Towards Extreme Requirements?

Ivan Maffezzini
Université du Québec à Montréal
CP 8888 Succ. Centre Ville
Montréal H3C 3P8 Canada
Maffezzini.Ivan@UQAM.ca

ABSTRACT

This paper presents an "Extreme Requirements" (ER) methodology for requirements engineering. After presenting two Unified Modelling Language models representing an IEEE definition of "requirement", a classical requirement process is outlined. The importance of the requirements artefacts is underlined and a composite specification, prototype oriented, is presented. In the second section, SCALCID, a project based on ER, is outlined. After a description of the relationship between forms (a textual structured representation of requirements) and prototype (dynamic description of functional requirements) the lessons learned and some questions for future works are presented.

Keywords: IEC 61850 - Methodology - Processes – Prototyping - Specification - Requirement

THEORETICAL FRAMEWORK

Introduction

In software engineering "Iterative and incremental development" (IID) has become a buzz expression characterizing "good" methodology. IID is deemed even better if it has UML (Unified Modeling Language) as a companion. At the moment the Unified Process is certainly the most famous IID process (or methodology). We claim that such a thing as a system or software engineering methodology appropriate to all kinds of projects or domains or companies cannot exist. But we think also that a methodology can be better than another in a certain domain or for a certain kind of project [1].

This paper presents a methodology for requirements engineering. We call this methodology "Extreme Requirements" (ER) by analogy with "Extreme Programming". In the same way that Extreme Programming emphasizes flexibility and dialogue in software construction, ER emphasizes flexibility and dialogue in requirements engineering. But whilst in Extreme Programming specifications are the "code servants", in ER code and specification are at the same level and constructed with the same method based on flexibility and adaptability.

To the extent that "The more specialized the context, the most efficient is the engineering method", ER leads to the following optimal context of use:

1. Big projects;
2. User's needs difficult to elicit and validate;
3. At least two categories of users with different skills, knowledge and points of view;
4. Requirement for the human-machine interface defined elsewhere;
5. A product that can bring about significant changes in the company culture.

The ER methodology is more akin to the Jackson methodology [2] than to a use case centered method like the Unified Process: that is to say the interaction between users and systems is an element of system design and not a requirement element.

What's "Requirement"?

In any engineering field, no matter what methods, approaches, tools, goals are, the requirement process can't be underestimated. Even if the two most known definitions of "requirement" are far from being universally accepted [2], we will take them as the definitions for this paper. The first definition is user-oriented and the second one is system-oriented.

1. "A condition or capability needed by a user to solve a problem or achieve an objective." [3]
2. "A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed documents." [3]

The two definitions seem very straightforward and doubtless the second one is such. But the user-oriented definition is far from being simple. It hides a huge problem: the link between "condition" and "capability" on the one hand and "problem" and "objective" on the other. To clarify this definition in Figure 1 we model the user-oriented definition for a system with more then one requirement. The concepts "problem" and "condition" are absent because, for our aims, they are subsumed by "Objective" and "Capability" respectively.

.

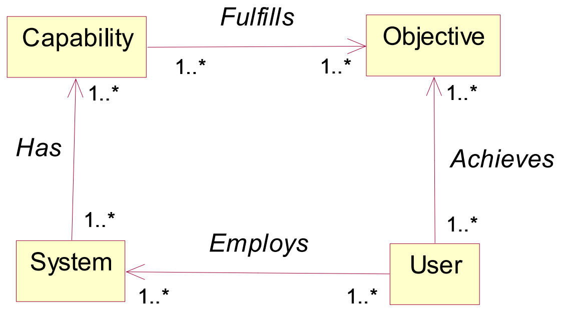

Figure 1: The requirement cycle

The requirements process is complex not only because the objectives and capabilities are difficult to define and to structure, but also because of the "one or more" (1..*) kind of multiplicity of all association's ends. For example: the fact that a capability can participate to the fulfilling of more than one objective and that one objective can be fulfilled by more than one capability can make very awkward the activities of assuring that a requirement is completely specified and that there is no inconsistencies.

