



Developing a New Use-Case Approach to Improve Analysis of SRS for a System

Soumen Biswas, Nirmalya Mukhopadhyay
Dept. of CSE, B.A.C.E.T.,
Jamshedpur, India

Abstract: System Requirement Specification is the first, foremost & bare important aspect for Software Development Life Cycle (often called SDLC). IEEE has set up the standards for writing system requirement specification (known as SRS or SyRS). Developing a SRS includes the identification, organization, presentation, and modification of the requirements. IEEE also addressed the conditions for incorporating operational concepts, design constraints and design configuration requirements into the specification. In IEEE Standard 1233, 1998 Edition, certain new formats have been designed for writing SRS for a system, where Use-Case diagrams are being used. Use-Case diagram is the most significant modelling tool in UML. In this paper we have presented a new proposal of constructing Use-Case in a different way to enhance the documentation of SRS. We hope this proposed new construction will improve the system by making it synchronized, indexed and user friendly.

Keywords: SDLC, SRS, Use-Case, UML, Functional Requirements

I. INTRODUCTION

The SRS document described in IEEE Std.12331998 is divided into a number of recommended sections to ensure that information relevant to stakeholders is captured. This specification document serves as a reference point during the development process and captures requirements that need to be met by the software product. Basic issues addressed in the SRS include functionality, external interfaces, performance requirements, attributes and design constraints. It also serves as a contract between the supplier and customer with respect to what the final product would provide and help achieve. Although the IEEE Std. 12331998 specifies the structure it does not choose one representation for requirements over the other. Neither does it specify what techniques should be used to populate the various sections. The Use-Case approach has become the defacto standard for capturing functional requirements. Many of the sections of the SRS document contain information that would be otherwise collected in UML Use-Case artifacts. A significant amount of effort could be spared if the description of functionality captured in these Use-Case artifacts is used to populate relevant SRS sections. For large projects, the number of Use-Cases and the amount of related documentation could quickly become unwieldy without the presence of an organization scheme. It is possible to systematically create and populate several of the SRS document sections if Use-Cases are documented using appropriate organization schemes. The advantage of systematic translation is avoiding duplicative specification efforts. After all, if time and effort have been expended creating the Use-Case artifacts, it makes sense to reuse the results of those efforts when writing the SRS document. It would also lessen the possibility of introducing inconsistencies that arise during duplication. Presently, there are no concrete techniques to identify and link Use-Cases to sections of the SRS. This process is at best ad-hoc, which generates inconsistencies in the final specification document. In this paper we show a systematic way to leverage through our modified Use-Cases to populate the SRS document. We do this with the help of various schemes for managing and organizing the Use-Cases, and by linking specific Use-Case types to related SRS sections. Our method provides additional support to analysts in preparing a standards compliant SRS document by avoiding redundant specification effort and through reduction in the cognitive load. We demonstrate how this taxonomy is used to develop a standards compliant SRS document with the help of case study.

II. SYSTEM REQUIREMENT SPECIFICATION

A software requirements specification describes the essential behavior of a software product from a user's point of view. Success in software requirements, and hence success in software development, depends on getting the voice of the customer as close as possible to the ear of the developer (Wiegers, 1999).

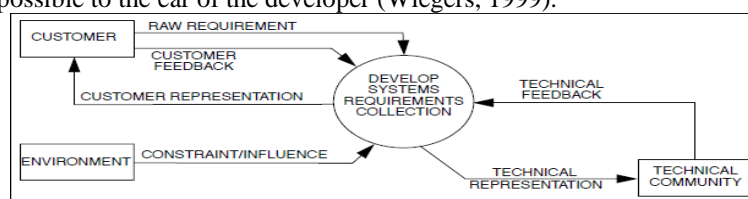


Fig. 1 Context for developing an SRS

The purposes of the SRS are to:

- **Establish the basis for agreement between the customers and the suppliers on what the software product is to do.** The complete description of the functions to be performed by the software specified in the SRS will assist the potential user to determine if the software specified meets their needs or how the software must be modified to meet their needs
- **Provide a basis for developing the software design.** The SRS is the most important document of reference in developing a design
- **Reduce the development effort.** The preparation of the SRS forces the various concerned groups in the customer's organisation to thoroughly consider all of the requirements before design work begins. A complete and correct SRS reduces effort wasted on redesign, recoding and retesting. Careful review of the requirements in the SRS can reveal omissions, misunderstandings and inconsistencies early in the development cycle when these problems are easier to correct.
- **Provide a basis for estimating costs and schedules.** The description of the product to be developed as given in the SRS is a realistic basis for estimating project costs and can be used to obtain approval for bids or price estimates
- **Provide a baseline for validation and verification.** Organisations can develop their test documentation much more productively from a good SRS. As a part of the development contract, the SRS provides a baseline against which compliance can be measured
- **Facilitate transfer.** The SRS makes it easier to transfer the software product to new users or new machines. Customers thus find it easier to transfer the software to other parts of their organisation and suppliers find it easier to transfer it to new customers
- **Serve as a basis for enhancement.** Because the SRS discusses the product but not the project that developed it, the SRS serves as a basis for later enhancement of the finished product. The SRS may need to be altered, but it does provide a foundation for continued product evaluation.

