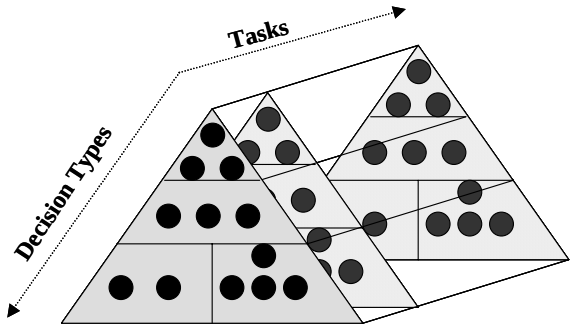

Figure 3: Set of task trees. Dots indicate the design decisions of figure 2

They can be categorized into 4 groups (decisions about tasks to be supported by the software, decisions about functions implemented by the software, decisions about the application kernel and decisions about the user interface). Each of these groups is on a different abstraction level. For the sake of brevity we will not go in the details of this model but refer the reader to Kohler and Paech [11]. The elements of the model allow us to argue on a conceptual level about different types of requirements. Therefore one can explicitly name, which type of the requirements should or need not be documented. As a simple example, if a new interface for an existing legacy system has to be implemented, it is important to specify the dialogs and navigation of the user interface, whereas the documentation of the application kernel requirements is less important.

The 12 identified elements of this model build a tree in the sense that the elements of lower levels depend on the decisions made before. The decision to support "the task of book ordering" leads to specific dialogs and functions like a "dialog to confirm selection" and "a function to calculate invoice". The tasks span a tree of dependent decisions. The set of all "tasks" to be supported builds a set of trees as illustrated in figure 3. Each triangle in the picture represents one "task tree". This does not mean that the decision to document a lower level requirement implies that the corresponding higher-level requirement must be documented. It must only be clear which tasks a specific functional requirement belongs to.
It is so far an open question of how to identify the essential tasks, which should be documented (at any level) at all.
Now we come back to the filtering process. Similar to testing of time-constrained projects [3], we use the concept of risk analysis to guide the selection of the essential requirements.

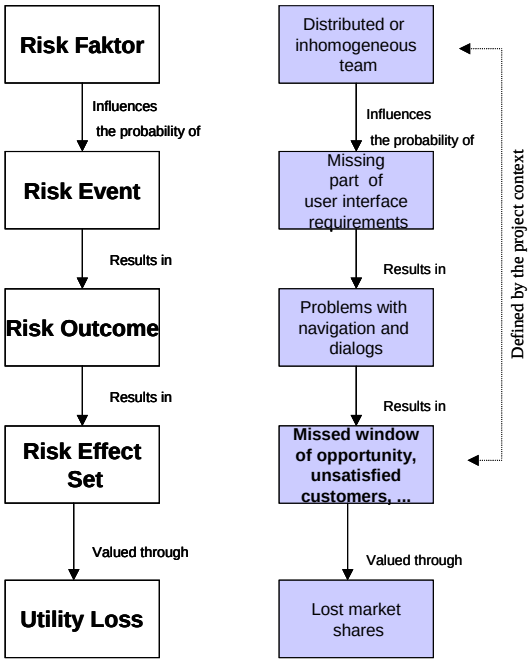

Figure 4: Kontios' risk effect chain instantiated for the requirements documentation

(2) **Risk** is the possibility of suffering loss [10]. This means that to asses the risk we have to know the probability and the loss linked with the risk. We base our risk analysis on a method introduced by Kontio [12] who defined an effect chain consisting of risk factor, risk event, risk outcome, risk effect and utility loss (see

figure 4, left hand). Without going into the details of Kontios' method we explain the elements in the context of the requirements documentation (see figure 4, right hand):

The risk event in our example is the missing documentation of requirements concerning the user interface (the design decisions we explicitly decide not to document). Missing documentation of user interface requirements can cause misunderstandings, which lead to wrong, superfluous or missing navigation and dialogs in the software (risk outcome). To judge the extent of the risk event and outcome one has to investigate which risk factors make the event likely (e.g. for a distributed team the probability[2] to cause a misunderstanding due to missing documentation is high) and which risk effects cause a high damage (e.g. a late product delivery due to superfluous functionality might result in a loss of market share due to a missed window of opportunity). The project context is essential for both of these questions. The project context defines the risk factors and therefore the probability that the risk event happens. And the project goals that are also defined through the context define the risk effects caused by the risk (the utility loss). Therefore the risk analysis can be reduced to the questions:

- What factors of the project context increase the probability of a misunderstanding?
- What project goals increase the utility loss caused by a misunderstanding?

