International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 6 (2015) pp. 15141-15163

http://www.ripublication.com

Capturing Architecturally Significant Requirements using Architect's Use Case Diagram

B. Sathis Kumar[1*] and Ilango Krishnamurthi[2]

[1]School of Computing Science and Engineering, VIT University, Chennai, Tamilnadu, India, sathisphd@gmail. com
[2]Computer Science and Engineering,
Sri Krishna College of Engineering and Technology, Coimbatore, Tamilnadu, India

Abstract

Today'sSoftware Industries' mostpreferred Modeling Language for a Software Development is Unified Modeling Language [UML] [6, 9]. System Analyst uses theUML Use Case Diagram [UCD] for representing functional requirements of the system in Requirement Analysis Stage[1]. The UCD givesonly the summary of the System. The Use Case Specification (UCS) document elaborates the full story of the system[10]. Quality attributes Requirement does not fit into UCD and UCShence it is captured in supplementary documents[10]. Some of the requirements from functional and nonfunctional requirementsinfluence the architecture of the system and these arestatedas Architecturally Significant Requirement [ASR][11]. Use Case diagram provides an effective communication medium for customer interaction. Several interactions between the System Analyst and customer arerequired infinalizing theContract[1]. Customer involvement and gathering the complete requirement are two important factors for any successful software project[1]. Most of the customers actively participate indescribing the functional requirements but they show less interest in describing Quality attributes and other nonfunctional requirements[12]. Omitted important ASR will affect the Quality of the System[12]. Hence improving Customer involvement in the Requirement stage is necessary. The goal of this paper is to analyzeUML Use Case Diagram'sweakness while capturing ASR and intern propose an enhanced UCD to capture ASR in a simple and effective way and we name it as Architect's Use case diagram [AUCD]. AUCD introduces Sub flow Connector, Security flow Connector, Alternative flow connector, Scalability indicator and Performance indicator to specify ASR. A Case study demonstrates the usage of Architect's Use Case Diagram and the results shows AUCD is helpful for capturing ASR during the requirement analysis stage and is also useful for effective communication with customer.

Keywords: Unified Modeling Language (UML), Architecturally Significant Requirement (ASR), Use Case Diagram (UCD), and Architect's Use Case Diagram (AUCD).

1. Introduction

Software architecture (SA) is the basis for any Software Development. Software Architecture defined as "A software system's architecture is the set of principal design decisions made about the systems"[12]. Software Quality depends on good Architecture of the Software System Functionality and quality attributes requirements are important factorsin Architectural design decision[12]"Security, Usability, Reliability, Performance andsupportability are quality attributes of the Software System"[11, 17]. These Architectural design decision requirements are referred as Architecturally Significant Requirements (ASRs)[12]. By definition, "Architecturally significant requirements (ASRs) are those requirements that have significant impact on software architecture"[12]. If collected ASR's are vague, erroneous and inadequate then by applying these ASR in Software Architecture would be incomplete and erroneous software[12]. However, in practice, during the requirement analysis stage Stakeholders not able to express ASR properly to System Analyst due to lack of clarity of the required requirement[12]. Usually Customers are familiar with their domain requirements only and not with most of the ASR[17]. Most of the customers show less interest to fill the Architectural Requirements Questionnaire because they feel that ASR is too technical[6]. Lot of methodologies are available for gathering domain specific requirements but only few techniques are there for capturing architectural requirementsand these techniques also do not have proper expressive power particularly in the areas of describing quality requirements for different stakeholders[12]. UML is a leading Modeling language in software industry to model the software System. UML 2. 4 introduces14 diagrams and each diagram focus on different views[14, 15]. These views are required to satisfy the various needs of the Requirement Analyst, Software Architect, Designer, Coder and Tester. System Analyst is responsible for producing Use Case Diagram; Use Case Specification and Supplementary documents in Requirement elicitation stage and these are inputs to Analysis, Design, Implementation and Testing Stage. Use Case Diagram plays an effective role in specifying the functional requirements and it gives only an abstract idea about the requirement. The detailed functional requirement specified in Use Case Specification and other missed out requirements particularly the nonfunctional requirements are specified in the SupplementaryDocumentation[10]. Using these resources the architect and designer will design the architectural model that contains Deployment, component and class design diagram[1]. The main problem in Use based Analysis is Architecturally Significant Requirements are scattered in various documents like Use Case Diagram, Use Case Specification and Supplementary documents[10]. Referring too many documents to find the ASR and finalize the design decision for Architecture is a complex task for any Software Architect. Most of the Architecturally Significant Requirements are described vaguely or embedded with other requirementsin Supplementary documents written in natural language[12]. This

vagueness and embedding style of ASR specification leads to wrong assumption of the requirements and this indeed leads to wrong choice of the Architectural style and design[12]. For example if the system requires efficient usability then Architect will introduce Event Driven Style so that it can handle any type of exception. Hence, someof the architectural significant requirements need the clear view of the customer to take final decision. Usually the customer is a non-technical person and most of the customers can understand the use case diagram only. But existing Use Case Diagram does not have the provision to represent architecturally significant requirements[1]. The proposed AUCD introduces Sub flow Connector, Security flow Connector, Alternative flow connector, size indicator and Performance indicatorwhich are helpful to specify ASR in Use Case Diagram. The proposed method is very simple so it improves the active participation of the customer. Important design decision is represented in one diagram and hence it is easy for Software Architect to take a quick decision to select the Architecture and also validate the Architecture. The rest of thesections are:Use Case Driven Requirement analysis and Misuse Case method is presented in Section 2and theproposed Architect's Use Case Diagram (AUCD)is discussed in Section 3. In section 4 ATM Case Study Model designed using AUCD is presented. In Section 5 Validation of Proposed AUCD is discussed. In Section 6 Conclusion and future enhancement of the proposed system is presented.

2. Related work

Most of Software Architect uses UML notation for analyzing and designing the Softwarearchitecture of the system. UML Use Case based Requirement Analysis is the most popular method used in the Software Industry. Howeverthis method lacks systematic specification of ASR which is an important drawback. The following section describes brief introduction of Use case driven analysis, Misuse Case Method and limitation of these methods.

2.1 Use Case Driven Requirement Analysis

Use Case Diagram, Use Case Specification and Supplementary Documents are important artifacts for Use Case Driven Requirement Analysis and these artifacts are input to the Analysis and Design Phase. Each artifacts focus on different requirement information in which abstract functional information are in Use Case Diagram, detailed scenario of the functional requirement are described in Use Case Specification and quality requirements are in Supplementary Documents. Using these artifacts Software Architect extractsthe decision requirements and finalizes the software Architecture. The following section describe Use Case Driven Requirement Analysis artifacts briefly

2.1.2. UML Use Case Diagram

"A use case diagram is a diagram that shows a set of use cases and actors and their relationships"[1]. "The use case diagram is depicts a static view of the system functions and their static relationships with external entities and with each other"[9]. "Stick figures represent the actors, and ellipses represent the use cases"[1]. Use case is a group

of associated scenario to achieve a specific goal and Actors are who initiate the system. Actors are human or othersystem[1].

