Journal of Computer Science & Its Applications, June 2007, Vol. 14 No. 1

From Requirements Elicitation to Requirements Modelling: An Experience Report

Daramola, J.O. and Adebiyi, A. A.
Department of Computer and Information Sciences
Covenant University, Ota, Ogun State, Nigeria.

wandi_ex@yahoo.com, ariyo_adebiyi@yahoo.com

ABSTRACT

Requirements elicitation is one of the vital aspects of requirements engineering that seldom gets sufficient attention during software development, although it has been found to play a pivotal role in the success or failure of most software development efforts. This is because it provides an opportunity for all stakeholders in the software development process to access, review and understand the user requirements for a system and the limitations of the software development activity.

In this paper, we give an experience report of the application of use case modeling as a technique for requirement elicitation in the development of a Course Registration System (CRS) for a Nigerian University. The results obtained confirmed the high payoff of placing adequate emphasis on requirements elicitation during a software development activity.

Keywords: ***Requirements Engineering, Requirements Elicitation, Use Cases Modelling, Object Constraints Language (OCL) Specification.***

1. INTRODUCTION

Requirements are the statements of what a system must do, how it must behave, the properties it must exhibit, the qualities it must possess and the constraints that the system must satisfy [1,2,3].The hardest single part of building a software system is deciding precisely what to build. Therefore, the process of emphasizing the utilization of systematic and repeatable techniques that ensures the completeness, consistency and relevance of the system requirements is called Requirements Engineering (RE) [4].

It has been observed that the inability to produce complete, correct and unambiguous software requirements is the major cause of software failure today [5]. Therefore, the requirement phase of the development process has the greatest tendency to cripple the resulting system if it is not done correctly. Also, it is more difficult to rectify problems at a later phase of development other than the requirement phase [5]. All of these make requirements engineering a very core activity of the system development life cycle with the key objective of specifying a system that can be successfully realized.

Requirements analysis is the aspect of requirement engineering process in which what is to be done is elicited and modeled. This process

has to deal with different viewpoints, and it uses a combination of methods and tools to produce requirements document. Some of the challenging issues of requirements engineering process include [6,7]:

a. Achieving requirements completeness without unnecessarily constraining system design;
b. Analysis and validation difficulty;
c. Changing requirements over time;
d. Users do not know what is technically feasible;
e. Users may change their minds once they see the possibilities more clearly;
f. Users may not be able to accurately describe what they do;
g. Discoveries made during the later phases may bring about change to requirements;
h. Social, political, legal, financial, and/or psychological factors;
i. It is impossible to say with high certainty what the requirements are, and whether they are met, until the system is actually put in place and running correctly.

The implication of lack of a complete requirements specification, which evolves from elicitation process, is a grievous one. According to Connolly et al. [8], one of the major reasons for the failure of software projects is lack of complete requirements. The problems that result from inept, inadequate, or inefficient requirements engineering are expensive and plaque most software systems and software development organisations [9]. Despite the current trend to rapidly develop applications companies are realizing that the consequences of inadequate requirements engineering are too great [10].

Many of the problems in creating and refining a system can be directly traced to elicitation issues. Yet, much of the research works conducted on requirements engineering have ignored this important phase of development. Indeed, there has been a concentration of research in the area of precision, representation, modelling techniques, verification, and proofs as opposed to improving the elicitation of requirements. Hence in [6] there was a clarion call for more research efforts should be directed towards methods and tools needed to improve the requirements analysis process, and in particular, to those providing more support requirements elicitation.

Some of the specific instances in literature where use case modelling have been used a medium for requirements elicitation include: The Object Oriented Software Engineering method by Jacobson [11, 12] which describes how use cases relate to system analysis and test models. The 4+1 view model of software architecture in [13] emphasizes the importance of use case modelling in the representation of architectural views of a system from the user's viewpoint. Use case diagrams are fully incorporated into the UML notation [13,14,15]. Also, UML based software development methods, such as COMET [14] and Rational Unified Process (RUP) [16], all start with use case modelling to describe software requirements, while the role of use cases in the requirements and analysis modeling phases of a model-driven software engineering process was presented in [17].

However our objective in this paper is to show the payoff of placing adequate emphasis on requirements elicitation during software development. This we have demonstrated through the use case modelling of an automated Course Registration System (CRS) for Covenant University, Nigeria.