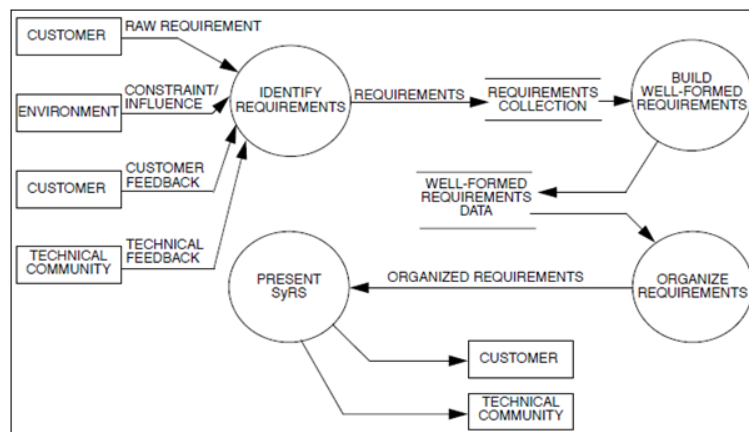


Fig. 2 SRS Development Process

III. ROLE OF SRS IN SDLC

SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

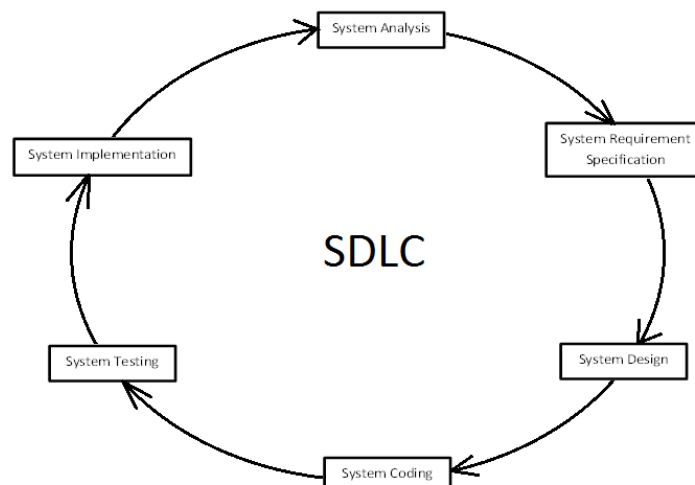


Fig. 3 Stages of Software Development Life Cycle

Stage 1: System Analysis: System (requirement) analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveyors and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational, and technical areas.

Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

Stage 2: System Requirements Specification: Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through 'SRS' – Software Requirement Specification document which consists of all the product requirements to be designed and developed during the project life cycle.

Stage 3: System Design: SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification. This DDS is reviewed by all the important stakeholders and based on various parameters as risk. Assessment, product robustness, design modularity, budget and time constraints, the best design approach is selected for the product.

A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the manifesto of the details in DDS.

Stage 4: System Coding: In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Developers have to follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers etc are used to generate the code. Different high level programming languages such as C, C++, Pascal, Java, and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

Stage 5: System Testing: This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However this stage of the product where products defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

Stage 6: System Implementation: Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometime product deployment happens in stages as per the organizations' business strategy. The product may first be released in a limited segment and tested in the real business environment (UAT- User Acceptance Testing).

Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.

IV. USE-CASE: A MODELLING TOOL

In 1986 Ivar Jacobson first formulated textual, structural and visual modeling techniques for specifying Use-Cases. In 1992 his co-authored book helped to popularize the technique for capturing functional requirements, especially in software development. Originally he used the terms *usage scenarios* and *usage case* – the latter being a direct translation of his Swedish term "*användningsfall*" – but found that neither of these terms sounded natural in English, and eventually he settled on *Use-Case*. Since then, others have contributed to improving this technique, notably including Alistair Cockburn, Larry Constantine, Dean Leffingwell, Kurt Bittner and Gunnar Overgaard. In 2011, Ivar Jacobson published an update to Use-Cases called Use-Case 2.0, the intention being to incorporate many of his practical experiences of applying Use-Cases since its original inception.

The Use-case modelling is a hallmark of the Unified Modelling Language (Rumbaugh, Jacobson et al., 1999) referred to as UML. A *Use-Case* is a *specification of sequences of actions, including variant sequences and error sequences, that a system, subsystem, or class can perform by interacting with outside actors* (Rumbaugh, Jacobson et al., 1999).

A Use-Case describes the interactions between one or more Actors and the system, in order to provide an observable result of value for the initiating actor. The Use-Cases partition the system behavior into transaction, such that each transaction performs some useful action from the user's point view. Each transaction may involve either a single message or multiple message exchange between the user and the system to complete itself.