Although the reduction of the multiplicity can alter the problem and even destroy its "core", we propose a simplification that facilitates the comprehension without destroying the problem at stake (what's requirements) [1].

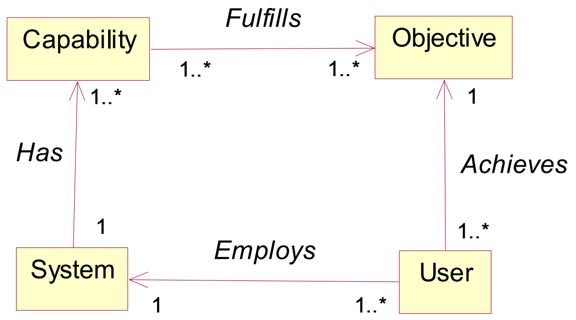

Figure 2: The requirement cycle: simplified view

Here is the list of the simplifications and the rational behind it:

- *A user has only one objective.* Even if this is seldom true, because the model retains more than one user, there will be still more than one objective. So the complexity due to the existence of several objectives is retained.
- *A user employs only one system.* If a user employs more than one system to achieve an objective we suppose that: 1) it is the user which goes from one system to the other or 2) an interface exist between the systems. In both cases is above all the system/software designer that is concerned and not the user or the requirements engineer[2].
- *A capability is instantiated in only one system.* System/software designer concern.

What cannot be simplified without loosing the problem's essence is the association between "Capability" and "Objective". If an objective can be achieved by more than one capability, the user (or someone else on his behalf) must choose, specify, validate these capabilities with regard to the objective.

Saying "specify the capabilities" is a way to enter in the second IEEE definition, the system-centered one. This is not quite by chance: the two definitions are profoundly entangled because "capability" is in one hand the "abstract thing" that fulfills the user objectives and in the other is what a concrete system must possess to operationally help the user to achieve her objectives. This entanglement is possibly the most important cause of so many system development errors [2].

Requirement process

Irrespective of the methodology, there is an agreement virtually unanimous about the activities of requirement process [4]:

- *Elicitation*: "It is the first stage in building an understanding of the problem the software is required to solve".
- *Analysis*: "Conceptual modeling [and] the classification of requirements to help inform trade-offs between requirements (requirements classification) and the process of establishing these trades-offs (requirements negotiation).
- *Specification*: "Production of a document, or its electronic equivalent, which can be systematically reviewed, evaluated, and approved."
- *Validation*: "Requirements validation is concerned with the process of examining the requirements document to ensure that it defines the right software (that is, the software that the users expect)."

This organization implies a temporal ordering that "the first stage" of the definition of Elicitation underlines. It is true that it is logically impossible to validate something that is not made explicit (specification) and it is impossible to specify without knowing the needs (elicitation). But this division, very useful pedagogically, has the same drawbacks than the classical software life cycle that practically all software engineering books underline: rigidity, cumbersomeness and inefficiency. The software life cycle "agile" solution is an indication of a good direction as long as we don't oppose the fictitious "goal-based process [to] deliverable-based

[1] It is very important to underline that our model is far from being a description of reality: it is a simple transposition of the IEEE standard definitions [3]. The most important "absence" is the organisation's objectives with regard to which "System" and "User" are mere means.

[2] It is evident that the design of a system/software implies requirements, but these kinds of requirements must not be mixed up (as it happens so often, particularly with a use case oriented method) with the user requirements.

process" [5]. This opposition is fictitious because, without deliverables, not only we can't know whether the goals are reached or not, but we can't even speak of goals.

The four activities are temporally (not logically!) overlapped and a requirements method that aspire to disentangle the requirement process is doomed to failure if it is not deliverable-based. So, in our opinion, not only the opposition goal-deliverable is fictitious but unlike the champions of agile development we think that we must focus on "some sort of deliverable". But a deliverable is not necessarily a huge document that slows down the project and bureaucratizes the process. Before analysing the deliverable at the center of ER, in Figure 2 we present a transformation of the model of Figure 1 (a modelling of a requirements cycle when the machine is operational) in a model at developing time with the deliverable-specification at the center.