The rest of the paper is briefly summarized as

follows: section 2 gives an overview of software requirements elicitation. In sections 3, we present use case modelling a requirements elicitation and a description of how to used as basis for complete requirements modelling the CRS for Covenant University. In Section 4, we give an experience report of the noticeable gains discovered during the later phases of development of the CRS and in section 5 we give our conclusion and generalizations.

2. SOFTWARE REQUIREMENTS ELICITATION

Software requirements elicitation is a process by which all parties involved in the development of a software system discover, review and understand user needs and the limitations of the development activity and the software [2]. Requirements elicitation serves as a front end to systems development, which involves various stakeholders: analysts, sponsors, developers, and end users. Requirements engineering process can be broken down into three main activities, which are [6]:

a. Generating requirements from various stakeholders by making the users the major focus;
b. Ensuring that the needs of all stakeholders are consistent and feasible; and
c. Validating the requirements derived in line with the needs of the stakeholders

This model involves a sequential ordering of activities in iterative mode and the activities should be properly documented. Requirements elicitation can be broken down into the activities of fact-finding, information gathering, information documentation and integration. The resulting product from elicitation is a subset of the goals of the various parties, which describe a number of possible solutions. The remainder of the requirements engineering process concerns the validation of this subset to see if it is what the stakeholders actually intended. This validation typically includes the creation of models to foster understanding between the parties involved in requirements development. The result of a successful requirements engineering process is a requirements specification, where the goodness or incongruity of a specification can be judged only relative to the user's goals and the resources available [6].

A good requirements elicitation process supports the development of a specification with a set of attributes such as those defined in [6]. There are major problems with requirements elicitation that inhibit the definition of requirements that are unambiguous, complete, verifiable, consistent, modifiable, traceable, usable, and necessary. However, the way out of these problems is to refine the processes of fact-finding, information gathering, information documentation and integration to specifically address these problems. This is where the ingenuity of the technical team comes into play. The Problems of requirements elicitation can be summarized into three major categories [6]:

a. Problems of scope, in which the requirements may address too little or too much information;
b. Problems of understanding, within stakeholders such as users and developers; and
c. Problems of volatility, which is the changing nature of requirements.

The following is a set of recommended requirements elicitation techniques [18]: Interviews, Document analysis, Brainstorming, Requirements workshop, Prototyping, Use cases, Storyboards and Interface analysis. These

techniques can be used in combination and they have been found effective in emerging the real requirements for planned development efforts and models.

2.1. Process of Conducting Requirements Elicitation

The following approach is recommended for carrying-out requirements elicitation. The approach is based on extensive review of literature combined with our practical experiences of requirements analysis. The purpose is to solve the requirements elicitation problems earlier identified:

a. Write a project vision and scope document.
b. List a project glossary and acronyms that provide definitions of domain words that are acceptable to the stakeholders for the purpose of effective communication.
c. Evolve the user requirements by focusing on product benefits and general acceptability.
d. Document the rationale for each requirement when necessary.
e. Provide training for requirements analysts and user representatives so as to achieve the following:
    i. identify the role of the requirements analyst, who are to work with end-users to evolve the requirements,
    ii. write good requirements document,
    iii. identify requirements errors and corrective measures,
    iv. identify investment needs on the project,
    v. understand the project and/or organization's requirements process,
    vi. suggest methods and techniques that will be used for the elicitation, and
    vii. understand the use of project's automated requirements tools.
f. Prioritize the requirements appropriately.
g. Establish a mechanism for requirements management.
h. Use peer reviews and inspections of all requirements work products.
i. Use an industry-strength automated requirements tools that:
    i. assign attributes to each requirement, and
    ii. provide traceability.
j. Use hybrid of good requirements gathering techniques such as requirements workshops, prototyping, and use cases.
k. Provide members of the project team (or stakeholders) with audiencespecific versions of the requirements when information is being shared.
l. Establish a continuous improvement ethic, teamwork approach, and a quality culture.
m. Involve customers and users throughout the development effort.
n. Perform requirements validation and verification activities in the requirements gathering process to ensure that each requirement is testable.
o. Store requirements in a requirements repository instead of a paper document.
p. Keep the granularity of the requirements repository small so that individual requirements can easily be entered, iterated, approved, traced, managed and published.
q. Ensure that requirements repository stores all kinds of individual requirements metadata and requirements models.