The Use-Case model represents a function or process model of a system. The functionality of a system is defined by different Use-Cases, each of which represents a specific goal (to obtain the observable result of value) for a particular actor. A *Use-Case diagram* is a visual representation of the relationships between actors and Use-Cases together that documents the system's intended behaviour. An Admission Use-Case diagram is shown below

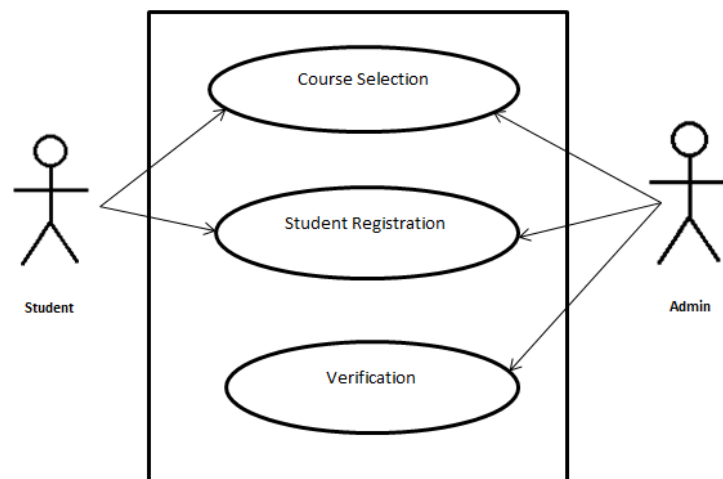


Fig. 4 Student Admission Use-Case Diagram

Arrows and lines are drawn between actors and Use-Cases and between Use-Cases to show their relationships. The default relationship between an actor and a Use-Case is the «communication» relationship, denoted by a line with a small circle. For example, the actor (student) communicating with the Use-Case (Course selection).

A Use-Case describes the interactions between the actor(s) and the system in the form of a dialog between the actor(s) and the system, structured as follows:

1. The actor <<does something>>
2. The system <<does something in response>>
3. The actor <<does something else>>
4. The system <<does something else in response>>

Each dialog of this form is called a “Flow of Events”. Each Use-Case will contain several flows, including one “Basic Flow of Events” and several “Alternative Flows”.

The Basic Flow of Events specifies the interactions between the actor(s) and the system for the ideal case, where everything goes as planned, and the actor’s goal (the observable result of value) is met. The basic flow represents the main capability provided by the system for this Use-Case. Alternative Flows specify alternative interactions associated with the same goal.

A. *Flow of Events – Structure:*

The two main parts of the flow of events are **basic flow of events** and **alternative flows of events**. The basic flow of events should cover what "normally" happens when the Use-Case is performed. The alternative flows of events cover behavior of optional or exceptional character in relation to the normal behavior, and also variations of the normal behavior. The straight arrow in the following figure represents the basic flow of events, and the curves represent alternative paths in relation to the normal. Some alternative paths return to the basic flow of events, whereas others end the Use-Case.

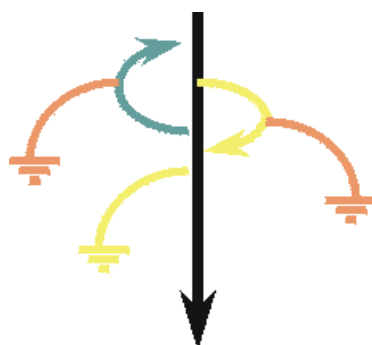


Fig. 5 Typical Structure of a Use-Case Flow of Events

To clarify where an alternative flow of events fits in the structure, it is very essential to describe the following for each detour to the basic flow of events:

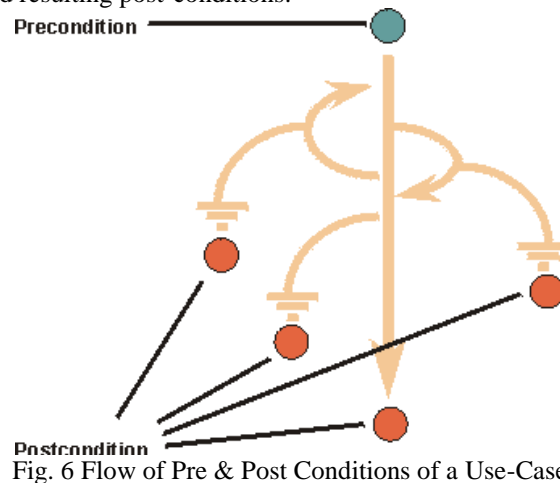
- Where the alternative flow can be inserted in the basic flow of events;
- The condition that needs to be fulfilled for the alternative behaviour to start;
- How and where the basic flow of events is resumed, or how the Use-Case ends.

It might be tempting, if the alternative flow of events is very simple, to just describe it in the basic flow of events section (using some informal "if-then-else" construct). This should be avoided. Too many alternatives will make the

normal behavior difficult to see. Also, including alternative paths in the basic flow of events section will make the text more pseudo-code like and harder to read.

B. Preconditions and Post-conditions:

A precondition is the state of the system and its surroundings that are required before the Use-Case can be started. Post-Conditions are the states the system can be in after the Use-Case has ended. It can be helpful to use the concepts of precondition and post-condition to clarify how the flow of events starts and ends. The following figure shows an illustration of preconditions and resulting post-conditions.



Consider the above **A precondition for the Use-Case Student admission process:** The student selects a degree course and enters the Registration Form.

A post-condition for the Use-Case Cash Withdrawal in the ATM machine: After successful submission of the registration form student, receives a system generated registration number.

