A Case against the Use Cases in the Classroom?

Bernardo Ventimiglia
TREMPET
University of Quebec At Montreal
CP 8888, succ. Centre-ville
Montreal QC H3C 3P8
Canada
Ventimig@Labunix.uqam.ca

Louis Martin
Dep. of Computer Science
University of Quebec At Montreal
CP 8888, succ. Centre-ville
Montreal QC H3C 3P8
Canada
Martin.Louis@uqam.ca

Abstract

The ubiquitous presence of the use cases approach is a fact. Its popularity is based on true success the approach has brought to practitioners. It's back with abundant literature. But are use cases a true requirement engineering approach or are they more a system design approach? Should methodologies based on use cases be the only ones taught in the undergraduate university program?Based on actual experiences in teaching requirements engineering, this paper argues in favor of a mixed approach where use cases are complemented with a more "domain oriented" style: the Jackson approach. Some implications of our teaching experience are presented.

1. Introduction

1.1. Requirements Engineering

Requirements Engineering (RE) as implicitly defined in [1] subsumes system and software requirements. The definition of requirements of [1]: "A requirement is a property that must be exhibited by a system developed or adapted to solve a particular problem" is a synthesis of the paragraph 1) et 2) of the requirement definition of [2]. The introduction of the RE discipline allowed the change of the requirement process from a start-up task to a set of tasks that spans all the product's life cycle. The main reason of this change of perspective is probably the end of the big hope that had been generated by requirements-centered development and the passage to an architecture-driven and evolutionary approach [3]. This span creates an interweaving of programming-oriented tasks with user and requirements-oriented tasks and creates also the bases for the success of UML as a language for requirements and design. The introduction of a seamless approach based on a language had, from our point of view, an inopportune consequence: the creation of a "foggy" zone where requirements analysis and system design are blurred. This "foggy" zone is dramatically dangerous in a classroom where the students have not the technical maturity that allows the best of practitioners to advance in spite of "fog".

The Unified Process (UP) is a very popular process for developing object oriented systems based on UML and that has the Use Cases (UC) as the central method for requirements specification.

This is the context in which we want to raise some questions about the use of use cases as a method for RE in the classroom.

1.2. The teaching context

The *Baccalauréat en informatique et génie logiciel (Undergraduate degree program on computer science and software engineering)* de l'*Université du Québec à Montréal* (UQAM) has three 45 hours courses on RE:

1) INF-5151 (Software Engineering I: Analysis and Modeling) is mandatory and is the first and main course in RE (see table 1 for the course description);
2) INF-4150 (Human-Machine Interfaces) is optional and concerns mainly the design of GUI;
3) INM-5151 (Analysis and Modeling Project) is mandatory and is a practical course where students must write a complete Software Requirement Specification (SRS) based on [4].

We will analyze only INF5151 because this is where all the main concepts of RE are introduced. In the Fall 2001, the course textbook was [5]. In Winter 2002, two of the three teachers used two textbooks: [5] and [6]. The majority of winter students were dissatisfied with the Jackson approach presented in [6]. Their main complaints were: "too theoretical", "nothing concrete", and "too vague". Even the teachers were dissatisfied. We decided to retry the experience of mixing UP approach as supported by [5] and the more problem-oriented approach of [6] in the Summer 2002 semester. Our decision was based on the fact that the students in the following course "Software Design", when prepared only with the UP approach, were not able to understand the domain problem of an industrial

SRS that was used as input for their software design course. Our hypothesis was that students equipped with the two approaches would be better prepared to face the challenge. This hypothesis will have to be confirmed or rejected with future work.

Because of the negative feedback on the Jackson approach, we decided to organize all the lessons around a well-known domain problem (we thought it was well known!) : the management of a family CD, DVD and video inventory.

The students, in group of two or three, had to produce three artifacts:

- A Concepts of Operations written according to [7]. The ConOps had to be delivered after four weeks based on a template provided by the teacher.
- A SRS written according to [4] with a table of contents modified to contain two context diagrams : one according to UP (use cases context diagram) and the other according to Jackson (context diagram).
- A first level GUI prototype.

2. The two approaches

In this section we present the main elements of the two approaches (UP-Larman based on [5] and Jackson based on [6]) that drove our teaching experience.