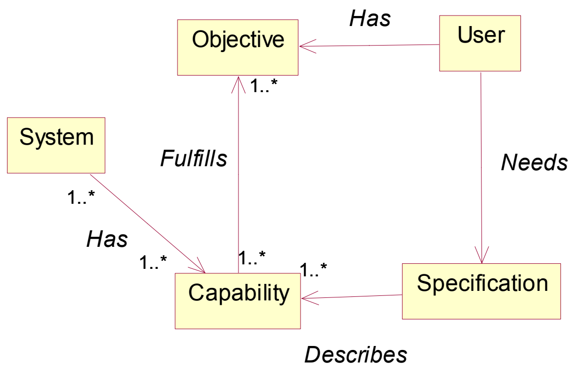

Figure 3: Specification

Before the construction of the system, the specification is the only access point to capabilities and, indirectly, to system. A "user *needs* a specification" as the model says, and "needs" is of paramount importance because it underlines the fact that there is no choice: if a machine must be built, the only access point before the machine construction is the specification. Is it a trite proposition? Yes, but beyond this trite remark there is a technical truth that a simplistic approach to agile development hides too often: somewhere we must "specify".

But the specification as an access point is also a filter and a noise generator. The representation of the capabilities in the specification cannot be totally accurate[3] and so some information is lost (filter) and some information is parasitically originated from write and speech acts (noise). So the problem is: how can one reduce the noise and augment the filter pass band? In our opinion a response for the class of systems indicated in the introduction, is that a specification must be a *composite specification prototype oriented* with a kernel functionally very similar to the system that must be realized.

[3] We do not consider formal specifications because, in a system as SCALCID, a mathematical formalism may introduce more problems that it can resolve.

ER specification context

The following figure presents a specification as a whole constituted of six (6) components.

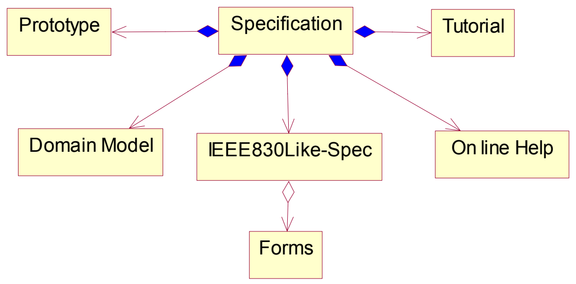

Figure 4: ER Specification structure

Prototype. A throw away prototype that, in addition to its participation at all four activities of requirement process, it's the stakeholder interface (user in particular) to all the other specification.

Domain Model. A description of the domain without any function or quality requirement.

IEEE830Like-Spec. A document whose table of contents complies with one of those presented in *IEEE 830 Recommanded Practice for Software Requirements Specification* [6].

- *Forms*. The forms constitute the core of the textual description aimed at defining quality requirements and functional requirements that are not "defined" in the prototype.

User Guide. A "classical" on line help and guide.

Tutorial. Explains some new concepts related to the IEC 61850 Standard. [3]

The model of Figure 4: ER Specification structure, allows us to specify some important aspects of ER:

1. There is not a specification document, a prototype, a domain model, an on line help and a tutorial, but there is ONE specification that contains all these components.
2. The prototype is not only the entry point for elicitation and validation as in a prototyping based approach. It is also the entry point for all other components of the specification. That is to say the prototype creates a specification environment very akin to the operational environment.
3. The relative weight of the components changes through the requirements cycle. The changes are dictated by:
   - The difficulties of implementation.
   - The knowledge and skills of stakeholders and of the "prototypers".
   - The project management requirements, etc.
4. The "forms" are structured representations of requirements akin to the person-machine

interface form so that the operations of capturing operational data and specification data are very similar.

SCALCID TEST BED

The context

HQ, a Canadian public utility, is developing ALCID II, a new distributed real time control system based on IEC 61850 standard [6]. IEC 61850 is a standard that specifies the protocols and application semantics for the interoperability between the substation Intelligent Electronic Devices. In this paper we analyze an ALCID II subsystem: the parameterization subsystem (SCALCID).