V. ROLE OF USE-CASE IN WRITING SRS

A Use-Case typically represents a major piece of functionality that is complete from beginning to end and captures a contract between the stakeholders of a system about its behaviour (Cockburn, 2000). The Use-Case model is an interpretation of the SRS. For ease of documentation, the Use-Case model along with the supplementary specifications document is used as the formal documentation for the project in any software industry. This may seem like an efficient system but it cannot be substituted for a formal SRS. The need for an SRS document is usually mandated by the industry standard. Under such circumstances, when an SRS standards document is unavailable, the Use-Case model is dissected and the Use-Case descriptions cannibalized in an attempt to populate the SRS. The functional requirements of any system can be converted in to modelling language by preparing Use-Cases for the required system. Changes in functional requirements in the specification document need to be reflected in the Use-Case model and vice versa. We should also point out that the Use-Case model is an abstraction of the system model. It does not capture all the relevant aspects of the system, especially non-functional requirements, which are required for completing the product documentation. An unstructured process for using Use-Cases to populate an SRS is inefficient and lacks traceability. The SRS forms the basis for testing plans at a later stage, further boosting its importance in software development process. So to prepare an effective Use-Case, first and foremost task is to understand the functional requirement of the system and create a Use-Case from collected requirements. At the same time we can over come from various functional redundancies of the system and thus Use-Case is a good modelling language.

VI. TRADITIONAL METHOD OF WRITING SRS

The SRS document usually contains all the user requirements in an informal form. Among all the documents produced during a software development life cycle, writing the SRS document is probably the toughest. An SRS document should clearly document the following aspect of a system:

- **Functional requirements:**
The Functional requirements should discuss the functionalities required from the system. The functional requirements of the system as documented in the SRS document should clearly describe each function which the system would support along with corresponding input and output data set.
- **Nonfunctional requirements:**
The nonfunctional requirements deal with the characteristics of the system that cannot be expressed as functions. Examples of nonfunctional requirements include aspects concerning maintainability, portability and usability. It also includes other aspect like reliability, accuracy of result, human-computer interface issues.
- **Goals of implementation:**
The goal of implementation section contain document issues such as revision to the system functionalities that may be required in the future, new devices to be supported in the future and reusability issues etc.

In this section we are concentrating on function requirements of a system. To discuss the functionalities of a system we are taking an example of “student admission process” for a university.



Fig. 7 Student Admission Process

A. SRS for Student Admission Process

R1: Student Admission

Description: The admission process function determines the type degree course for which students want to take the admission. The student selects the course type and department. After successful selection of degree and course type system displays a registration form. A student has to fill the form. After successful submission of the form, system generates a unique Registration number for every student. Later student can communicate to college through this Registration number.

R1.1 Select Department and Degree

Input: System Prompts for Department (populated from list box) and Degree (populated from list box). Student can select a course type for admission.

Output: System checks the valid combination from selection list and displays a Registration form.

R1.2 Fill the registration Form

Input: After successful course selection system displays a Registration form. Students have to fill the registration form and clicks on submit button.

Output: after successful submission of the registration form system validates the form and generates a registration number.

R.1.3 Get Registration Number

Input: Students gets a new registration number.

Output: For any update information, student can communicate to the University through this registration number.

VII. TRADITIONAL METHOD OF WRITING USE-CASE

As per IEEE standard 1233, 1998 Edition the traditional method of writing Use-Case is being followed by us to construct a “Student Admission Process”. The method is as follows:

A. Use-Case Specification: Student Admission

1. Use-Case Name

Student Admission

1.1 Brief Description

This Use-Case allows the student personnel to get registered for admission process in a degree course. Based on student’s information, the Administration personnel will allow the students for Registration and the system displays necessary information. Student Admission process handles 2 functionalities-

- i. Course Selection
- ii. Student Registration

2. Actors

2.1 Primary Actor

2.1.1 Student, Administrator

2.2 Supporting Actors

2.2.1 Registration Module.

3. Flow of Events

3.1 Basic Flow

The system window displays the list of buttons like Home, Department, Admission, Logout, About us. Each button contains various functionalities. User clicks on admission button to avail the admission process.

The system prompts for the type of course selection (Degree/Department). After successful selection of a degree course, system prompts for registration process. The flow continues as in 3.1.1 and 3.1.2 and respectively depending on the user input.

3.1.1 Course Selection

3.1.1.1 The System window displays brief information about the admission process and how to apply for a course.

3.1.1.2 The system prompt for two list box regarding course related information like Department (data populated from list box) and Degree (data populated from list box). The student personnel checks/un-checks any combination of the search option, populated from the list box. Searching will not be initiated unless at least one search option is checked from system populated list box. A student can select any department and Degree based on his/her interest.

3.1.1.3 After successful selection of a degree course and department, student clicks on “select” button. Then system will check and verify that, the selected degree course is available for a particular department or not. If matches found then, system will respond with a Registration form.

3.1.1.4 In case the selected degree course is not available for a particular department, then the flow is as 3.2.1.

3.1.2 Student Registration

This Use-Case begins when a user receives a Registration form after successful selection of a degree course for availing admission process.

3.1.2.1 The system displays a Student Registration Form.

3.1.2.2 The system prompts for mandatory information related to the student like Name, Father’s Name, City, State, District, Country, Sex (system populated list box), E-mail, Phone, Occupation, Department, Degree, Date of birth (system populated list box in dd/mm/yyyy format), Upload your photo(browsing button), Board, Year of passing, % of marks.