The reflections about risk can be transferred to our conceptual information model. For each of the 12 elements of the model one has to consider which context factors enlarge the risk that a missing requirement of this type leads to a misunderstanding. But before this can be done, we have to define how to describe the context. How can one be sure to consider all relevant factors?

(3) **Project context**: The problem of context description gained especially importance within the last years in the domain of knowledge engineering and experience factory for software projects [4]. But due to the fact that this is a young research community and the relevant factors pretty much depend on the usage of the packaged experience, there is no silver bullet of context description. We used a scheme proposed by Birk [6] and adapted it to our needs in requirement engineering. Table 1 contains all factors to consider for a project characterization.

With the information model, the risk analysis and the attributes of a project context description we have now all tools at hand to support filtering of essential requirements. A risk analysis of a given project can be conducted by combining the elements of our information model with the various attributes of the project context. For each project attribute one has to determine the effect and the probability of a misunderstanding caused by missing documentation.

Attribute	Example
Stakeholders	
➢ Number	Number of people in the development team
➢ Experience	Experience in OO technologies
➢ Roles	Requirements engineers, user interface developer, graphic designer, …
➢ Customer	Development for customer X
➢ Suppliers	Usage of COTS products
➢ Distribution of stakeholder	Requirements engineers and developers at the same location or distributed development
➢ Availability of stakeholders	People that developed the former product version left the company, user access
Product	
➢ Application Domain	Web system, embedded system
➢ Lifecycle & History	Reimplementation of an existing product, maintenance
➢ Type of Product	Consumer Product
➢ System Size	20 components, 100 TLOC
➢ Lifetime of the Product	The product has to be supported until end of 2010
➢ Architecture	Client/Server architecture
Goals	
➢ Product Quality Goal	Reliability is more important than usability
➢ Business Goals	Time-to-market is more important than quality
Process/Technology	
➢ Development Process	Iterative development, RUP
➢ Techniques	Onsite customer interviews for elicitation of requirements, Prototyping of User Interfaces
➢ Tools	DOORS to manage requirements
➢ Standards	Documentation of requirements according to standard IEEE 1233-1998
➢ Weighting of activities	30% of the total development effort are spent for the requirements phase
➢ Duration of the project	The development of the project will be finished within one year
➢ Workproducts	Test-cases have to be documented

Table 1: Attributes for the context description

[2] Note that for now we do not propose to denote probabilities by number

But our approach supports practitioners not only by guiding them to the risk analysis. In addition we provide a set of heuristics. By conducting the risk analysis on a generic level (without considering a specific project context) and limited to the four main groups of elements in our information model (tasks, system functions, application kernel, user interface) we identified generic heuristics for the selection of documentation content. They are listed in table 2.

Risk event	**Risk factor that enlarge the probability**	**Risk effect that enlarges the damage**
Missing documentation of **tasks**	**- Lifecycle and History**: Big changes from "as is" to "will-be" tasks **- Type of Product**: Consumer product with a large variety of different users, complex user tasks to be supported by the software	**- Business Goals**: Tasks to be supported by the software are critical for the business success
Missing documentation of **functions**	**- Suppliers**: Parts of the system have to be built by COTs products. Evaluation of COTs product is based on system functions	**- Business Goal**: Time-to-market is important and forces to buy COTs products to hit window of opportunity
Missing documentation of **application kernel**	**- Experience**: developers are not experienced in algorithms	**- Product quality goal**: The accuracy of the product has to be improved to increase market shares.
Missing documentation of **user interface**	**- Lifecycle and history**: No experience of users in usage due to new development	**- Product quality goal:** Usability of the product is required due to mass production with high support costs otherwise **- Business goal**: Usability is important to reduce costs for training and support

Table 2: Heuristics for the risk effect chain

The left hand column lists the risk event. We listed one risk event for each decision type: tasks, function, application kernel, user interface. The second column contains context attributes (risk factors) that enlarge the probability of the risk event. E.g. for a big change from "as-is" to "will-be" tasks (risk factor) the probability that a missing documentation of tasks causes damage is high. The right column contains risk effects and their related product and business goals (given by the project context). They define the size of the damage caused by the risk. E.g. if time is critical for the business success of the company (risk effect) damage caused by the risk event "missing documentation of tasks" is high. If for a given project, more than one attribute in a row matches, the risk of omitting this type of requirements in the documentation is high. If only one, either the risk factor or the risk effect, is high, it has to be judged individually. In that case the table gives a hint for a potential risk.

Our tables help in executing the risk analysis, but, of course, this risk analysis still requires some extra effort. However, the identification of essential requirements is also necessary for the project managers to focus development efforts. We strongly believe that an explicit risk analysis is the best compromise between "all or nothing".