2.1. *RE: Use Cases centered (UP-Larman)*

UC is "a description of a set of sequences of actions, including variants, that a system performs that yields an observable result of value to an actor" [8] and is very useful to "clearly identify the boundary of the system"[10]. UC is, presently, an ubiquitous approach to system and software analysis. Larman in [5] summarizes clearly the relationship between UC and requirements: "*Use cases are requirements* […]. *In the UP — and most modern methods — use cases are the central mechanism that is recommended for* […] *the discovery and definition* [of functional requirements]". Even if Larman insists that UC are textual, it's clear that the strength of UC lies in their discreteness — the fact that they are organized in short numbered paragraphs. This discreteness facilitates the move toward the System Sequence Diagrams (SSD). The students like SSD because they get the impression that they founded the key-events to limit the system. We wrote "impression" because SSD are made before the writing of the contracts. At that stage, a lot of ambiguity is still present, for instance, the formal definition of the parameters are not fixed. But in the UP-Larman approach, the contracts are specified in the design discipline. In this approach, the context is functional and is described via a transparent rectangle containing the names of the CU (see Figure 1 for an example),

In Larman, the "domain model" is produced after the UC definition. The "domain model" is composed of the concepts with their attributes and the association between them relevant to the problem under study.

It is certainly not useless to emphasize that a object oriented process as UP has a functional description (UC) as a "central mechanism".

2.2. *RE: Domain centered (Jackson)*

The theoretical foundation of Jackson's approach can easily be grasped via his definitions of requirement and specification: "A specification forms a bridge between requirements engineering, which is concerned with the environment, and software engineering, which is concerned with the machine" [9]. In defining specification, implicitly, Jackson establishes a base for a definition of RE that is different from the SWEBOK definition [1]. But such an important difference for the software engineering foundation is not essential for our topic. If we introduce a "machine" to help solving a problem: "[t]he problem in not in the computer interface – it is deeper into the world, further away from the computer" [6] .

All artifacts made during RE concern mainly the understanding of the problem domain (psychological, technological and functional imbalance between the present and the anticipated situation). The term "world" in the M. Jackson's sentence must of course be interpreted as "the reality as presented in human words". But if the world is described in human words, it's clear why as stated in [11] "The requirements engineer must be a good listener". "Being a good listener" is not a moral quality, but is about intelligence and natural language skills. Briefly: the requirements engineer must "listen to the world" because the problem is in the world and the world speaks only through human beings (the stakeholders in the RE jargon).

The Jackson approach is based on world analysis and problem structuring, because "Our problems and requirements are in the world, not in the computer. We must focus in them directly, and describe them conscientiously" [6].

The first step is to draw a context diagram (see the examples in Figure 2a and Figure 2b) where all the "relevant" domains are represented with their interfaces with the machine. The second step is to draw a problem diagram where the requirements are added to the context diagram. The "drawing" of the problem diagram is not a simple task and is not a task that you can do perfectly the first time. The quality of the specification (the bridge that allows the transition from the problem in the world to the problem in the machine) is deeply dependent on the problem diagram that is, at the same time, the source and the center of the elicitation and analysis.

Jackson introduces the concept of "problem frame" as problem "stereotype" that can help in the specification. Problem frame is the equivalent of design patterns in software design with an important difference: the design patterns are the smallest "grains" of the design and, on the other hand, problem frames can (must) be subdivided to fit in smaller problems (subproblems) that are projections[1] of the problem that initiates the process.

2.3. Context diagrams: UC versus problem

The UP-Larman context diagrams and the Jackson context diagrams are very different and they are a consequence of two "opposite" views of the RE. We think that Jackson can accept the definition of [5]: *The context defines the environments in which the system lives*, only if we substitute "system" with "machine"[2]. The main difference between the two approaches are about the relative importance of system and environment. In a UC centered approach as UP what is central is the system as seen by the actors, in the Jackson approach the center is the domains that constitute the environment (the problem).

In one case, we organize the requirements starting from functions and their decomposition through UC, in the other case, we analyze the domains "as part of the world that are relevant". The difficulties for the analyst is the choice of the UC and of their level of details for one approach. On the other hand, the difficulties are in the choice of the "relevant domains". If from a pedagogical point of view it is important to straighten the differences between the two approaches, in practice there are some elements in common because the "relevance" of a domain is related to the functions.

3. Case Study

In this section, we present the problem that the students had to solve and some of the difficulties in the problem understanding that we think could be defeated only if UC are not the "central mechanism". That does not mean that what we are saying is true for all the problems but that there is at least one kind of problems where the domain analysis must precede the UC.

The table 2 presents a summary of the statement triggering the session work.

3.1. *The context diagrams : functions and domains*

When the students can draw a diagram they are "happy", and so is the teacher. Why? Because they have the feeling of precision : lines, circles, rectangles… everything is so neat! When they write a context diagram they are happier, because the system is now bounded.