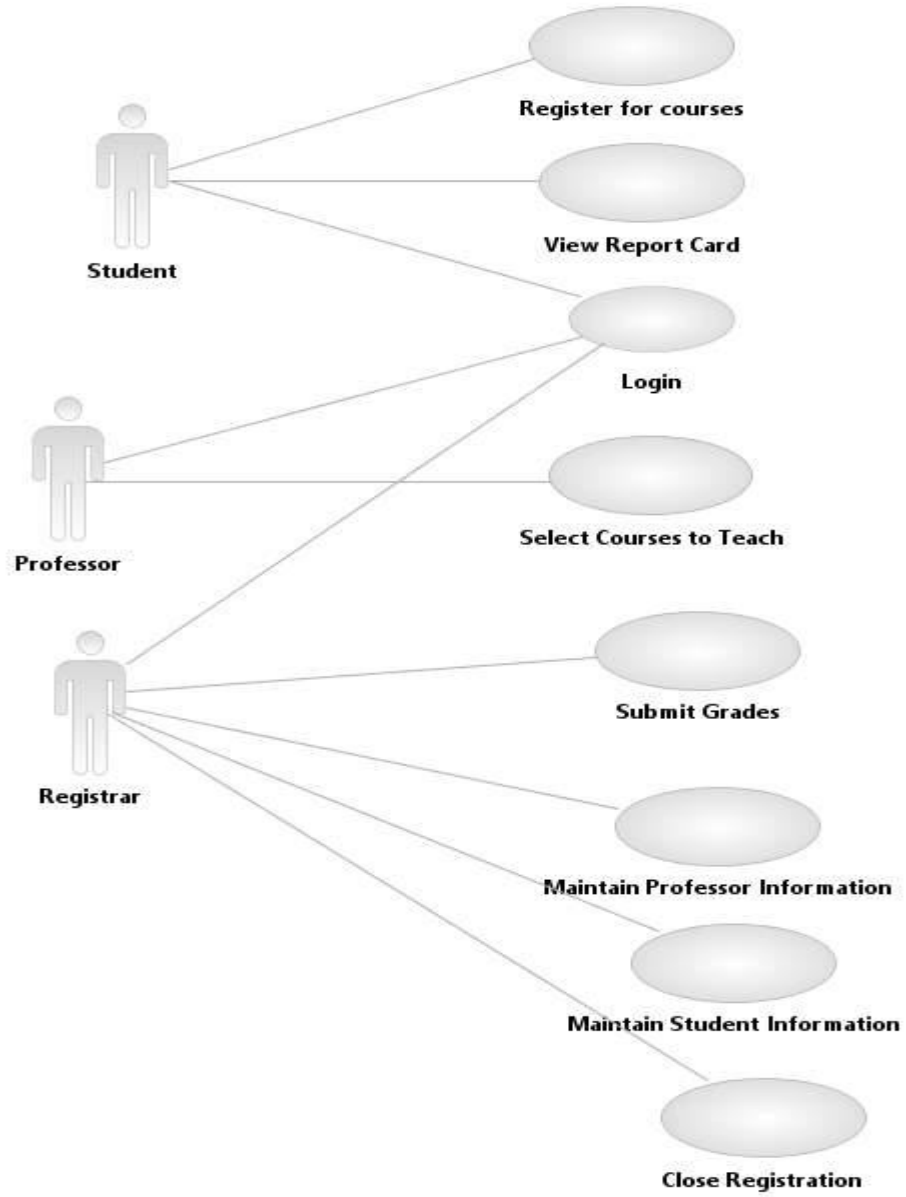

Figure 1. Course Registration System Use Case Diagram

For example, Figure 1 shows the Course Registration System. Student, Professor and Registrar are actors. These actors are interacting with the Course Registrations System. The requirement of these actors is modeled by the use cases. The Student can "Login" into the system and "Register the courses". At the end of the Semester student can "View Grades" from the system. Similarly Professors can

"Login" into the system and "Select Course to Teach". After the evaluation, Professor can "Submit the Grades" into the system. Registrar takes responsible for "Maintain Professor Information", "Maintain Student Information" and "Close Registration". "Maintain Professor Information", "Maintain Student Information" "Close Registration Login", "Register the Courses", "View Grades", "Select Course to Teach" and "Submit the Grades" are use cases.

2.1.2 Defining Relationships between Use Cases

The relationship between the use cases is specified by using generalization, include and extend relationships[9]. "An include relationship between use cases means that the base use case explicitly incorporates the behavior of another use case at a location specified in the base. Include relationship is used to avoid describing the same flow of events several times"[9] and also to specify base use case should compulsory call the included use case before it completes the steps[5]. For example, Figure 2 shows Register for courses use case allows student to select the course to study. Select Courses to teach use case shows Professor to choose thesubject to teach. To complete the registration both use casesneeds to verify the availability of the course. So, common functionality "check availability" can be extracted as separate use case andusing include relationship these use case can be communicate with each other. Here, the include relationship shows compulsoryinvocation of included use case and how to handle the reuse of the same functionality.

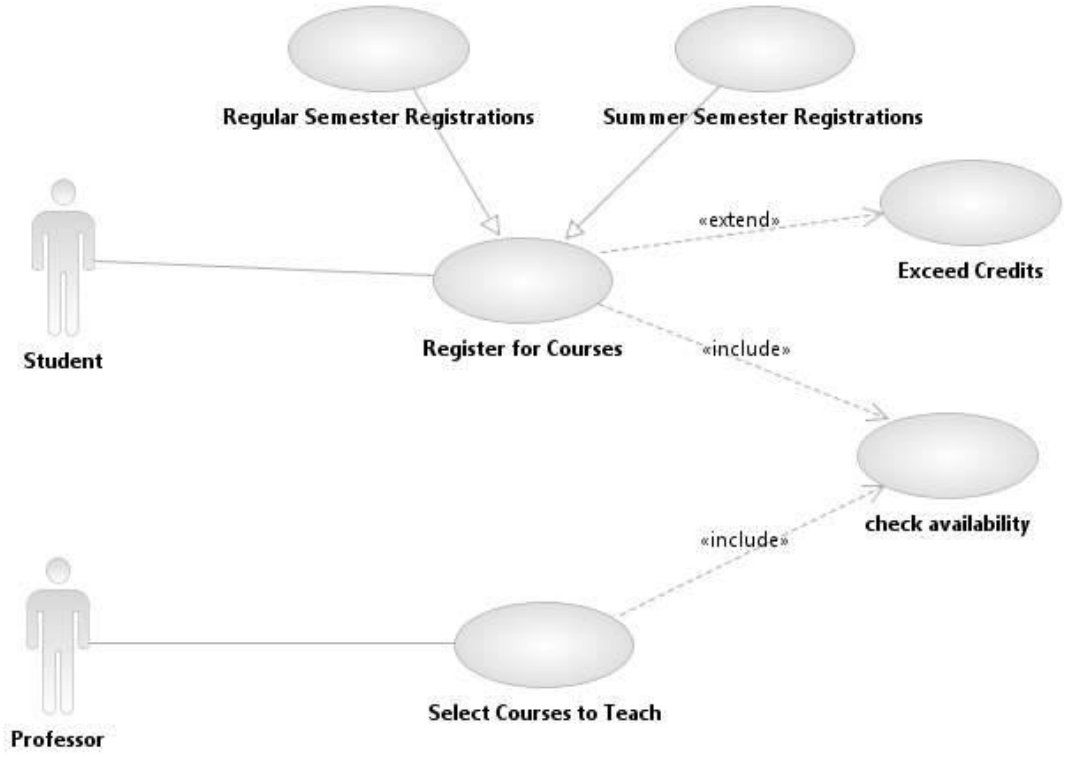

Figure 2. Use Case Relationships

The generalization association is to specify relationship between parent and child use case. Parent use case includes all possible common behavior and the child use cases inherit the parent behavior and some additional behavior. Figure 2 shows usage of the generalization in the Course Registration System, the Register for Course is a parent use case. Regular semester use case registration and Summer Semester use cases are child use cases. Here Child use cases are similar to parent course registration use case but Regular semester and Summer Semester child use case registrations have different constraints and working principles.