3.1.2.3 The system also prompts for Reset and Submit button. After successful submission of the registration form, the user clicks on submit button. Then system displays a preview of the student Registration form which has been filled by the student.

3.1.2.4 The student personnel can edit/update the entered information by click on “Reset” button. Finally clicks on “Submit” To save the student data.

3.1.2.4 After successful submission of Registration form, system displays a message box containing “student Registration number” and other necessary information. The Registration number will be generated by the system.

3.1.2.5 Later student can check his/her status by entering the Registration Number.

3.2 Alternative Flows

3.2.1 Invalid Course Selection

3.2.1.1 In case of course selection, system checks suitable combination for degree and course. If matches not found, then displays a message “select another Degree Course”.

4. Preconditions

4.1 <Successful course selection>

This Use-Case can occur only after successful course selection by the student personnel.

4.2 <Registration>

After successful course selection student can avail the Registration process.

5. Post Conditions

After successful registration student receives a system generated registration number.

VIII. MODIFIED WAY OF WRITING USE-CASE

Without violating the IEEE standard we have designed the following proforma for writing a Use-Case for the same “Student Admission Process”. The process has been described below:

A. *Use-Case Specification: Admission*

Functionality Description	<p>This Use-Case allows the student personnel to select a degree course and register for admission process. Based on student’s information the Administration personnel will allow the students for Registration and the system displays necessary information.</p> <p>This Use-Case handle functionalities are-</p> <ol style="list-style-type: none"> 1. Course Selection 2. Student Registration <p>Depending on the user input, the flow continues as in 1.1 and 1.2 and respectively.</p>
----------------------------------	--

1.1 Course Selection					
Description	This Use-Case begins when the student requests for new admission Process.				
Primary Actor	Student Personnel				
Input Required (Mandatory/Optional)					
Field Name	Source	Type/ Unit	Valid Range	Restrictions	Error Messages
Department	Mouse/ Keyboard	Selection	List box populated from master database	mandatory	“Please Select a Department” if not selected
Degree	Mouse/ Keyboard	Selection	List box populated from master database	mandatory	“Please Select a Department” if not selected
Mandatory Fields Department, Degree.					
Processing					
<ol style="list-style-type: none"> 1) The system prompts for different information related to the student (like Department, Degree). The student personnel enter the relevant information as prompted by the system. 2) A student can select any department and Degree based on his/her interest. 3) If the selected degree course is available for a particular department then, system responds with a Registration form. 4) This Registration Form contains various entries related to the qualification and other necessary information of a student, which they have to fill carefully. 5) If the selected degree course is not available for a particular department then system responds with a message “select another Degree Course”. 					
Preconditions	User selects the admission button. Then system prompts for degree (system populated list box) and Department (system populated list box).				
Post conditions	After successful selection of degree course for a department, system displays a Registration Form.				
Exception Path	The attempt may be abandoned at any time.				
1.2 Student Registration					
Description	This Use-Case begins when the student Personnel requests for Registration in a degree course. After successful submission of the registration form, each student personnel will be provided a Registration number, generated by the system. For any updated information given by the institution, a student has to communicate through the Registration number.				
Primary Actor	Student Personnel				
Input Required (Mandatory/Optional)					
Field	Source	Type/ Unit	Valid Range	Restrictions	Error Messages
Name	Keyboard/ Mouse	Character	Up to 40 character	mandatory	Please enter the name
Father’s Name	Keyboard/ Mouse	Character	Up to 40 character	mandatory	Please enter the Father’s name
City	Keyboard/ Mouse	Character	Up to 40 character	mandatory	Please enter the City name
State	Keyboard/ Mouse	Character	Up to 40 character	mandatory	Please enter the state name
District	Keyboard/ Mouse	Character	Up to 40 character	mandatory	Please enter the district name
Country	Keyboard/ Mouse	Character	Up to 40 character	mandatory	Please enter the Country name

Sex	Keyboard/ Mouse	Numeric	List box populated from master data	mandatory	Please select a sex type
E-mail	Keyboard/ Mouse	Character	Up to 40 character	mandatory	Please enter the E-mail address
Phone	Keyboard/ Mouse	Numeric	Up to 40 digit	mandatory	Please enter the phone number
Occupation	Keyboard/ Mouse	Character	Up to 40 character	mandatory	Please enter the occupation
Department	Keyboard/ Mouse	Character	Up to 40 character	mandatory	Kindly choose a Department
Degree	Keyboard/ Mouse	Character	Up to 40 character	mandatory	Kindly choose a Degree
Date of birth	Keyboard/ Mouse	Selection	List box populated from master data	mandatory	Please enter date of birth
Upload your photo	Keyboard/ Mouse	Selection	List box populated from master data	mandatory	Please choose a photograph
Qualification					
Board	Keyboard/ Mouse	Character	Up to 40 character	mandatory	Please enter the Board name
Year of passing	Keyboard/ Mouse	Numeric	Up to 40 digit	mandatory	Enter the year of passing
% of marks	Keyboard/ Mouse	Numeric	Up to 40 digit	mandatory	Please enter the % of mark
Reset	Keyboard/ Mouse	Button		User dependent	
Submit	Keyboard/ Mouse	Button		User dependent	
<p>Mandatory Fields Name, Father's Name, City, State, District, Country, Sex, E-mail, Phone, Occupation, Department, Degree, Date of birth, Upload your photo, Board, Year of passing, % of marks.</p>					
<p>Processing</p> <ol style="list-style-type: none"> 1) The system Displays a Student Registration Form. 2) The system prompts for mandatory information related to the student Registration. Student enters the information and click on Submit button. To edit the previous information, student clicks Reset button. 3) After successful submission of the registration form, the system displays all the information entered by the student. Then system prompts for Reset and Submit button. To update the information user can click on Reset button. To save the data student can click on Submit button. 4) After successful submission of Registration form, each student personnel will be provided a Registration number, generated by the system. 5) The administrator personnel checks all information entered by the student and set a valid or invalid status. 5) Later student can check his/her status by entering the Registration Number. After entering a registration number system verifies the status field and displays a message "valid or invalid registration number". 					
Precondition	System displays a Registration Form.				
Post condition	On successful submission of the Registration Form student receives a system generated Registration Number.				
Exception Path	The attempt may be abandoned at any time.				