3. Use Cases Technique for Effective Requirements modelling

Use cases are pictures of actions a system performs, depicting the actors and they describe the behaviour of a system when a particular stimulus is sent by one of its actors. Use cases are used during the analysis phase of a project to identify and partition system functionalities. They are powerful tools when combined with their textual description to give better semantic information. Many developers believe that use cases and scenarios facilitate team communication. They provide a context for the requirements by expressing sequences of events and a common language for end users and the technical team. Use case modelling facilitates and encourages stakeholders' participation, which is one of the primary factors for ensuring project success. In addition, it provides a means of identifying, tracking, controlling and managing system development activities. Other requirements elicitation techniques should also be used in conjunction with use cases to provide enough information that enable development activities.

There are two primary artifacts involved when performing use case modelling. The first is the use case diagram, which graphically depicts the system as a collection of use cases (represented by oval shape), actors (represented by stick object) and their relationships, which are basically: association, extension, dependency and uses. Use case and actor icons are assembled into large system boundary diagram. This diagram shows all use cases in a system surrounded by a rectangle. Outside the rectangle are all actors of the system and they are tied to their use cases with lines. Inside the system boundary are use cases that are part of the system being modelled and everything outside is external to the system. The diagram communicates the scope of business events that must be processed by the system. The details of each business event and how the users interact with the system are described in the second artifact, called the use case narrative, which is the textual description of the business event and how the users will interact with the system to accomplish the task [19].

3.1 Use case modelling of Course Registration System

Use case modeling of Course Registration System (CRS) of Covenant University was undertaken after detailed study of the activities of the key actors in the registration process using use case cases and uses case narratives. The CRS is designed to enable students of the University to independently register course to be taken on per semester basis after initial clearance from the staff course advisers. It enables each student to select from the pool of relevant available courses to be taken in a particular semester without exceeding the maximum limit and after all necessary prerequisites have been satisfied. The program also validates the student's financial standing before registration is allowed. The use case diagram showing the several possible use case scenarios of the CRS is shown in figure 1 while the use case narratives are shown in tables 1-4 [20].