Extend relationships can be used to represent optional or exceptional system behavior. Figure 2 shows how extend relationship are used to highlight an exception case. Usually students are allowed to register aMaximum of 27 credits per semester, but insome exceptional casesRegistrar may permit some students to exceed 27 credits.

2.1.3 Use Case Specification and Supplementary Documents

The Use-Case Specification document explains the complete scenario of each use case by using alternative and basic flow of events. The Basic flow defines the normal way of achieving the goals of the use case. Alternative flows extend the basic flow to cater to variants and exceptions. Sub flows can be used to make a complex flow of events easier to read. Usually Sub flow describes additional functionality of the system for example In Course Registration System, Create Schedule, Delete Schedule and Update Schedule use cases are sub flows of Transaction use case. Alternate Flow describes that the use case goal is achieved notby usingthe basic sequence of steps for example In Course Registration System Unfulfilled Prerequisites, Course full and Schedule Conflicts are alternative flows. Student selects the course which does not satisfy the prerequisites is an Unfulfilled Prerequisite alternative flow. Selected course schedule is already filled and selected course schedule classes with existing course are other possible alternative flows.

Supplementary specification specifies Quality attribute requirements of the system. For example the course registration should be complete within 60 seconds is a performance requirement. It also specifies thatlegal, programming language, standards and other requirements do not fit into the use case specification.

2.2 Limitation of Capturing Architecturally Significant Requirements using Use Case Driven Requirement Analysis

Capturing Architecturally Significant Requirements using Use Case Driven Requirement Analysis has the following drawbacks

2.2.1 Scattered information:

Software Architect needs to know complete Functional, Nonfunctional Requirements and Quality Requirements but these information'sare scattered in various documents[6, 10. Hence] it is a tedious job for Software Architect to capture ASR.

2.2.2 Important Architecturally Significant Requirements are specified in natural language:

Functional requirements focus on what the system should do and nonfunctional

requirements focus on how the system should do with respect to what?. Nonfunctional Requirements (NFR) create major impact in Quality of the software and User satisfaction. NFR covers the quality attributes like accuracy, reliability, usability, security and performance. Software Architecture is composed of a collection of design decisions. Use Case Specification and Supplementary documentation areimportantdesign decisions and it is written in natural language. Most of the Use Case and Supplementary specification document describes the requirements vaguely or it neglects the important requirements or therequirements may be hidden in other requirements. This issue will definitely create a serious impact in Architectural Design Decision and these are discussed as follows:

Vague Requirement statement Issues:
If the requirement statement is vague then it will seriously affect the Architect design decision. For example In Course Registration whenever student deletes a course for security reasons student should receive an alert message immediately. This is vague statement because Architect may give an immediate email solution to this issue which is also an acceptable solutionand Client Serveris suitable architectural style. But Customer expectsareal time SMSnotification system; hence herePublish-Subscribe Architectural Styleis appropriate.

Hidden Requirement issues:
Most of the ASR are hidden in other requirements. For example: A In Course Registration whenever student deletes a course for security reasons student should receive an alert message within 60 seconds. Here performance requirement is embedded with functional requirement and it is specified in the Use Case Specification. Hence it is a difficult task to track the hidden requirements.

Customer Active participation issues:
The alternative flow plays important role in capturing the ASR, for example Alternative flow use cases Amounts exceed withdrawal limit, Amount exceeds daily limit, No response from Bank, Money not received discussed in section 2. 1. 3 shows that the system requires high usability, hence the architect will include high exception handling mechanism in the Architecture. Initially System Analyst identifies few alternative and exception flows based on his knowledge and inputs from the customer. Most of the customers show less interest to read the Use case Specification and they consider this document as a technical document. This clearly shows that Use Case Specification using natural language reduces active interaction between customer and system analyst and this in turn will affect the capturing ASR.

Supplementary specification issues:
Most of the project members involved in the software development think that term supplementary influence is not an important document[10]. Success of the project depends on how the supplementary documents are handled and implemented, because major nonfunctional requirements are specified in the supplementary specification

only[10]. For example one of the projects was failed in the acceptance test because the development team completely ignored the supplementary specification and they designed the software to support 10 concurrent users but the customer requirement was to support 500 concurrent users.

2.2.3 Usage of <include> and <extend> relationship in Use Case Diagram is confusing:

The <include>, <extend> and generalization are useful to state relationship among the use cases[9] in the Use Case Diagram. The creator of UML Grady Booch, Ivar Jacobson, and James Rumbaugh recommends "Organizing your use cases by extracting common behavior (through include relationships) and distinguishing variants(through extend relationships) is an important part of creating a simple, balanced, and understandable set of use cases for your system"[9]. According to Fowler most of the teams confuses the usage of include and extend relationship. The reason is Include and Extend relationship definitions are almost same[8]. Because Extend is used when the condition is reached then it would call extended use case and Include is used when the requirement of include use case is reached then it would call included use case[8]. Finding appropriate relationships between use case and its sub flows is challenging. For example Figure 4 shows Register for Courses have 3 possible sub flows. Student can create the schedule is a sub flow andif student have created the schedules already then student are allowed to delete and update the schedule which are the other sub flows.

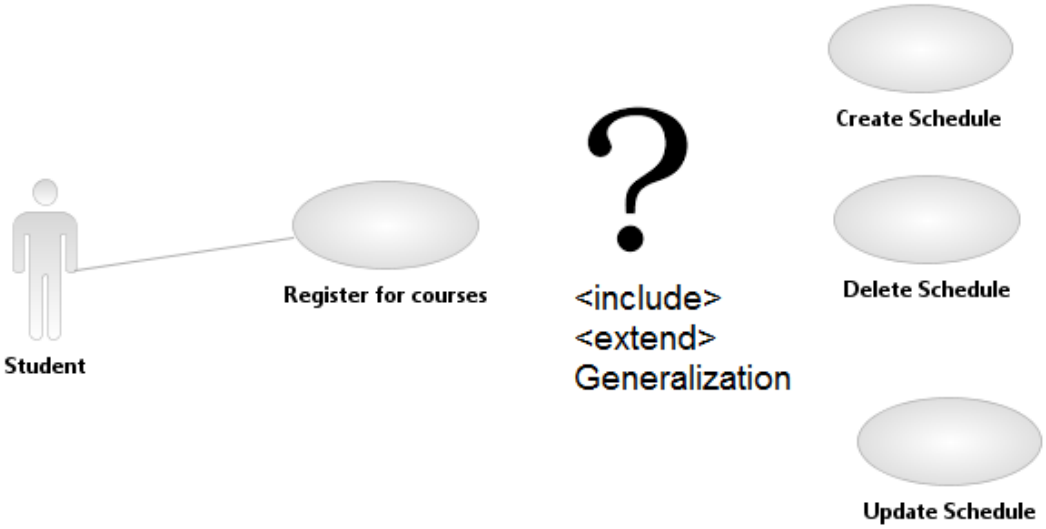

Figure 3. Use Case Relationship Decision

If we use include relationship in Figure 3 then the view changes as follows

i. Create, update and delete schedule are common functionality and itis used by another use cases. But these use cases are only useful for Register for course only.

ii. Whenever register for course is called it is necessary to call create schedule, update schedule and delete schedule. But it is not true for all the cases.