IX. COMPARISON BETWEEN TRADITIONAL & PROPOSED METHODS OF WRITING USE-CASE

Our proposed method (with compared to the existing one):

- Include new Input fields like- Field name, Source, Type/Unit, Valid range and Error Message.
- Describes each Use-Case separately and very precisely. Due to its simple structure a programmer can understand the relation between various functionalities of the system.
- Communicates easily between design part and coding part.
- Helps both developers and users to imagine a view of the new system, whereas in the traditional method Use-Cases are not of much use when it comes to defining user interface.
- Allows the developer to easily understand and take necessary steps as & when required.
- Provides brief classification about the input requirements irrespective of the design of the system.
- Makes it easy to create a database table view.
- Follows the chain: Use-Case description → Field Description → Mandatory Fields → Processing.

X. EFFECT ON SRS FOR THE NEWLY CONSTRUCTED USE-CASE

Preparing a Use-Case model from the SRS is not new. Our main aim is to construct Use-Case model for a system in a way such that it can be more functional and easily understandable by industry personnel as well as client.

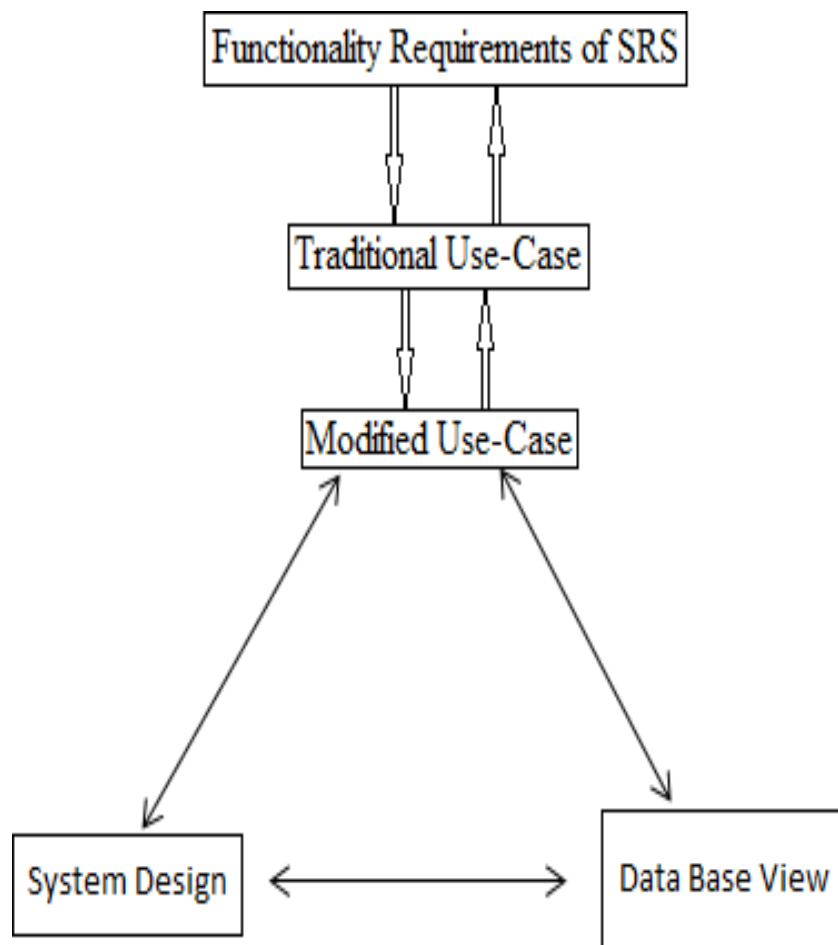


Fig. 8 Proposed Model of Our Construct

If we expand our proposed model using this above diagram, then the flow goes like

- Collect functional requirements from the user and write it on simple language which is understandable by the industry personnel.
- To remove redundancies and understand the relationship between various functionalities, developer uses the Use-Case modeling.
- To make the system more transparent, we have used our proposed model which correlates between the system design and database view.
- By using this proposed model, both developers and end users can easily understand and get a brief knowledge the whole system.