We asked the student to insert two context diagrams in their SRS: one prepared according to UP and the other one according to Jackson. The fact of putting the two diagrams in the same document was a way to show to the students the differences between a context diagram based on functionalities as proposed by UP-Larman and a context based on the domain as proposed by Jackson.

In the case of our project, the UC where straightforward. Even the students who were not very interested made a "good" UC context diagram. See Figure 1 for the typical UC context. There were unimportant differences between the UC context drawn by each student. All diagrams were a simple transposition of the primary functions described in the ConOps.

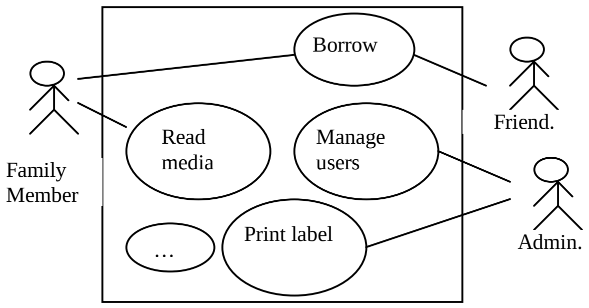

Figure 1. UC Context. A typical diagram presented by students,

The domain context was more difficult to draw, because the choice of the "relevant domains" generated a lot of questions. Is the house a domain ? and the family and Outlook ? Why ? The questions forced the students to analyze the problem and not only the functions. Figure 2.a shows the simplest context diagram drawn by the students — too simple, so simple that his only usefulness in for the Human Machine Interface design. But even if it is simple, it contains a domain that is not directly connected to the system as the UP actors. This topology "forces", in a sense, the student to analyze media independently of the machine. The figure 2.b presents the most complex context where the students take into consideration seven (7) relevant domains besides the machine domain. They were forced to study things that they thought were simple and that they thought they can ignore at the beginning. For example only one group put the house as a relevant domain and only one group understood that the "label" identifying the position of the album was not a simple string. They proposed a structure for the label that was the consequence of the analysis of the house domain. The others groups not only didn't see "the problem" at first, but even when we explain our solution they did not understand the importance of structuring the label according to the house structure.

[1] Projections and not partitions as in UC decomposition.
[2] The change is not insignificant, particularly with the students that like to put the word "system" everywhere. In our course we decided to ban the use of the term "system", because of its pedagogically dangerous ambiguity.

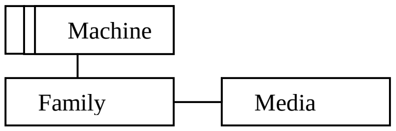

Figure 2a. "Jackson" Context: The simplest diagram of the course

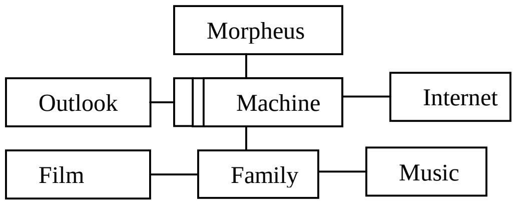

Figure 2b. Jackson" Context : The more complex of the course. Outlook is considered relevant by the student because in the *ConOps* we introduced a constraint about the necessity of automatically sending a letter to a lender when his/her state was "late".

3.2. The conceptual diagram: the "movement" trap

In the previous section, we made a quick reference to the difficulty of analyzing the "label". In this section we will take into consideration another point with which all the students had difficulties. From the functional point of view, the problem was straightforward: the program must compare the length of classical music movements. The length of a movement is written on the album, so… no problem ! But, often in RE, when there is "no problem", there is a big problem.

All the students were familiar, of course, with pop music but only 3 out of 40 listened sporadically to classical music. They had no idea about the concept of musical movement. The majority of them made a direct association between a song and a physical track on CD. When confronted to the well-known "Piano Sonata No.21 in C major, Op. 53 Waldenstein-sonate" in three movements, then they accepted the mapping of a movement to a track but had some difficulties to figure out that the track number did not correspond to the movement number. The difficulties increased when they tried to understand that "allegro con brio" was an attribute of a musical movement but duration was an attribute of the movement execution itself. Should they know that? In theory yes, because one user has expressed the need to compare the movement duration. Moreover, a "domain specialist" was invited in the classroom to respond to any question students might have. They had the time to prepare themselves for that interview and they were cued to cover the movement subject. Why it was so difficult to grasp that duration of the movement was not an attribute of the piano sonata but of its execution ? We think because they don't analyze enough in detail the relationship between the movement characteristics established by the creator and "freedom" of the interpreter. But why they stopped to early their analysis ? May be because they are not used to work hard on the clarification of concepts expressed in the natural language.