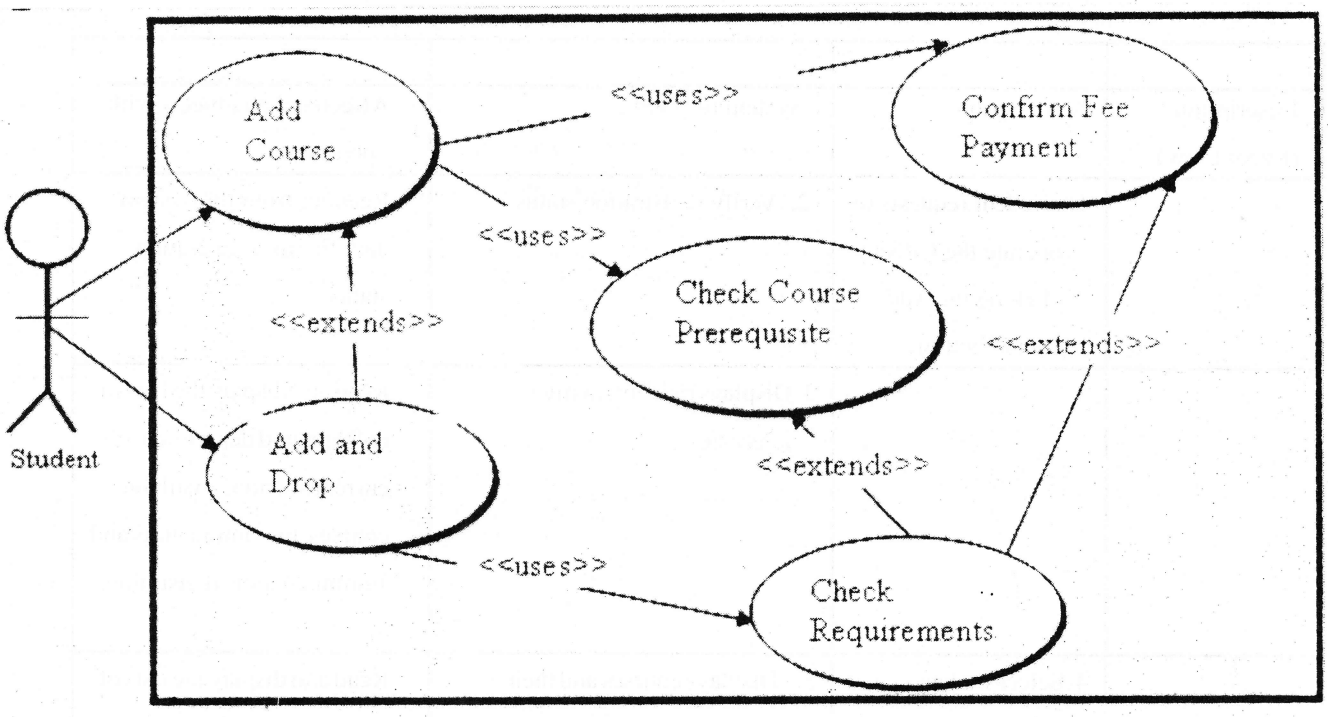

Fig. 7: Use Case Diagram for CRS

Table 1. Use Case Narrative for Add Course

Use Case 1	Add Course
Goal in content	Student will be required to add registerable courses via CRS if registration process is opened i.e. before the closing date, a user interface containing list of registerable courses is displayed with add buttons activated.
Level	This use case uses the confirm fees payment use case to determine student's eligibility to register courses based on whether the course prerequisite has been passed or failed.
Parameters	In: matricno, course code, college and department hosting the course, programme. Out: current session, minimum and maximum unit allowed per semester, unit registered per semester, students' name and photograph, list of courses available in the department and their corresponding prerequisites
Preconditions	Course registration is not closed; user is a good standing student of the university. Graded courses cannot be added. Total course unit registered per semester cannot exceed the maximum or fall below minimum allowed. Therefore a course is only added, after the prerequisites must have been passed.
Post-conditions (Success End)	Adding of course(s) is achieved.
Post-condition (Failed End)	Adding of courses is not achieved, cancel button is pressed and the process is canceled.
Actors	Students
Trigger	A student requests to operate the CRS for a CRS operation Add course operation

Description (Event Flow)	Actor action	System respond	Affected data objects with operations
	1. Student requests to operate the CRS by clicking the Add-course button	2. Verify registration status	Reading from the system data file for registration status
		3. Displays rich interactive interface	Read and display the lists of colleges and departments, current session, available courses, maximum units and minimum units registrable.
	4. Select student's college, department and programme	5. Displays courses and their prerequisites	Read and display the lists of courses and their prerequisites
	6. Enter Matricno and press start button	7. Verify student's fee payment status, disciplinary record, performance record.	Read from fee payment file, disciplinary record file, performance file.
	8. Select a course from course list and press Add button	9. Verify course to add	Check if all course prerequisites have been passed by student. If passed, add course to course list, if not display message that course cannot be registered because of failed prerequisites; update total course unit so far registered. If total course unit registered is more than maximum per semester display message stating that course cannot be added, else add course to course registration list.
Extensions	Add and drop use case		

Table 2. Use Case Narrative for Add and Drop Course

Use Case 2	Add and Drop Course		
Goal in content	Student could be required to add or drop a course via CRS if registration process is opened, a user interface is displayed with add and drop buttons activated		
Level	This is an extend use case from Add course use case		
Parameters	In: matricno, course code, college and department hosting the course. Out: current session, minimum and maximum unit allowed per semester, unit registered per semester, students' name and photograph, list of courses available in the department and their corresponding prerequisites		
Preconditions	Course registration is not closed; user is a good academic standing student of the university. If user is a fresh student, course prerequisite is not checked. Graded course cannot be added or dropped. Course unit registered per semester cannot exceed the maximum or fall below minimum allowed		
Post-conditions (Success End)	Adding or dropping of course(s) is achieved		
Post-condition (Failed End)	Adding and dropping of course(s) is not achieved, or cancel button is pressed and then the process is cancel		
Actors	Students		
Trigger	A student requests to operate the CRS for a CRS operation such as Add or Drop		
Description (Event Flow)	Actor action	System respond	Affected data objects with operations
	1. Student requests to operate the CRS by clicking the correct button	2. Verify registration Status (if registration is still open)	Reading from the system data file for current registration status
		3. Displays rich interactive user interface	Read and display the lists of colleges and departments available, maximum and minimum units registrable, and current session
	4. Select student's college, and department from list of colleges and departments	5. Displays courses and their prerequisites	Read and display the lists of courses and their prerequisites
	6. Enter Matricno and press start button	7. Verify student	Read from student bio-data file and display courses registered
	8. Select a course from course list and press Add button	9. Verify course to add	Update course unit registered per semester and add course to course registration list
	10 Select a course from course registration list and press Drop button	11. Verify course to drop	Update course unit registered per semester and drop course if possible
Extensions			