To examine & elaborate our proposed model we have taken one example & implemented our newly constructed model through one-to-one mapping technology. The diagram is shown below:

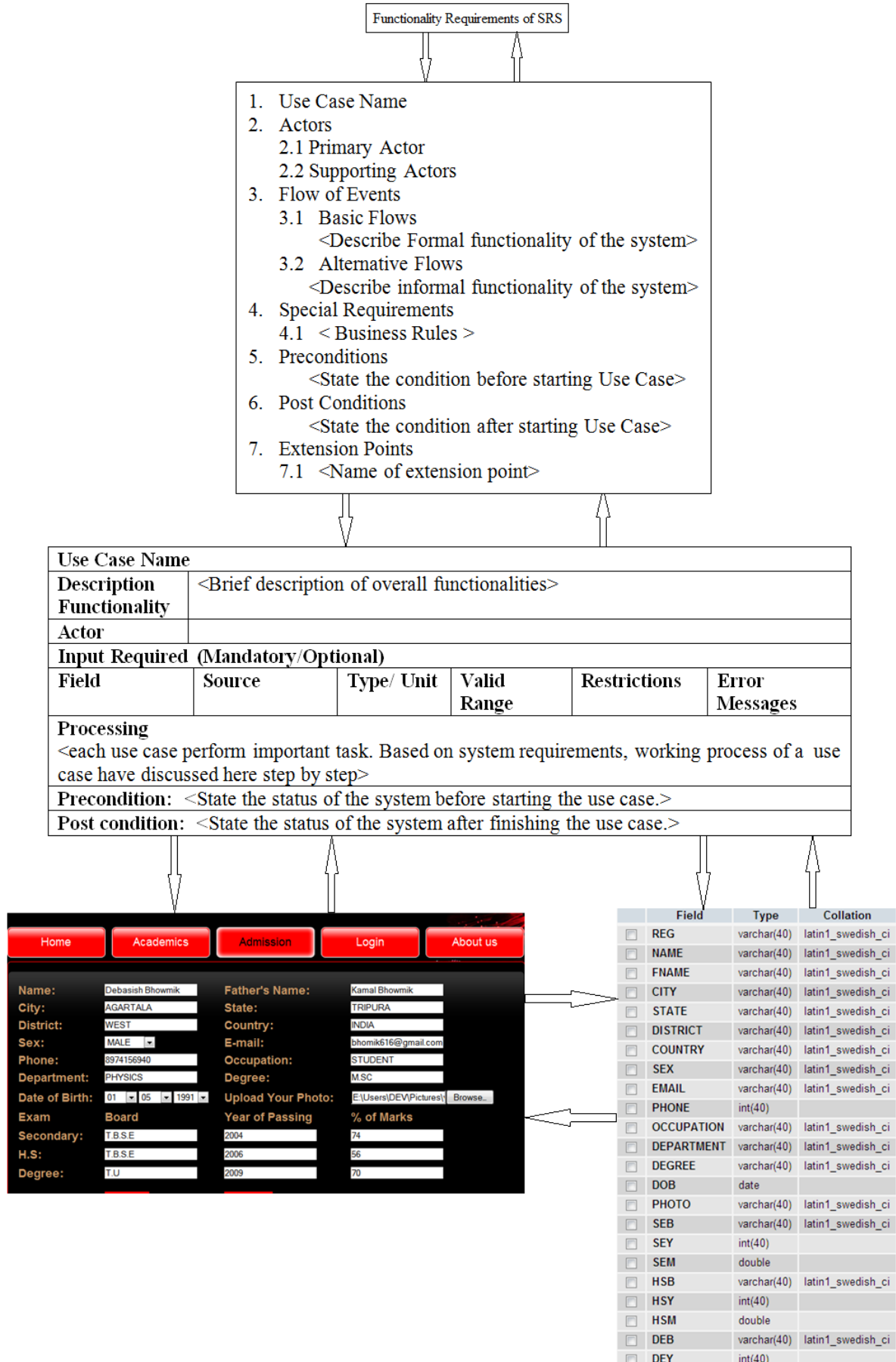


Fig. 9 A Case Study for Implementation of Our New Model

XI. CONCLUSION

UML provides a language and notations for identifying, documenting, and communicating system requirements, and among these Use-Case descriptions and diagrams are most frequently used during the requirements definition stage of a project. The SRS document prepared in compliance with the Std.12331998 can ensure unambiguous communication between the users & the developers. The Use-Case model alone cannot serve as the core piece of documentation as it is only an interpretation of the SRS document. But, it can shorten the time required to generate a standards compliant document if existing Use-Case description could be reused in some modified manner. After considering various aspects of System Requirement Specification, we can say that Use-Case diagram gives a better representation of the system. It becomes easier to correlate system design with database view. After all the whole SRS documentation has been enhanced. The new format of constructing Use-Case that we described here provides us an improved user view, developer view and system view of the SDLC. We have maintained all the standards according to the IEEE. We have made a modification on the description part of the Use-Case to make it more understandable by the customer who will use the software. This modification will help the developers to get a better vision of the fact: "What to do?" and they will be able to easily implement "How to do?" factors. We hope that this modified view of the Use-Case will produce much future scope of enhancing the existing methodologies.