4. Teaching the ambiguity

If the road from vague needs to a formal and unambiguous SRS is long, hard and full of traps for an experienced software engineer, how can we teach it without losing the intellectual richness which springs from the complexity of the realm that the language tries to master ? The famous sentence that Hegel applied to philosophy is perfectly at home in RE too: "*the general drift is a mere activity in a certain direction, which is still without its concrete realization; and the naked result is the corpse of the system which has left its guiding tendency behind it*" [12]. The bare SRS is only a corpse when students have no access to an expert doing RE!

The ambiguity around the definition of the analysis and its current prescriptive artifacts has great impacts on the way we can and should teach RE. These impacts are not necessarily negative. Consider two extreme situations:

1. The teacher who is not conscious of the ambiguity around analysis will teach in a way that everything appears to be well defined. This lack of consciousness is not always linked to a lack of knowledge in pedagogy or in the ER domain itself but can be the result of a lack of practical experience in complex system analysis or in an excessive confidence in the "standards" of the day. This teacher will tend to negatively evaluate students that seam to not see thing "clearly" and s/he will try to direct them toward "formal approaches" to make the perceived ambiguity disappear. Today some of these "formal approaches" include system sequence diagrams, state diagram, etc. For the majority of the students, this teacher will bring a secure way of seeing "reality".
2. The teacher who is well aware of the ambiguity of the "reality" will teach in a way to uncover some of the multiple and all-valid points of view of a problem domain. This approach risk to discourage students who mainly seek a recipe to analyze problem domains. They will have the feeling that they are not capable to grab the essence of a domain and that they will never be able to do so.

5. UC centered approach, easiness and trend

To place and understand better our critic, let's summarize here two « hidden paradigms » which orientate our teachings :

1. When they are short in time (15 weeks to learn RE) and facing an alternative, students, as any human being would do, choose always the

easier way, the one which give them the impression to speed up.

2. College education does not give enough importance in speaking and well writing (and therefore well listening), and then students do not push hard enough the possibilities of natural language, specially if teachers insist on its ambiguity

To our mind, it is clearly more simple to establish the frontiers of a « system » and solicit it with events than to examine the environment in which it stands. Finding constraints imposed by the world to a machine that we want to build is complex and cannot be reduced to exchanges between the actors and the system. This apparent simplicity often generates *ad hoc* systems which later will be difficult to adapt to changes. In the context of undergraduate studies, it generates a "work style" which is not demanding enough to their ability to manipulate natural language.

The Larman view ("*UC are requirements*") is interesting because it says a lot about the synonymity, made by almost all UML followers, of UC and functional requirements. But we feel uncomfortable with this formula : UC are only one method, one tool to help requirement elicitation and specification. An experienced analyst can benefit from a UC centered approach because it is one of many tools he have in his hands to better understand a problem domain. An experienced analyst knows the limits and the strengths of the tools he uses, this is the essence of experience.

For an undergraduate student, it reminds us the phrase: "If the only tool you have is a hammer, you tend to consider everything around you as a nail". But even if in [5] the differences between the domains are not revealed, the practices and the coverage of UC in the context of closed systems, real time systems, information systems are quite different.

Teaching use cases to undergraduate students in IT with no real working experiences of any domain and when their prerequisites are mainly programming languages, computer architecture and mathematics induces a bias toward a design view of the subject instead of a more comprehensive view toward the domain itself. Even if we use problems in a domain they know well: music, CD, etc.

Actually we consider that the UC approach is used in sector where it can be harmful. The tendency to do so is based on a few factors:

- it's the trend;
- it's easier to use an approach that reassure us instead of trying to explore a domain where nothing seem certain.

6. Conclusions

The feedback from the students was positive, even if they found that the Jackson approach was harder and less evident than UP. They didn't think that it was "too vague": eventually they judged it "too concrete" because of the necessity of analyzing in a great detail the "physical" albums. We too were satisfied because we have the impression that, at least the best students acquired a "style" more RE oriented and less system design oriented than the students that we have had the previous years.

RE is about understanding the environment where the machine will operate from the points of view of the distinct domains that contribute to the problem. RE is not only about events and system : events and system are above all elements of system design. RE it's about finding commonalities between stakeholders points of view as well as contradictions (and to try to resolve them if possible). In RE, apart from some very special domains, the requirements engineer works with natural language to build the bridge between needs and a formalized SRS. If this is our vision of the ER ; if we think that a good modeling tool should be transparent and help the understanding of the reality it tries to model ; if we think that a UC centered approach mix-up RE and system design, why we didn't teach only the Jackson approach ? Because the students have others courses in the curriculum where the UC approach is a precondition, because there is others teachers that do not necessarily agree with us, but above all, because employers request UP, UML and Rational Rose mastering[3].