By definition "You may use an extend relationship to model a separate sub flow that is executed only under given conditions"[9]. Hence <extend> relationship is suitable for Figure 4 scenario, Based on the selection of any one option of create, schedule, update then the control will be transferred to appropriate extend use case. But extend relationship is also used to highlight exceptional use case. For example Exceed credit use case in Figure 3 is an exceptional use case applied for special consideration. Registrar will allow a student to exceed the Maximum credits allowed. Sub flow is describing different features or functionality of system and exceptional flow explains about the rare cases.. The usage of extend relationship in sub flow and exception flow will affect the understandability of system because user may wrongly understand all the sub flows are rare cases or all exceptional cases are general sub flows. Functionality is an important ASR. If functionality is not defined clearly it will affect the Architectural design decision hence usage of include and extend affects description of functionality.

2.3 Misuse Case

Use case focus on what function the proposed system should do. Misuse case is reverse of Use Case and it represent what function proposed system should not do[3]. Guttorm Sindre and Andreas Opdahl introduced the Misuse Cases and its objective is to specify security requirement in the use case diagram[3]. Filled oval symbolis used for misusecase and filled stick men is used to specify misusers. Threaten and mitigate relationshipare introduced to specify the relationship between use case and misuse cases. By definition "A threaten relationship from a misuse to a use case indicates that the misuse case exploits one or more vulnerabilities within the use case. A mitigate relationship from a use to a misuse case indicates that the use case prevents, thwarts, detects, or otherwise responds to the misuse case"[13].

Limitations of Misuse Cases:

Misuse case focuses on specifying security requirements in detail. Misuse Case diagram specify all the possible vulnerabilities of the system and Mitigation flow is included in the use case specification apart from Basic flow and alternative flow. The mitigation flow describes steps to handle the vulnerabilities. Hence this clearly shows Misuse Case diagram and Specification is purelya technical document and it can be handled by Architect and Security designer. So, this document cannot be used with customerfor further discussion.

3. Need for improved Use Case Diagram to specify ASR

UML is a leading Modeling language in software industry for software development. Most of In Software Architecture analysis Software Architect collects the ASR fromUse Case Diagram, Use Case specification and other supplementary documentation. This is a tedious job for Software Architect. To improve the quality of the software, recent methodology like Agile recommends interaction between the customers should be high and customer should be the part of the software development team. To understand the requirement and to improvise the system

discussion between the Software Architect and Customer is essential. So a proper graphical representationis required to specify ASR. We identified the Use Case Diagram is suitable for specifying ASR. The reason is the among 25 UML diagrams Use Case Diagram is simple and easy for the customers to understand. The reasons for selecting Use Case Diagram to specify ASR are as follows:

1. UCD is simple and it is easily understood by stake holders
2. UCD helps customer to checkwhether the system fulfills his requirement
3. UCDhelps in overall understanding of system by the Projectmanager, so that monitoring and planning of the project is easy.
4. UCD helps Analysts to describe and document what the proposed system is expected to do
5. UCD helps Developers to understand the system requirement completely and plan for development.
6. UCD helps Testers to understand the requirement and plan for verification

The following section describes how the enhanced used case diagram called AUCD is useful to specify ASR visually.

4. Proposed Architects Use Case Diagram (AUCD)

The Main objective of this research work is Capturing Architecturally Significant Requirements in a systematic method. The proposed Architect's Use Case Diagram [AUCD] is used to capture major design decision requirements visually, hence it is easy to understand by everyone and easy to verify and validate. The proposed method introduces Sub Flow Connector, Alternative Flow Connector, Security Flow Connector, Response time and Size Indicator to set the relationship between the use cases. The reasons for introducing these connectors and usefulness are discussed in detail as follows:

4.1 Sub Flow and Alternative FlowConnectors

Sub flows facilitate the reuse of flows within the same use case. "Sub flows will be written by moving a self-contained, logical group of detailed behaviour to a sub flow, and then referencing this behaviour in summary form within the flow of events"[9]. A sub flow may have alternative flows. Sub flow use case are notincluded in the Use Case Diagram becausesub flow are internal system processing[20] and it is described in the Use Case Specification.

Requirement of Sub flow and Alternative Flowin Use Case Diagram

Including Sub flow use case in the use case diagram will help to identify more ASR of the system. The sub flow may demand some special functional requirement or Quality attributes requirements. The sub flow may have alternative flows and these alternative flows may help to identify ASR. Based on the behaviour of sub flowand alternative flows the Architect may decide appropriate Architecture. For exampleIf Number of alternative flow is high due to usability of the system then good exception handler is suitable, if alternative flowsdemands performance of the system then good parallel

processing style is suitable. Hence including sub flow and alternative flow in use case diagram is mandatory.

Need for Sub flow and Alternative flow Connector

Include, extend and generalization relationships are used to specify how the use cases are connected to complete the task. By using include, extend and generalization relationship it is possible to specify some of Sub flows and Alternative flows. But it is a complex task to identify the appropriate relationship. Extend Relationship is used for specifying optional and exceptional use cases which creates a confusion to the user. In section 2. 2. 3 we discussed in detail the drawback of the extend relationship. Risk in wrong usage of this relationship is high due to the confusing definition of include and extend relationship. The wrong relationship may misinterpret the requirement to the developing team. Most of the authors advice that usage of these relationship should be avoided[7, 13] because it would affect the readability of the system.

The main objective of including sub flow and alternative flow in the use case diagram is to make a customer activelyparticipate in capturing Architecturally Significant Requirements. So usage of technical relationships like include, extend and generalisation will affect readability. Hence simple and understandable relationship should be used in the use case diagram to make the customercomfortable. The sub flow and alternative flow connectors are useful to set a simple relationship. The use cases are initiated in two ways either bythe choice of actor or it is based on internal processing logic of use case. Sub flow connector relationship is introduced to handle actor initiated use cases, generally menu driven choices. Alternative flow connector relationship is introduced to handle internal processing of logic of use case.