Table 3. Use Case Narrative for Confirm fee payment

Use Case 3	Confirm fee payment		
Goal in content	Student may check correctness of fee payment records before making attempt to register courses.		
Level	This is a basic use case for Add course, Drop course operation		
Parameters	In: matricno, current session. Out: Student name, photograph, department, programme, college, state of account (credits in blue, debit in Red), registrable students (True or false).		
Preconditions	Student are only considered eligible for registration for a new session if up to 90% of total tuition fee has been paid, otherwise registrable status is false.		
Post-conditions (Success End)	Student fee payment information is displayed.		
Actors	Students		
Trigger	A student requests to operate the CRS for confirm fees payment operation		
Description (Event Flow)	Actor action	System respond	Affected data objects with operations
	1. Student requests to operate the CRS by clicking the confirm fee button	2. Verify student's financial status	Reading from the system data file for fees payment records.
		3. Displays prompt for input	Display prompt to capture student's matricno and current session.
	4. Enter matricno and press start button	5. Displays student's record	
	6. Read from fee payment data file and display name, college, department, programme, photography, account balance, eligibility status (true or false)		
Extensions	Check Requirements		

Table 4. Use Case Narrative for Course Prerequisite

Use Case 4	Course Prerequisite		
Goal in content	Student could check the prerequisite for a particular course prior to adding it to list of registered courses.		
Level	This is a basic use case for Add course, Add and Drop course operation		
Parameters	In: Course code, current session. Out: List of prerequisite courses (course code, course title, course unit, course status).		
Preconditions	Course code entered as input must be valid i.e. exist within the available courses in the university academic program.		
Post-conditions (Success End)	Checking of course prerequisite is achieved.		
Post-conditions (Failed End)	Checking of course prerequisites not achieved, or cancel button is pressed to terminate process.		
Actors	Students		
Trigger	A student requests to operate the CRS for a check prerequisite operation		
Description (Event Flow)	Actor action	System respond	Affected data objects with operations
	1. Student requests to Check prerequisite by clicking the input button	2. Request student input	Display a prompt for student matricno.
	3. Student enter matricno and click next button.	4. Verify student	Reads student's database file to verify studentship status.
		5. Display user interface for checking course prerequisites	Display list of colleges, departments and programmes
	6. Student clicks on colleges, then department and then programme	7. Display list of courses available for selected programme	Reads and filter out courses that pertain to selected programme from the programme courses data file.
	8. Student selects from the course list, the particular course of interest	9. Display list of prerequisite courses	Read and display the list of prerequisite courses to select course. The information displayed include course-code, course title, course unit, and course status.
Extension	Check Requirements		

3.2 Class Entity Modelling of CRS

The detailed use case descriptions provided by the use case narratives artifact also provided adequate basis for accurate identification of objects and objects' interactions within the CRS. The class entity model of the CRS showing the entity classes and their associations within the CRS is shown in Figure 2 with the public and private attributes of each class member denoted with (+) and (-) respectively.