Our future work involves comparing the effectiveness of our technique with traditional ad-hoc approaches. This would involve evaluating the completeness of an SRS Std.12331998 document prepared using our method and comparing it with the existing one. The method will then be verified to ascertain the extent cognitive load experienced in preparing the SRS document, which would serve as another evaluation metric.

REFERENCES

- [1] [BP98] R. Banach and M. Poppleton: "Retrenchment: an engineering variation on refinement". In *Proceedings of the 2nd International B Conference*, Lecture Notes in Computer Science, 1393:129–147, 1998.
- [2] [BGHJ98] S. O. Beskenis, D. F. Green, P. V. Hyer, and E. J. Johnson: "Integrated Display System for Low Visibility Landing and Surface Operations". Technical report NASA/CR-1998-208446, 1998.
- [3] [CCMM99] E. Ciapessoni, A. Coen-Porisini, E. Crivelli, D. Mandrioli, P. Mirandola, and A. Morzenti: "From formal models to formally-based methods: an industrial experience". *ACM Transactions On Software Engineering and Methodologies*, 8(1): 79-113, 1999.
- [4] [CHR91] Z. Chaochen, C. A. R. Hoare, and Anders P. Ravn. "A calculus of duration", *Information Processing Letters*, 40(5):269–276, 1991.
- [5] [FMMPRS04]. C. A. Furia, D. Mandrioli, A. Morzenti, M. Pradella, M. Rossi, and P. San Pietro: "Higher-Order TRIO". Technical Report 2004.28, Dipartimento di Elettronica ed Informazione, Politecnico di Milano, 2004.
- [6] [FRSMK] C. A. Furia, M. Rossi, E. Strunk, D. Mandrioli, and J. C. Knight: "Bringing Formal Methods Up To the Requirements Level", in preparation.
- [7] [FRMM06] C. A. Furia, M. Rossi, D. Mandrioli, and A. Morzenti: "Automated compositional proofs for real-time systems". *Theoretical Computer Science*, 2006. (To appear).
- [8] [Gre00] D. F. Green: "Runway Safety Monitor Algorithm for Runway Incursion Detection and Alerting". Technical Report, NASA Langley CR 211416, 2002.
- [9] [GGJZ00] C. A. Gunter, E. L. Gunter, M. A. Jackson, and P. Zave: "A Reference Model for Requirements and Specifications". *IEEE Software* 17(3):37-43, 2000.
- [10] [HJLNR02] J. G. Hall, M. Jackson, R. C. Laney, B. Nuseibeh, and L. Rapanotti: "Relating software requirements and architectures using problem frames". In *Proceedings of the 10th IEEE Joint International Conference on Requirements Engineering*, pp. 137-144. IEEE Computer Society, 2002.
- [11] [HR03] J. G. Hall and L. Rapanotti: "A Reference Model for Requirements Engineering". *Proceedings of the 11th IEEE International Requirements Engineering Conference (RE'03)*, 2003.
- [12] [HRJ05] J. G. Hall, L. Rapanotti, and M. Jackson: "Problem frame semantics for software development". *Journal of Software and Systems Modeling*, 4(2):189-198, 2005.
- [13] [IEEE98] IEEE Recommended Practice for Software Requirements Specifications. IEEE STD-830, 1998.
- [14] [Jac00] M. Jackson: *Problem Frames: Analyzing and Structuring Software Development Problems*. Addison-Wesley, 2000.
- [15] [JHJ06] C. B. Jones, I. Hayes, and M. Jackson: "Specifying Systems that Connect to the Physical World". Technical Report, CS-TR-964, University of Newcastle upon Tyne, 2006.
- [16] [MT01] S. P. Miller and A. C. Tribble: "Extending the Four-Variable Model to Bridge the System-Software Gap". In *Proceedings of the 20th Digital Avionics Systems Conference (DASC'01)*.
- [17] [OMG05] Object Management Group: "UML 2.0 Superstructure Specification". Technical Report, OMG, formal/05-07-04 (2005).
- [18] [ORS92] S. Owre, J. M. Rushby, and N. Shankar: "PVS: A Prototype Verification System". In *Proceedings of CADE-11*, Lecture Notes in Computer Science, 607: 748–752, 1992.
- [19] [OS99] S. Owre, and N. Shankar: "The Formal Semantics of PVS". Technical Report CSL-97-2R. SRI International. March 1999.
- [20] [PM95] D. L. Parnas and J. Madey: "Functional Documents for Computer Systems". *Science of Computer Programming* 25(1):41-61, 1995

- [21] [PRM05] M. Pradella, M. Rossi, and D. Mandrioli: “ArchiTRIO: a UML-compatible language for architectural description and its formal semantics”. In *Proceedings of FORTE’05*, Lecture Notes in Computer Science 3731: 381-395, 2005.
- [22] [Rus89] J. Rushby: “Kernels for Safety?”. In T. Anderson, editor, *Safe and Secure Computing Systems*, chapter 13, pages 210–220. Blackwell Scientific Publications, 1989.
- [23] [ZJ97] P. Zave, and M. A. Jackson: “Four dark corners of requirements engineering”. *ACM Transactions on Software Engineering and Methodology* 6(1):1-30, 1997.