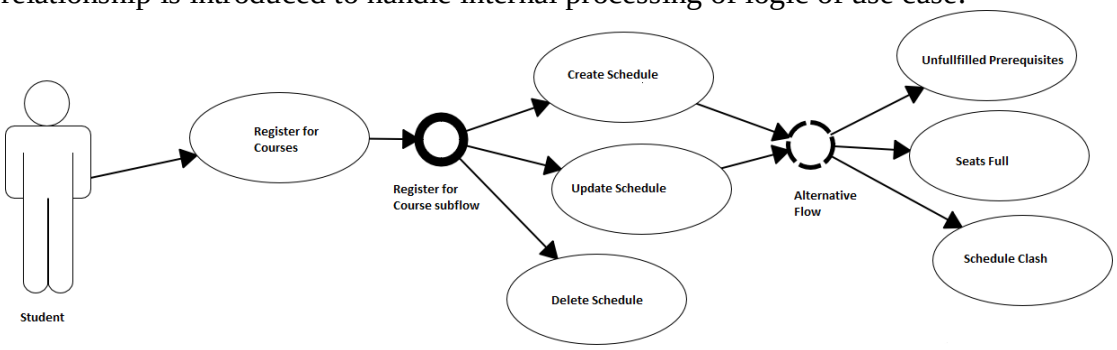

Figure 4. Usage of Sub flow and Alternative flow connector

For example in Figure 4, Register for course use casee gives 3 different options to the user To create, delete and update schedule. Based on the need user can select the options. The relationship between Register for course use case and optional or menu driven use cases arerepresented using subflow connector

Create schedule and update schedule have common alternative flows which are Unfulfilled Prerequisites, Course full and Schedule Conflicts. Unfulfilled Prerequisites, Course Full and Schedule Conflicts use case will be invoked ifa student

selects the course which does not satisfy the prerequisites or selected course schedule is already filled or selected course schedule classes with existing course respectively.

The following section describes the guidelines for usage of Sub flow connector and Alternative flow connectors

The Guidelines for usage of Sub Flow Connector

- Sub flow connector is useful to set the relationship between base use case and sub flow of the base use case
- The solid dark circle symbol is a sub flow connector
- The sub flow connector can be addedto your model when the base use case has menu driven function and it is selected by user. Each function is considered as sub flow of the base use case
- A base use case may have any number of sub flow use case
- Base use case and sub flow use case can use the existing include, extend and generalization relationship if required..

The Guidelines for usage of Alternative Flow Connector

- Alternative flow connector is useful to set the relationship between base use case or sub flow use case with Alternative flow use case
- The dotted circle symbol is an Alternative flow connector
- The Alternative flow connector can be added to your model when the base use case or sub flow use case achieve the use case goal by usingalternate flow use case.
- A base use case or sub flow use case may have any number of alternative flow use case
- Alternative use case can use the existing include, extend and generalization relationship if required.

4.2 Security Flow Connector

Security is one of the important architecturally significant requirements. The security requirements areproperly presented in the earlier stage of software development whichhelps the software architect and designer to take the design decisions easily. Alternate Flow describes that the use case goal is achieved by not using basic sequence of steps. Alternative flow connector is used to represent all the possible alternative flows in the use case. Some alternative flow creates security issues of the system. For example in the course registration System, handling invalid login credentialsis analternative flow use case for login use case. Hacker trying to access the login, using random password is a possible vulnerability. Specifying securityuse case to handle the security vulnerability in alternative flow is an important Architecturally Significant Requirements. If the System demands high security then architect will choose the appropriate architecture to ensure the security aspects. Hence specifying security flow in the use case diagram is important and it also helps active interaction between customer and System Analyst to identify more Security Requirements

The Guidelines for usage of Security Flow Connector

- Security flow connector is useful to set the relationship between base use case or sub flow use case or Alternative flow use case with Security flow use case.
- Double Circle Connector symbol is a security Flow Connector
- The Security flow connector can be added to your model when the base use case or sub flow or Alternative flow use case has a sequence of steps to ensure the security of the system. The sequence of step for ensuring security of the system is called Security flow use case.
- A base use case or sub flow use case may have any number of security flow use case
- Identifying appropriate security solution use case is recommended. If System Analyst does not have solution at the requirement stage then specifying default Security Requirement1, Security Requirement2… etc is also an acceptable security use case. This approach will ensure that security requirement is required in that use case.

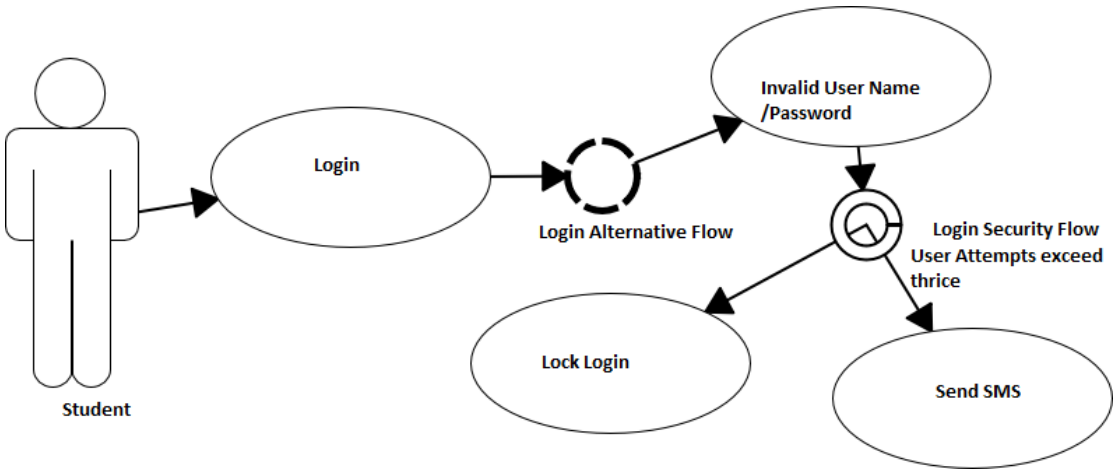

Figure 5 Security flow Connector in AUCD to Specify Security Requirements

Figure 5 explain the usage of security flow connector in handling login vulnerability. The system will allow 3 times to enter wrong credentials. Ifthe user input is wrong in the 4thattempt then security flow connector is invoked which locks the user account and a message is send to user. This will alert the student and help the administrator to do the necessary steps to find the hacker. The handling mechanism described in the use case diagram is very useful to the customer to decide whether the mechanism is suitable for his environment. Moreover if you represent all the possible vulnerabilities then it will be helpful in the review meeting to identify more vulnerability. Hence this methodology will work as a prototype model.

4.3 Specifying Quality Requirement Attributes Indicators in Use Case Diagram

All the quality attributes are architecturally significant requirements. In UML model

these quality attributes of the system are specified in the Supplementary Documents[10]. The Supplementary Documents are described using natural language. One of the main reasons for the failure of software project is, the project members completely ignore the quality attribute requirements specified in the supplementary documents[10]. Introducing important quality attributes in Use Case Diagram helps the Architect to ensure whether the Architecture satisfies the quality attributes or not and it also helpsto select the appropriate architecture style. For example If the proposed System requires high performance then parallel processing supporting Architectural style is suitableand depending on the number of concurrent users the architectural style differs. Success of the project depends how we balance functional and nonfunctional requirements effectively. Specifying architecturally significant functional and nonfunctional requirements in a simple and understandableway using Use Case Diagram will help to improve the active participation of the customer in the project discussion effectively. If the Architect knows all the ASR completely then architecture design decision can be achieved quickly and effectively. Hence, In the proposed method we introduce Security flow connector for specifying Security requirement and we introduce two more important quality attributes which are Response time and Scalabilityindicatorsto specify performance and size requirements in the Use Case Diagram.

Response time Indicator

Performance quality attribute is focus on Recovery time, Response time, Shutdown time and Start-up time. Recovery time is time taken to recover from failure, Startup and Shutdown time specifies time takes to Start and shutdown the system respectively. Response time is time taken to complete the task. Response time is an important architecturally significant requirement because it measures the Performance of the system.