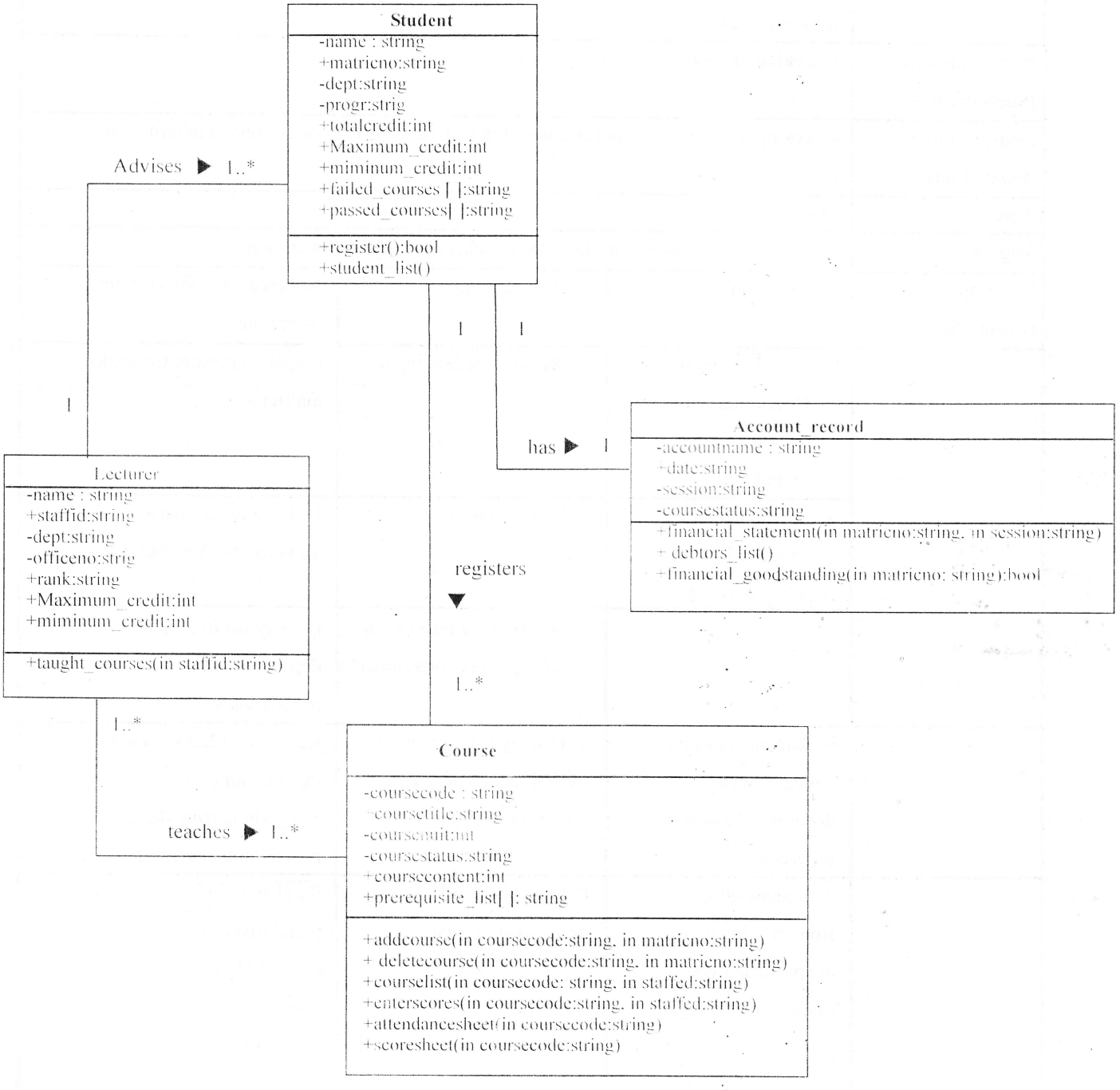

Figure 2: Class Entity Model of CRS

3.3 Formal Specification of Constraints on Requirements models

Object Constraint Language (OCL) is a formal notation developed so that more precision can be added to the specifications of systems being modelled with UML. All of the power of logic and discrete mathematics is available in the language [15]. The first step to OCL specification is to develop a UML model. Most typically a class diagram. Once the class diagram has been used to specify the relationships among the objects in the system, the OCL expressions are then used to specify the constraints on model elements so that implementers of the system can known more precisely what they must ensure remains true as the system runs.

In a diagram, a constraint is shown as text string enclosed in braces {...}and placed near the associated element or connected to that element by dependency relationship. Constraints can also be put in notes (comments). An alternative notation is to keep the constraints in a separate text file. The key terms in an OCL expression are:

- **Invariants**: is a property that must remain true throughout the life of a model element, such as an object. The reason for introducing invariants is to specify the constraints under which the system can operate. When a system carries out a task, the invariant should be true at the start of the task, and remain true at the end of the task.
- **Pre-condition**: A pre-condition is something that must be true before a particular part of the system is executed. The designer and implementer can use precondition to perform checks before an operation is executed. This is a way of presenting the system from entering illegal states.
- **Post-condition:** A post-condition is something that must be true after a particular part of the system is executed, if that execution was legal (that is, all pre-condition were met) and the system has successfully carried out its action.