The difficulties with "labels" and with "movement" help us to understand that only an analysis of the problem independent from the exchanges at the machine border can bring to a machine that is truly stable.

We think that teaching, when is linked to the world outside the university, allows to see with a greater clarity the drawbacks of certain choices. Teaching ER to software engineering students with the intent to bring them to specify requirements and not to design the machine, makes obvious the fact that a clear cut in necessary between the ER and the rest of the development. This separation does not contradict an iterative approach as UP but emphasize the distance in skill, knowledge, and methods between the "discipline" of requirements and the discipline of system and software design. A consequence of this "belief" is not only that every seamless approach is doomed to fail but also that RE is more comprehensively thought based on [9] than based on [1].

As last point, we want to emphasize that our position is not against UC centered approach as much as to promote a shift of focus in teaching RE because we consider harmful to teach only one point of view like "une pensée unique" even if this point of view has a lot of valid aspects and is accepted by well-known "thinkers" of the RE.

[3] This is an important constraint, we form undergraduate students mostly for the industry.

Table 1. INF5151 Software Engineering: Analysis and Modelling. Course description.

Explore the basis and evolution of analysis methods. Complete the detailed study and application of a method. Define the user's role in all the steps of design and development. Notions about systems and a systemic approach. Choice of the life cycle model. The various steps of the process. Exploration of the domain, feasibility study, definition of the system (hardware and software), requirements specifications. Dynamic and functional object modeling. Tools for requirements definition. Critical review of the methods used in industry and evolution of the user's role in system development. Quality of specification and interface

Table 2. A summarized statement of work for the session

A modern family composed of four generations (children, parents, grandparents, grand grandparents) has over 1000 CDs, 200 DVDs and 400 videocassettes. The family members use a total of 7 PCs at home and one at work. The principal needs are: • Managing the inventory including for every album: title, localization (to be printed on a label), owner, cost, and titles of the each part. If the album is lent: lender's name, to whom, when and the anticipated date of return. The borrower must have a degree of trust. • Being able to obtain lists for music based on: author, singer, orchestra, orchestra director, artist, year of recording, language, author's or singer's country, category, etc. • Being able to obtain lists for films based on: director, scriptwriter, actors, year, original language, film director's country, category, etc. • Specifically for Deutsch Gramaphon series, being able to do exhaustive research. • The same GUI should please the 96 years old grand grandfather and also the computer fan. • The software should work on Windows and Access. PS: Is it possible to transfer data between Morpheus Web site and the software? Also is it possible to manage music recorded of the hard disk with the software? PPS: The father wants to be able to compare the movement's duration

Acknowledgements

Particular thanks to Guy Tremblay for his wise commentaries and to Marguerite Deville and Véronique Dassas for the hidden paradigms.

References

[1] P. Sawyer, G. Kotonya, "Software Requirements", *SWEBOK, trial version 1.0* (Abram, J.W, Moore, editors), May 2001

[2] IEEE/EIA Std 610-12-1-1990, *IEEE Standard Glossary of Software Engineering Terminology*.

[3] G. Booch, *Object Solutions,* Addison-Wesley, 1996.

[4] IEEE Std 830-1998, *IEEE Recommended Practice for Software Requirements Specifications.*

[5] Craig Larman, *Applying UML and Patterns,* Addison-Wesley, 2002.

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

[7] IEEE/EIA Std 1362-1998, *IEEE Guide for Information Technology – System Definition – Concept of Operation Document.*

[8] G. Booch, J. Rumbaugh, I. Jacobson, *The Unified Modeling Language User Guide,* Addison-Wesley, 1999.

[9] M. Jackson, "The meaning of requirements", *Annals of software engineering, Volume 3 (1997) - Software Requirement Engineering,* Baltzer Science Publishers**.**

[10] G. Schneider, J. P. Winters, *Applying Uses Cases,* Addison-Wesley, 1998.

[11] C. K. Chang, M. Christiansen, "Blueprint for the ideal Requirements Engineer", *IEEE Software,* March 1996.

[12] G.W.F. Hegel, *The phenomenology of Mind,* translated by J.B. Baillie , http://www.ets.uidaho.edu/mickelsen/ToC/Hegel%20Phen%20ToC.htm.