The Guidelines for usage of Response time Indicator

- Response time Indicator is useful to specify maximum response time required to complete the task initiated by an actor or use case
- The triangular symbol is a Response time Indicator
- The Response time Indicator canbe added to your model in any place to specify the Maximum time to complete the task.

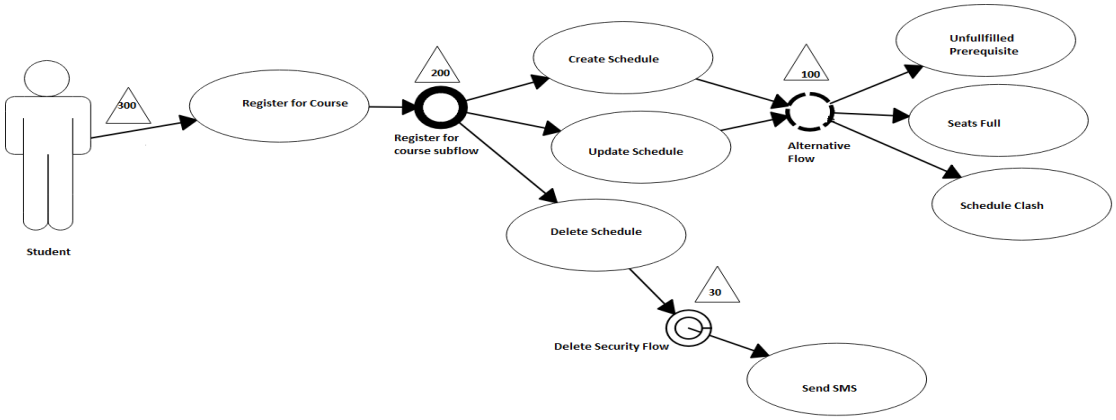

Figure 6 Usage of Response Time Indicator

For example, the maximum response time for the students to complete the course registration is 300 seconds. This response time is also verified in each stage. If the student selects all right courses then the response time is 200 seconds. If the student chooses wrong options then an additional response time is 100 seconds.

Scalability Indicator

Scalability is an architecturally significant requirement because size is important parameter that influencesSupportability of the system By definition Scalability is "What volumes of users and data will the system support?"[18]. For example the number of concurrent users using the system is an important parameter for calculating the Supportability of the system.

The Guidelines for usage of Scalability Indicator

- Scalability Indicator is useful to specify maximum data transfer or maximum number of users access in particular time.
- The double square symbol is a Scalability Indicator
- The Scalability Indicator can be added to your model in any place to specify maximum volume of data transaction or maximum number of users access in particular time.

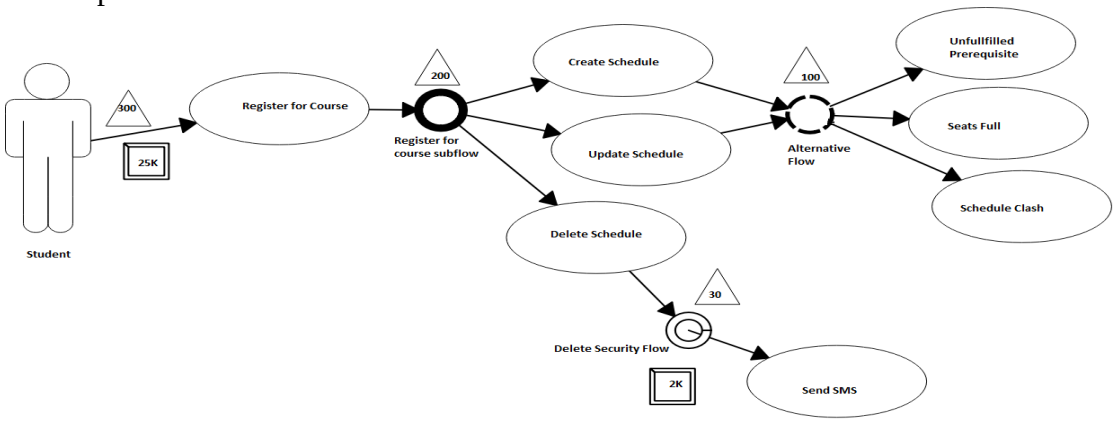

Figure 7 Usage of Size Indicator

In the course registration system during the registration maximum of 25, 000 students will be scheduled at a particular time. The concurrent user requirement is specified in the Figure 7 using Triangular Symbol. Specifying the Size requirement in the Use Case diagram is helpful to confirm the Scalability of the system. Based on the concurrent user the Architectural design will be decided by the software Architect.

7. An application system example: ATM

In the following section we describe an ATM Bank system case study that illustrates the Architect Use Case diagram. ATM System has many modules like Customer Authentication, Withdraw, Deposit etc. In the following section withdraw requirement specification is described

Problem: Requirement Statement for Withdraw Money

Customer can withdraw the money in any one of the fast or custom withdraw option. Fast cash is used for specified amount under options 1000, 5000, 10000, 15000 and 25000, whereas custom withdraw is entering a specified amount. The maximum execution time to complete withdraw operation is within 30 seconds. During Withdraw the following events may occur.

1. Customer inputs an amount which is not a multiple of hundred, then ATM instructs to input an amount in multiples of hundreds.
2. Customer inputs an amount which is beyond the balance orit is exceeds the maximum withdrawal of the day, then ATM instructs to input a lower amount than balance or within daily limit respectively.
3. Customer inputs an amount which is beyond the amunt available in ATM, then ATM instructs to input a lower amount.
4. If The network link broken for 3 seconds then ATM instruct try later.

Solution:

The Use Case Model for Requirement Analysis is designed is2 ways either UML Use Case Model designed without Relationship (Figure 8) or with Relationship (Figure 9).