Pre-condition and post-conditions can be applied to operations or to use cases, and as use cases are translated into operations through the analysis and design process, so they ultimately become translated into post-conditions on operations.

All OCL statements must take place within a context. This might be a class, association or a use case. The statements can be invariants, pre-condition or post-conditions. The UML convention is that the keyword **context** is written in bold type, and the stereotype of the constraint is written in bold type as **inv** for <<invariant>> **pre** for <<precondition>> and **post** for <<postcondition>>. Some of the OCL expressions specifying constraints on some elements of the class entity model of the CRS are given as follows:

```
Package package::CRSpackage
Context course inv:
        Course->exists(x/x.coursecode)
Context Course::addcourse (coursecode: string,
matricno:string)
pre:
        Course->exists(x/x.coursecode)
    Student->exists(x/x.matricno)
                                    - - only a course
        previously failed or not previously graded can be
        registered
    Student.failed_courses->exists(x/x.Course.coursecode) or
    not(Student.passed_courses-
>exists(x/x.Course.coursecode))
        Date <= Closing_registration_date
                                                    - -
registration can only be done before closing date
    (Student.totalcredit+ Course.courseunit) <=
Student.maximum_credit
                        - - maximum total course unit for a
                semester must not be exceeded
post:
        Student.totalcredit <= Student.maximum_credit
Context
Course::deletecourse(coursecode:string,matricno:string)
pre:
        Course->exists(x/x.coursecode)
```

```
        Student->exists(x/x.matricno)
        Date <= Closing_registration_date
        Student.totalcredit > 0
post:
        Student.totalcredit < Student.totalcredit@pre
Context
Course::enterscores(coursecode:string,staffid:string)
pre:
        self->exists(x/x.coursecode)
        Lecturer->exists(x/x.staffid)
      Student->forall(x/exists(x.matricno))
        Date <= Closing_registration_date
Context Student inv:
        self->exists(x/x.matricno)
Context Student::Register(): bool
pre:
        self->exists(x/x.matricno)

Account_record::financial_goodstanding(matricno) = True
post:
        True
Context Student::studentlist()
pre:
        Student->forall(x/exists(x.matricno))
Context Account_record inv:
        Student->exists(x/x.matricno)
Context
Account_record::financial_goodstanding(matricno)
 pre:
    Student->exists(x/x.matricno)
Context
Account_record::financial_statement(matricno,session)
 pre:
    Student->exists(x/x.matricno)
    Session->notEmpty
 Context Account_record::debtorslist()
 pre:
    Student->exists(x/x.matricno)

      Account_record::financial_goodstanding(matricno) =
False

endpackage
```

4. Post Requirement Modelling Experience

The decision to place adequate emphasis on requirement elicitation facilitated a number of benefits that were noticed during the later stages of development, particularly the design and the implementation phases of the CRS project. These we have summarized as follows:

- **Completeness of Requirements**: After the initial fact finding initiative which included interviews and interaction with key operators of the registration process. A use case model was constructed consisting of use cases and use case narratives by the developer team. This then formed the basis for interaction during a 2 day requirements workshop between the developer team and user representatives. The user representatives included head of departments, course advisers, data entry operators, student representatives and database administrators. The use case scenarios and use case narratives were given to each participant to carefully study from the first day of the workshop till the final day of the workshop. This result of this was that the user were able relate well with the use case model provided such that they asked questions and made contributions that remarkably furthered the quality and clarity of many issues that had been vaguely attended by the developer team.
- **Improved mutual understanding:** The use case model presented a view of the system that was easily comprehensible to the prospective users of the system, and this provided a platform for healthy interaction and mutual understanding of the expectations and the limits of the CRS.
- **Handling of volatile requirements:** The users also had the opportunity to make changes to some initially conceived requirements during requirement review and validation sessions of the requirements workshop. These were premised on emergence of new realities that were occasioned by better understanding of the potentials of the CRS.
- **Efficiency of Implementation:** The CRS was implemented using a rapid application development (RAD) tool named C++ builder 6.0. C++ builder 6.0 enables event

driven programming and has rich full-featured capabilities for window based enterprise software development, integrating well with major database management system platforms such Oracle, SQL Server, Paradox, Interdev, MS Access. Our experience after implementation and acceptance test by the user was that the CRS possesses features that were completely traceable to the requirements model. This connotes that the software completely satisfies its specified requirements which were accurately captured and represented thorough the use case modelling approach adopted for requirements elicitation.