The main components of SCALCID are:

1. A relational data base (Oracle) that centralizes the data of all the HQ substations;
2. A system configurator for all the substations connected to the data base;
3. An Intelligent Electronic Devices configurator responsible for downloading the parameters into the IED.

The system configurator and the Intelligent Electronic Devices configurators exchange data via XML files whose schema is defined in IEC 61850.

The two main user roles are:

- *Standardization agents.* The instigators of the IEC approach are mainly concerned with establishing rules for restricting the technological and functional choices. They must be acquainted with all the elements of IEC-61850, even though they do not need to know all the details.
- *Functional engineers*. The main concern of functional engineers is that all the functions carried out by the old system are executed by the new system without any changes.

Our experience with previous HQ projects showed that:

- The elicitation of the requirement was often a simple acceptation of the expert's points of view (software and system engineers).
- A reliable validation of a SRS was difficult — practically impossible — to obtain with reviews.
- A prototype-based validation was often too concerned about human-machine interface details and so the user did not have a comprehensive functional view.

To overcome these difficulties we decided to use an approach where the elicitation and validation were done while the users were working on a "true" project.

This approach was possible only if:

- A data base existed (so the user could update the ALCID II parameters).
- An application existed.

But because the requirements must be elicited and validated one cannot have the final application! Because of all these requirements and constraints, we decided to adopt an ER approach for the requirement process.

Specification environment

Since May 2006, *L'Institut Trempet* at the *Université du Québec à Montréal* is implementing a specification environment (SEER) to facilitate an ER approach to the development of SCALCID. The main objective of SEER was to support the stakeholders in the requirement engineering process. To better achieve the main objective two more objectives were added: *Database design and implementation* and *IEC 61850 learning support.*

The database design and implementation were iterative and brought forward in concert with the function's specification. This allows users to fill the database step by step and work with "true" data. The necessity for the learning support was dictated by the fact that the IEC 61850 standard is very complex and the comprehension is facilitated by "learning while working" approach [8]

One can ask what exactly makes SEER a specification environment and not a first release of the final system. Or, in other words, what was absent from prototype so that, when it is considered with the database, it is not an evolutionary system but a simple prototype? Two elements that are fundamental for a operational and maintenable system are absent:

1. *The quality of code*. The code was written without concern for the maintenability; and maintenability, for a system with a life cycle of about twenty years as SCALCID, is of paramount importance
2. *The human machine interface.* No usability life cycle was applied and there was not a reliable design of the human-machine interface: the forms and menus were built posthaste to satisfy user demands or to propose something to facilitate the requirements elicitation. This choice was dictated 1) by time constraints; 2) by the stakeholders belief that the human-machine interface is not the concern of software engineering and, last but not least, that a ugly and not user friendly interface can facilitate the learning [8].

Forms versus prototype

As we have seen the textual structured forms are the hard core of the IEEE 830 like specification. Here are the most significant fields of the SCALCID forms:

Project management related information: *Necessity*, *Priority* and *Stability***.** These fields concern the final product, not the specification. The necessity and the priority of prototype's requirements were specified in a document not integrated into SEER. *Stability* is particularly important in SCALCID because the order of capabilities implementation is dictated by its magnitude (Most Stable Element First approach [10]).

Requirement engineering related information: *Input*, *Pre-conditions*, *Post-conditions*, *Description*. The content of these fields was jointly prepared by a standardization agent and a requirement engineer. *Origin, Derivated*. These fields describe the "parent" of the requirements and his children as in SysML [9].

Software engineering related information: *Rationale***,** *Conflict*. These fields facilitate the design of the final product and the early detection of conflicts (which does not necessarily means "conflicts resolution"). The conflict field, in particular, is a means to go deeper into the analysis.

Explicit interaction: *Open questions.* This field was introduced to favor the interaction between developers and functional engineers and between functional engineers and standardization agents. As for the *Conflict* field, the answer does not have to be given quickly: it is the life cycle of the question that is more important. The duration and, above all, the sub-questions generated, are a means to deepen the problem understanding (the *conditio sine qua non* for requirements validation).