Figure 8 The UML Use Case Model designed without Relationship

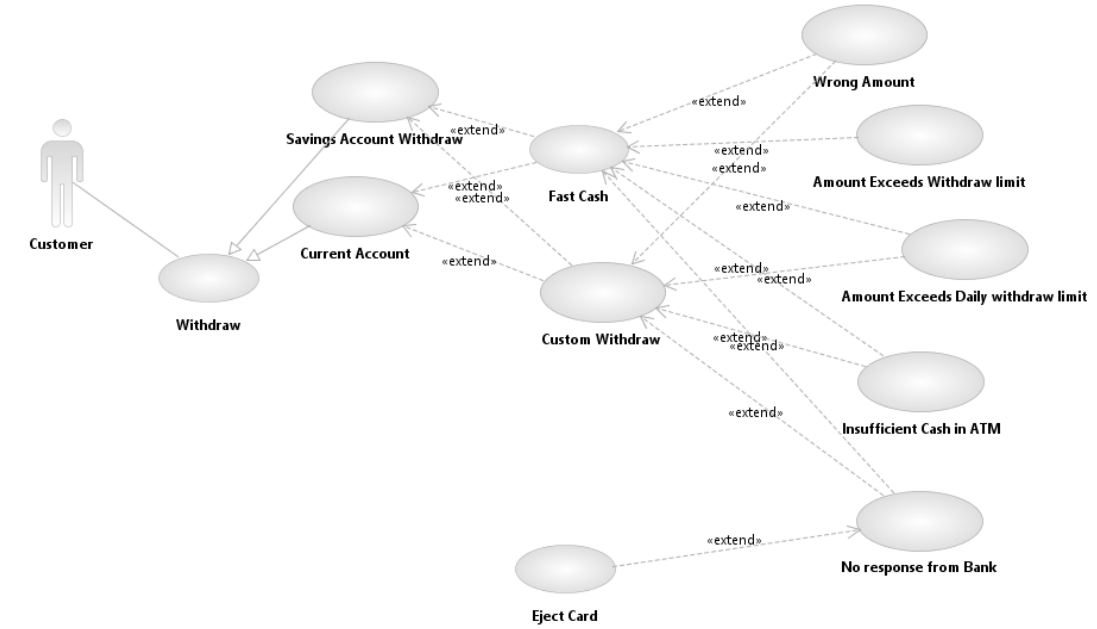

Figure 9 The UML Use Model designed with Relationship

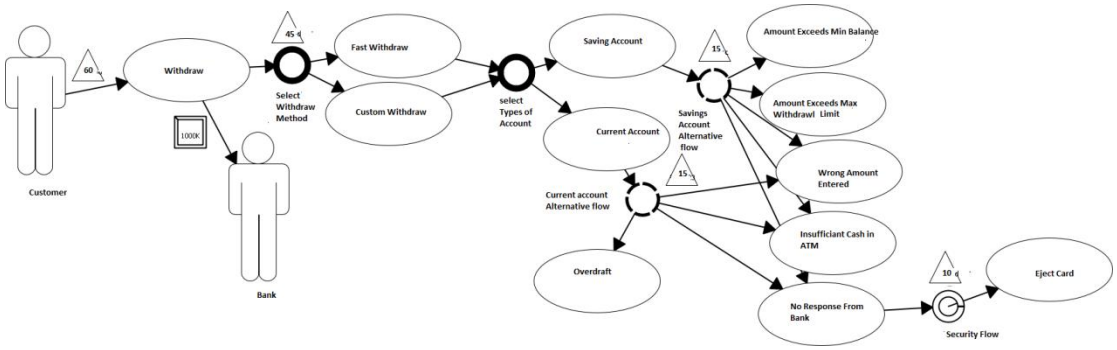

Figure 10 Model designed using Proposed Use Case Diagram

UML use case model without relationship is simple and it gives abstract information to System Analyst, Architect and customer. Figure 8 gives abstract information that customer can withdraw money from the ATM. Using this information System Analyst and Architect does not get any architectural design decision requirements.

UML use case model with relationship is complex to understand but it gives better information than the UML use case model without relationship. But usage of include and extend relationship in the Use case Model makes it difficult for the customers tounderstand. Extend relationship can be used for optional behavior and alsoexceptionalbehavior use case. Hence usage of extend relationship is also confusing.. Readability of this diagram in Figure 9is very poor, and also

differentiation of optional use case and exception use case is difficult. Software Architect and System Analyst can understand the diagram but it difficult task for the customer to understand. Figure 9shows only functional requirement but it does not give any information about nonfunctional requirements like security, performance and scalability.

The proposed Architects Use Case diagram (AUCD) solves the customer readability issues. This method uses different notation to describe the relationship. Sub flow connector specifies optional (menu driven) use cases that are connected to the subflow connector (Arrow indicates subflow connector to use case) which are invoked based on the user input. Alternative flow connector specifies use case connected to the alternative flow connector (Arrorw indicates alternative flow connector to use case) which are invoked based on the internal logic of the system. Security flow connector specifies use case which handles the security vulnerability connected to the security flow connector (Arrow indicates security flow connector to use case). Response time Indicator specifies expected time to complete a particular task. Scalability Indicator specifies expected volume of data transaction or numbers of concurrent users in the system. These notations are simple to understand and using these notation most of the ASR are specified. Specifying most of ASR in graphical Use Case Diagram helps active participation of customers for identifying more ASR. Identifying all possible ASR at early stage is helpful to design a quality Architecture.

Figure 10 describing Model designed using Proposed Architect's Use Case Diagram. After selecting withdraw option customer have the options of Fast withdraw and Custom withdraw. After selecting the withdrawal option customer also have an option to select the account type (Savings or Current Account). If customer enters valid input basic flow use case (savings or current use case) complete the task. If customer enter invalid input like amount exceed minimum balance or excess daily transaction limit or wrong amount then respective alternative flow use case will be invoked. Customer enters a valid withdraw amount but due to insufficient cash or network issues no response from bank is also a possible alternative flow. After completing all the withdraw steps and cash dispenser does not eject the money which is a serious security issue. Security flow connector handles this issue. Network failure is another important security issue because during the transaction customer ATM card is inside the ATM machine. Security flow connector is used here to handle this flow. Scalability indicator specifies that a maximum of 1000 concurrent users will access the Bank Server thro ATM Machine.

Figure 10 gives following Design Decision information to Software Architect

ASR details in AUCD	**Architect Observation**
1000 concurrent users will access ATM Network	Transaction depends on Bank network, so proper mechanism is required to handle network delay
Maximum time to complete the withdraw is 30 seconds	Different options withdraw type, account type is involved and more than 6 alternative flows and two security flows is involved. So completing withdraw options with handling alternative flow is difficult. Discussion is required to finalize the performance requirements.
6 alternative flows	High exception handling mechanism is required
2 Security flow	Discussion with the customer is required to handle the security issues

Comparing the features of UML Use Case Diagram method with Architect's Use case Diagram

Table 1 is comparing the advantages of the Architect's Use Case Diagram with the Existing Use Case Diagram.

5. Validating Architect's Use Case Diagram for specifying Architectural Significant requirements

Validating any requirement specification is based on the four properties complete, consistent, unambiguous, and correctness. Apart from the above four properties the most desirable property is verifiable and Traceable. Using these properties the following section AUCD is validated.

5.1 Complete:

The Requirements model is complete if all possible scenarios are represented including exceptional behavior[4]. Architecturally significant requirement requires complete specification of core functions and quality attributes like performance, security and scalability.

Incompleteness in UML Use Case Diagram

The UML use case Diagram with relationship and without relationship only specifies functional requirement and the diagram do not give any information about the nonfunctional requirements. Hence it is an Incomplete Requirement document. Figure 8 and Figure 9 does not give performance, security and scalability information.

Complete requirement specified using Proposed Architect's Use Case Diagram

To represent a complete requirement the proposed Architect's Use Case Diagram introduces Sub flow, Alternative flow and Security flow. The sub flow is very useful

to represent choice in the use case. Alternative flow is useful to represent all possible alternative paths to achieve the goal. Security flow is useful for highlighting security fall and correction method. Figure 10 specifies complete requirement.

5.2 Correct

By definition an individual requirement is correct, if it "represents something that is required of the system to be built"[2].

Incorrectness in UML Use Case Diagram

The Usage of extend relationship gives an incorrect information to the project members because extend is used to specify optional use case as well as exceptional use case. Hence the usage of extend may give an incorrect information to project members as all the optional cases are exceptional cases or all the exceptional cases are optional cases.

For example in Figure 9 The usage of extend association in withdraw use case (Figure 9) is for specifying that withdraw can be done in two modes fast withdraw or custom withdraw and withdraw can be done from any one of the account of current or savings or credit card account. All these options are taken by the user. The extend association is also used to specify the exceptional behavior like amount exceeds daily withdrawn limit, amount exceeds minimum balance, no response from bank, insufficient cash in ATM.. Hence using the extend relationship for selecting user options and exceptional behavior based on the user input will create a confusion to the user. Hence this specification is incorrect.