5. CONCLUSION

There is a seemingly prevalent disposition that relegates the importance of requirements elicitation as compared to other activities of requirements engineering such as requirement specification, requirement validation and requirement management. These other activities usually attract more attention during development and are currently being supported by a large number of automated tools unlike what obtains for requirements elicitation.

However, the result of our empirical study with respect to the automation of the Course Registration System (CRS) project of Covenant University gives credence to the fact that when adequate emphasis are placed on requirement elicitation some of the pressing challenges of requirements engineering which naturally limits the success of software development efforts can be alleviated. This also compels us to make a strong case for requirement elicitation as an equally important aspect of requirement engineering.

In our future work, we intend to look into issues of providing more software tools support for requirement elicitation as a response to existing relative dearth of automated tools support.

REFERENCES

1. Brooks, Jr. F., P. (1987) "No Silver Bullets-essence and Accidents of Software Engineering", IEE Computer 20(4) 10-19.
2. Thayer, R.H. and Dorfman, M. (1997): Software Requirement Engineering 2nd Edition, Wiley-IEE Computer Society, pp 145-163.
3. Faulk, S.R. (1997): Software Requirements: A Tutorial, In M. Dorfman and R.H. Thayer, Software Engineering, The IEEE, Inc., pp 128-149.
4. Sommerville, I. (1996): Software Engineering: Addison-Wesley, England, pp 207-285.
5. Davis, A.M (1993): Software Requirements: Objects, Functions and States, Prentice-Hall, Englewood Cliffs, NJ, pp 64.
6. Micheal, G. Christel and Kyo, C. Kang, "Issues in Requirements Elicitation", http://www.sei.cmu.edu./pub/documents/92/reports/pdf/tr12.92.pdf
7. Anthony, Aby (2000): Requirements Engineering. http://www.opencontent.org/openpub
8. Cannolly, T., Begg, C and Strachan, A. (1999), Database System: A Practical Approach to Design, Implementation and Management, 2nd Ed., Reading, M.A. Addison-Wesley.
9. Sawyer, P., Sommerville, I., and Viller, S. (1999), Capturing the Benefit of Requirements Engineering, IEEE Software, 16(2), pp 78-85.
10. Weigers, K. (2000), When Telepathy Won't Do: Requirements Engineering Key Practices.

http://www.processimpact.com/articles/telepathy.html

11. Jacobson, I., M. Christerson, P. Jonson, and G. Overgaard, Object-oriented Software Engineering: a Use Case Driven Approach. 1992, Reading, MA: Addison-Wesley.
12. Jacobson, I., M. Griss, and P. Jonson, Software Reuse: Architecture, Process and Organization for Business Success. 1997, Reading, MA: Addison-Wesley.
13. Krutchen, P., Architectural Blueprints - The "4+1" View Model of Software Architecture, in IEEE Software. 1995. p. 42-50.
14. Gomaa, H., Designing Concurrent, Distributed and Real-Time Applications with UML. 2000: Addison-Wesley.
15. OMG, UML 2.0 OCL Specification, in OMG Final Adopted Specification ptc/03-10-14. 2004, Object Management Group.
16. RUP : OMG Group, http://www.128.ibm.com/developerworks/rational/library/content/03july/10001251/1251_bestpratices.TP026B.pdf.
17. Gomaa, H. and Olimpiew, E.M. :The Role of Use Cases in Requirements and Analysis Modeling, http://www.ie.inf.uc3m.es/wuscam-05/5-wusCam.pdf
18. Ralph R. Young (2002): "Recommended Requirements Gathering Practices" htttp://www.stsc.hill.af.mil/crosstalk/apr2002.htm
19. Xiaoqing B., Peng, L.C., Huaizhong L (2004): An Approach to Generate the Thin-Threads from the UML Diagrams. COMPSAC pp. 546-552
20. Ibiyemi, T.S., Olugbara, O.O., Ikhu-Omoregbe, N.A. and Osamor, V. (2003), Developing Web-Based Enterprise Application for Effective College Administration, CST Project, Covenant University