In the first months of the project the sharing of the description between prototype and forms was mainly based on subjective considerations about complexity. When the complexity of implementation was high and the complexity of description was low the prototype was considered as a simple access point to the other specification's components and the functions were described on the form. On the other hand, when description was complex and implementation simple, there was no functional description in the form and the function was "described" only via the interface of the prototype — in this case the form contains the project management related information (*necessity*, *priority* and *stability*).

But because relationships between forms and prototype are dynamic, the requirements description can partially or totally flows from forms to prototype and vice-versa. The main flow at the beginning of the project was from forms to prototype and in the end from prototype to forms. This flow created a trend toward a specification where the requirements were partially on the prototype and partially on the form.

If the complexity was the trigger of the sharing, the final sharing was influenced by: difficulties of the IEC 61850 concepts, user's roles, necessity of working on "real" data and human resources availability.

NOTE Practically the flow was more complex because the on line help and tutorial too participated to the requirements movement. END OF NOTE.

CONCLUSION

Despite similarities, ER is not a "classical" prototyping method because the prototype is not only the entry point for elicitation and validation but also and above all:

- The entry point for requirements specification via structured forms that will possibly become a starting point for a new version of the prototype.
- A working and learning tool.

Next table presents the author's personal qualitative evaluation of the importance of the specification's components for each activity of the requirement process.

Table 1 Specification components versus activities

	Elicitation	Analysis	Specification	Validation
Prototype	++	+	++	++
Form	+	++	++	+
Domain modelling	+	=	++	++
On line Help	+	=	+	++
Tutorial	=	=	+	+
++ (very important); + (important); = (not important)				

Lessons

The following is a short list of problems and/or lessons learned through the application of ER to SCALCID,

1. The verification of the specification consistency is harder than the verification in a more traditional approach.
2. The update of forms contents requires a more sophisticated mechanism than e-mail, telephone and files sharing.
3. The flow of requirements information from prototype to form can be excessively slowed down because it's psychologically difficult to destroy something concrete in order to create a more abstract view of requirements.
4. The association between the prototype Java classes in charge of the access to forms and the files and directories containing the forms is too difficult to manage.
5. The interactions with users are too dependent of the project leader.
6. ER can generally be applied only if the tools in SSER are better integrated than in the present solution.
7. A Database must be considered above all as a model of the real word and not a computer science object [2].

We shall end the paper with some questions whose responses, in my opinion, are far from being easy.

Could the integration of wiki based approach [11] to ER be a way to partially settle the problems number 2, 5 and 6 of the above list?

Is it possible to establish a formal method to manage the information flow from prototype to forms?

The existence of a domain model is it a precondition for the ER approach?

ACRONYMS

ER	Extreme Requirements
IID	Incremental and Iterative Development
SEER	Specification Environment for Extreme Requirements
SRS	Software Requirements Specification
UML	Unified Modeling Language

REFERENCES

[1] I. Maffezzini, “Discours des méthodes”, **Génie logiciel***,* Mars 2007, No 80.

[2] M. Jackson, **Problem Frames***,* Addison-Wesley, 2001

[3] IEEE, std 610.12 1990, **IEEE Standard Glossary of Software Engineering Termnology**.

[4] IEEE, **SWEBOK Guide to Software Engineering Body of Knowledge**, 2004

[5] J. Arlow, I. Neustadt, **UML2 and the Unified Process***,* Addison-Wesley, 2005.

[6] IEC, IEC 61850, **Communication networks and systems in substations**.

[7] IEEE, std 830 1998, **Recommended Practice for Software Requirements Specification**.

[8] I. Maffezzini, A. Premiana, “E-Learning whilst Eliciting ans Working – IEC 61850 Substation, case study”, In Proceeding, **1st Intern. Conf. on E-Learning in Industrial Electronics**, ICELIE’2006**.**

[9] OML, **Systems Modelling Language Specification**, Final Adopted Specification, ptc/06-05-04.

[10] I. Maffezzini*, Most Stable Elements First Approach* In Proceeding **3rd International Conference on Cybernetics and Information Technologies, Systems and Applications,** CITSA 2006

[11] B. Decker, et alii, “Wiki-Based Stakeolder Paticipation in Requirements Engineering”, **IEEE Software**, March 2007.