5.3 Correct requirement specified using Proposed AUCD

Architect's Use Case Diagram introduced sub flow and alternative flow connectors. Sub flow connectors are used to specify menu types and alternative flow connectors are used to specify exceptional events. Hence different usage of symbols helps to specify correct requirements

In withdraw (Figure10) use case diagram sub flow connectors are used to specify the user options for withdraw modes Fast cash or custom withdraw and also to specify withdraw options of current account and savings account

5.4 Verifiable

The Architect's Use Case Diagram representation ensures to verify the important nonfunctional requirements like Performance and Security. The existing UML use case diagram does not represent the nonfunctional requirements.

In account verification use case diagram (Figure 10) it is clearly expressed that the performance requirements withdraw should take maximum of 60 seconds to complete. Maximum number of concurrent users (size) is also represented in Architect's Use Case Diagram.

5. Conclusion and future work

Software Architecture plays vital role in success of the software Project. Good

Software Architecture depends on ASR. But in practice stake holders and requirements engineer frequently fail to express or effectively communicate ASR's to the architects due to lack of techniques for capturing architectural requirements and these techniques also do not have proper expressive power particularly in the areas of describing quality requirements for different stakeholders. Usually most of the stake holders are a non-technical person who can understand the use case diagram only. But existing Use Case Diagram does not have the provision to represent architecturally significant requirements. The proposed AUCD method enhanced features to accommodate ASR. This method uses different notation to describe the relationship. Sub flow connector specifies menu driven use cases. Alternative flow connector used to specify possible alternative use cases are involved to achieve a goal. Security flow connector specifies use case which handles the security vulnerability. Response time Indicator specifies expected time to complete a particular task. Scalability Indicator specifies expected volume of data transaction or numbers of concurrent users in the system. These notations are simple to understand and using these notation most of the ASR are specified. Specifying most of ASR in graphical Use Case Diagram helps active participation of customers for identifying more ASR. Identifying all possible ASR at early stage is helpful to design a quality Architecture. We also developed the tool to implement AUCD method. The Major Architecturally significant nonfunctional requirements of Security performance and Scalability are represented in AUCD. It is possible to represent other nonfunctional requirements in AUCD. Using this method it is possible to generate automated test plan and test cases. Hence automated tool may be developed for that purpose. Architect's Use Case Diagram (AUCD) for specifying Architectural Significant Requirements (ASR) will be helpful to the requirement analyst, developer, tester, cost estimator and customer.

Table 1: Comparison of the Existing Use Case Diagram and Architect's Use Case Diagram

Activity	Using Existing Use Case Diagram	Using Architect's Use Case Diagram
Highlighting security requirement	Security requirement can be represented as one of the use case but it will not highlight as a security requirement.	Using Security connector it is possible to highlight the security requirement with proper link with the base use case
Effective class diagram	Designing class using use case diagram alone is a difficult task. It is necessary to refer the use case specification to identify the functions for the class	All the sub functions and alternative functions are represented. so it is an easy task to identify the functions for class

Identifying functional test cases	Only few test cases can be identified from use case diagram. By referring use case documentation it is possible to identify more test cases. But it is very difficult task to extract the test cases.	All the flow of the events are represented in the Architect's Use Case Diagram. Hence each flow is a functional test case.
Identifying test cases for performance and security test case	It does not have information about performance and security detail. This details are only available in supplementary documents	Using performance, size and security connector the performance and security requirements are represented. Hence it is easy job to write the test case for performance and security.
Constructing Factor Table for Architectural decision support	It does not have information to construct Architectural factor table	Using performance, size, security, sub flow, alternative flow connector it is possible to describe the complete quality scenario. Hence it is easy task to build an architectural factor table
Measuring the complexity of the use case	Manually calculated	All the possible use cases are represented hence it is an easy task to calculate the complexity of the base use case

Reference

[1]. Ahmad K, Shuja, Jochen Krebs, IBM Rational Unified Process Reference and Certification Guide, IBM Press, Pearson Education, 2008, 106-107,

[2]. Alan M, Davis, Software Requirements: Analysis and Specification, Prentice Hall Press, Upper Saddle River, NJ, USA, 2nd edition edition, 1993,

[3]. Asoke K, Talukder, Manish Chaitanya, Architecting Secure Software Systems, CRC Press, 2008

[4]. Bernd Bruegge, Allen H, Dutoit, Object-Oriented Software Engineering, Pearson, 2013(page 215)

[5]. Charles Babers, Architecture Development Made Simple, CJC Publishing, 2003

[6]. Christian F, J, Lange, Michel R, V, Chaudron, and Johan Muskens, In Practice: UML Software Architecture and Design Description, IEEE SOFTWARE, 2006, 40-46

[7]. Daryl Kulak, Eamonn Guiney, Use Cases: Requirements in Context, Addison-Wesley, 2012

[8]. Gonzalo Genova, Jan Llorens, and Victor Quintana, Digging into Use Case Relationships, Springer, The Unified Modeling Language Lecture Notes in Computer Science Volume 2460, 2002
[9]. Grady Booch, James Rumbaugh and Ivar Jacobson, The Unified Modeling Language User Guide, Addition Wesely, 2001
[10]. Kurt Bittner, Ian Spence, Use Case Modeling, Addison Wesley, 2002
[11]. Len Bass, John Bergey, Paul Clements, Paulo Merson, Ipek Ozkaya, Raghvinder Sangwan, A comparison of Requirements Specification Methods from a Software Architecture Perspective, Technical Report, Carnegie Mellon Software Engineering Institute, Pittsburgh, August 2006
[12]. Lianping Chen, Muhammad Ali Babar, and Bashar Nuseibeh, Characterizing Architecturally Significant Requirements, IEEE Software April 2013
[13]. Markus Schumacher, Eduardo Fernandez-Buglioni, Duane Hybertson, Frank Buschmann, Peter Sommerlad, Security Patterns: Integrating Security and Systems Engineering, John Wiley & Sons
[14]. OMG Unified Modeling Language (OMG UML), Infrastructure, Version 2, 2, 2009, Object Management Group http://www, omg, org/spec/UML/2, 2/Infrastructure
[15]. OMG Unified Modeling Language (OMG UML), Superstructure, V2, 1, 2, 2007, ObjectManagement Group http://www, omg, org/spec/UML/2, 1, 2/Superstructure/PDF
[16]. Pascal Roques, UML in Practice: The Art of Modeling Software Systems Demonstrated through Worked Examples and Solutions, John Wiley & Sons, 2006
[17]. Peter Eeles, Capturing Architectural Requirements http://www, ibm, com/developerworks/rational/library/4706, html
[18]. Peter Eeles, Sample Architectural Requirements Questionnaire, http://www, ibm, com/developerworks/rational/library/4710, html
[19]. Richard N, Taylor, Nenad Medvidovi and Eric M, Dashofy, Software Architecture Foundations, Theory, and Practice, John Wiley & Sons, Inc, 2010
[20]. S, Lilly, Use case pitfalls: top 10 problems from real projects using use cases, Technology of Object-Oriented Languages and Systems, 1999, TOOLS 30 Proceedings IEEE(1999) 2-3,
[21]. Sayed Mehran Sharafi, SHADD: A Scenario-based approach to software architectural defects detection, Advances in Engineering Software, Elsevier Ltd, 2011,