4. Choose the appropriate modeling technique

After having decided what to document it's now the question of how to document. There is a large variety of modeling techniques ranging from natural language to formal languages like Z [18]. One has to choose the modeling technique, which fits best for representing the requirements. It has to fit to the content that is documented as well as to the people that read the document.

During the last years a variety of methodologies have emerged that aim to guide the selection of technologies or methods:

- ACRE [15] is a framework containing 12 methods for requirements acquisition, which are judged by the authors according to their suitability in different projects. But the framework is limited to the 12 methods and does not cover modeling techniques for documentation. Furthermore there is a limited number of project characteristics covered by the framework, that do for example not consider the needs of specific applications like interactive or embedded systems
- As part of the PROFES project [6] the concept of PPDs (Product-Process-Dependency-Models) was developed. PPDs describe the impact of a specific technology, like inspections, on a specific product quality goal, like reliability, when applied in a certain process. PPDs also contain a description of the context. Although this approach seems to be very promising, it is very generic because it is

suitable for all kind of software engineering techniques and not especially tailored for our purpose of requirements documentation.

- In Agile Software Development Cockburn brought up the concept of the Crystal Family [8]. He proposes to choose the appropriate development approach dependent on the three characteristics: number of people, criticality of the software and project priority. We believe that this approach does not consider enough project characteristics.

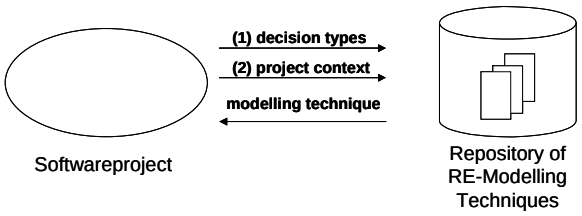

Figure 5: A repository of RE-techniques for the selection of the appropriate modeling technique.

We propose a two-folded approach for the selection of the appropriate modeling language. As illustrated in figure 5, we assume the existence of a repository containing a characterization and description of all available RE modeling techniques. By specifying the essential content in terms of its design decision types and by specifying the project context the appropriate modeling technique is determined. We will elaborate on this by referring to the underlying concepts: the decision types and the project context description:

(1) By defining the essential content, one already selected a subset of **decision types** as defined in the conceptual information model. The type of decisions determines a subset of suitable modeling techniques, because not every modeling technique is suitable to describe all types of design decisions. Kohler and Paech [11] provide a tabular overview of models used in common processes, like RUP [2] or Usage Centered Design [9], to document design decision types. Typically, after this selection process there is still more than one modeling technique left to choose from. Figure 6 illustrates this with an example. Based on the decision type "user interface", three techniques use-cases, storyboard and abstract prototypes are selected by function S1, which represents the selection based on the decision type.

(2) To further narrow down the appropriate modeling technique the **project context** is considered. Therefore the output of function S1 together with the project context are input for a second selection function S2. In S2 for each of the specified techniques the project context attributes are compared to the attributes of the technique description. The technique with the best match is the result of this final selection mechanism. In the example of figure 6, the project is characterized by a distributed team consisting of software engineers. Therefore use cases are selected as appropriate modeling technique. Storyboards require special drawing skills and are more difficult to exchange and discuss between different development sites.

Especially the context attributes describing the stakeholders and the process and technology (compare table 1) support this part of the selection process. Most important are the stakeholders. Their experience with special modeling technique and their roles (testers, graphic designers, …) have a very high impact on the acceptance of the modeling technique by the stakeholders. This is supported by empirical findings from McPhee and Eberlein [14] who showed that the usefulness of a RE techniques is correlated to the familiarity of stakeholders with that technique. In addition other "non RE" tools and techniques that are already established in the process may influence the selection process. E.g. if UML class diagram are in use to document the systems design, one should also use them to document the input/output data of the user interface.

So far our approach of choosing the appropriate modeling technique consists of very first ideas. Especially the relationship between context attributes and the selection of the modeling technique needs further investigation.

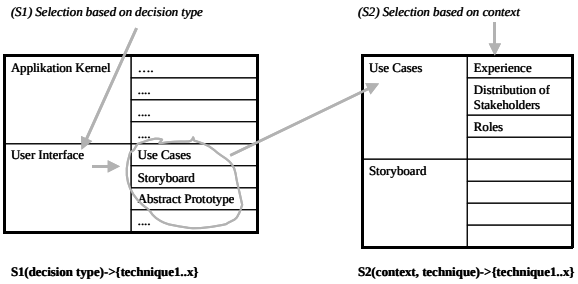

Figure 6: Selection of the modeling technique based on decision type and context.

5. Summary and Future Work

Our approach fits the recommendations for documentation of agile software development given by Cockburn [8] who states: "… don't ask for requirements to be perfect …" and "… capture just enough". But whereas Cockburn does not give any advice for how to find out what is "enough", we give concrete guidance on the selection process of critical elements. In addition it substantiates the demand for adaptiveness as postulated by the agile community [1]. In that sense we make the "agile requirements process" more concrete. Therefore we built our approach on two

sockets: the systematic consideration of risk and project context to drive the documentation in time-constraint projects.

So far we provided the skeleton which is now ready to be filled with more details to guide practitioners in their RE process. Our future work will concentrate on the following topics:

- We want to validate the context attributes through expert interviews, similarly to Vegas [21] who validated a characterization schema for testing techniques By doing this we will further adapt our characterization scheme for the needs of RE.
- We want to provide more heuristics similar to those listed in table 2. They should not only support the risk analysis for the selection of decision types, but also for the selection of a subset of task trees as illustrated in figure 3 to determine which tasks must be documented and which can be omitted.
- During future projects, by collecting empirical data we want to investigate the dependency between project context characterization and the selection of the content and the modeling language. We want to better understand the relationship between context attributes and the choices that have to be made based on these attributes. Knowing these dependencies would be a first step to automatically support the selection process.
- And of course we want to evaluate the complete approach to empirically prove, that our approach is beneficial for the requirements phase of time-constrained projects.

Literature

[1] www.agilealliance.org

[2] Armour, F., Miller, G., *Advanced Use Case Modelling*, Addison Wesley, 2000

[3] Bach, J., Heuristic Risk-Based Testing, *Software Testing and Quality Engineering Magazine*, 11, 1999

[4] Basili, V., Caldiera, G., Rombach, H., Experience Factory. In John J. Marciniak, *Encyclopaedia of Software Engineering*, volume 1, pp. 469-476, John Wiley & Sons, 1994

[5] Beck, K. *Extreme Programming Explained: Embrace Change*, Addison Wesley, Boston, 2000

[6] Birk, A, PhD Theses in Experimental Software Engineering, Vol. 3, *A Knowledge Management Infrastructure for Systematic Improvements in Software Engineering*, Fraunhofer IRB Verlag, 2001

[7] Cockburn, A., *Writing Effective Use Cases*, Addison Wesley, 2001

[8] Cockburn, A., *Agile Software Development*, Addison Wesley, 2002

[9] Constantine, L., Lockwood, L., *Software For Use*, Addison Wesley, 1999

[10] Dorofee, A. J., Walker, J., Alberts, Ch., Williams, R. et al., *Continuous Risk Management Guidebook*, Software Engineering Institute, Pittsburgh, August 1996

[11] Kohler, K., Paech, B. Anforderungsspezifikation für interaktive Anwendungen, *IESE-Report No. 016.02/D*, 2002

[12] Kontio, J., The Riskit Method for Software Risk Management, version 1.00, CS-TR-3782. *Computer Science Technical Reports*. University of Maryland, College Park, MD, 1997

[13] Kovitz, B.L., *Practical Software Requirements. A Manual of Content and Style*, Greenwich: Manning Publications Co., 1998

[14] McPhee, Ch., Eberlein, A., Requirements Engineering for Time-to-Market Projects, Proceedings of the 9th *Conference and Workshop on the Engineering of Computer-based Systems*, ECBS, Sweden, 2002

[15] Maiden, N., Rugg, G., ACRE: Selecting Methods For Requirements Acquisition, *Software Engineering Journal*, May, 1996

[16] Mead, N. R., Why Is It so Difficult to Introduce Requirements Engineering Research Results into Mainstream Requirements Engineering Practice?, *Proceedings of the Fourth International Conference on Requirements Engineering*, Illinois, June 2000

[17] Morris, P., Masera, M., Wilikens, M., Requirements Engineering and Industrial Uptake, *Requirements Engineering*, 3, 1998

[18] Potter, B., Sinclair, J., *Formal Specification and Z*, Prentice Hall, London, 1996

[19] van Schouwen A.J., Parnas, D. L., Madey J. Documentation of Requirements for Computer Systems, *Proceedings of IEEE International Symposium On Requirements Engineering*, San Diego, California, 1993, S.198-207

[20] van Lamsweerde, A., Requirements Engineering in the Year 00: A Research Perspective, *Proceedings of the 22nd International Conference on Software Engineering*, Ireland, June 2000

[21] Vegas, S., PhD Thesis, *Characterization Schema for Selecting Software Testing Techniques*, Universidad Prolitecnica de Madrid, February 2002

[22] Wiegers, K.E: Read My Lips: No New Models! *IEEE Software*, September/October 1998