# EFFECTIVE REQUIREMENT & USE CASE MANAGEMENT FOR SOFTWARE DEVELOPMENT

Vithal Kadarmanadalgi, PMP, Business Analyst/Project Planner, IMSI
Deni Martin, Team Lead/Project Planner, IMSI

## Introduction

In the 1990's, the Software industry made a concerted effort to become more process-oriented. Standards for measuring and certifying effective software developmental processes were introduced and popularized. An increasing number of software tools were developed to define and apply these new and effective development processes to software application projects. The result of implementing new processes and tools was a higher quality end product.

However, there is still a high incidence of software project failure today. Why are many, if not most, software projects still plagued by delays, budget overruns and quality problems? How can we improve the quality of the system we build as our businesses, national economies and daily activities become increasingly dependent on effective and efficient software?

This paper suggests that one of the critical elements in delivering software on time and on budget is the identification, implementation and processing of requirements. We will refer to this activity as "Requirement Management." We'll also explain "Use Case Management" and how it is used in conjunction with requirements to facilitate effective software development and delivery.

## Requirement Management

### Why Invest in Requirement Management?

The price of failure can be enormous if business requirements are not accurately captured for a software development project. Quite a few studies indicate software projects spend roughly one-third of their overall budget correcting errors that originate in the requirement gathering process. Here is what some additional research reveals:

- The Standish Group's CHAOS Reports from 1994 to 1997 established that the most significant contributors to project failure relate to requirements
- In December 1997, Computer Industry daily reported on a study of 500 IT managers in the United States and the United Kingdom and found 76% of the respondents had experienced complete project failure during their careers. The most frequently named cause of project failure was "changing user requirements."

Effective requirement management is a key contributor to successfully delivering quality software. For example, customer satisfaction is enhanced if your clients/stakeholders are actively part of the requirement gathering and prioritization process. Whether defining requirements for new software, software that will be purchased or existing software to be enhanced or maintained, it is easier for the customer to compare what was asked for with what was delivered into their hands.

### So “What is a Requirement”?

There are many definitions of "requirement" suggested by different bodies (for example - IEEE). Here are some of the definitions related to software development:

- A software capability needed by the user to solve a problem or achieve an objective
- A software capability that must be met or possessed by a system or system component to satisfy a contract specification, standard or other formally imposed documentation
- A defined operational capability of a system of process that must exist to satisfy a business need.

The generic term "requirement" often falls into two categories or types - Functional and Non-Functional.

- A Functional requirement evolves from a user requirement (i.e. tasks that users need to achieve using the software).

- A Non-Functional requirement covers quality attributes of the software such as the performance, system needs such as security and archiving, and technical constraints such as coding language and database structure.

## A Multi-Level Information Gathering Process

Overview

Requirements are the product of a requirement management process that requires discovery, definition and implementation. A key player in this process is the project analyst. His or her challenge is to think of requirements from three perspectives – the business level, the user level and the technical level.

The Business Level

At the "business" or highest level, the analyst concentrates on understanding and clarifying the project's business goals and objectives. A vision is defined on how the product will achieve those outcomes and the vision becomes the most critical or core requirement. Every subsequent requirement is measured against the vision's goals and objectives.

Here's an example of an IT project's vision from the banking industry: provide the consumer with the ability to use a single card at convenient locations to withdraw cash from one of his/her accounts. This vision was actually cast in the 1970's and led to the development of the first ATM machines. Deni Martin helped test Michigan National Bank's first prototype ATM in 1978. This core requirement has matured over the years, so today you can use one card to pay for medicine at the corner drugstore, buy groceries, rent a car or withdraw cash.

The business level not only includes articulating a project vision, but also requires identifying the entire project's stakeholders and direct users of the system. Who will be impacted directly or indirectly by the software? These findings are recorded in a template document specified by the organization that is managing the project. (Standard templates are available in the marketplace or on the web.)

The User Level

The "user" level focuses on the direct users of the system to define the user requirements. The analyst's challenge is to identify different types of users called "'actors" and the tasks they will be performing with the software. Later in this paper we'll explain "use cases." Use cases are the best modeling construct for defining user tasks.

The Technical Level

The "technical" level of requirements include functional requirements and non-functional requirements based on user input. Traditionally, software development projects have used text-based functional hierarchies to describe the user requirements in declarative statements. A more recent development has made "use cases" the de facto standard for describing interactions between users (actors) and the system. Some projects employ use cases to augment their functional hierarchies, while other projects utilize detailed use cases that cover the software's entire behavior and serve as a substitute for functional hierarchies.

### Requirement Gathering Techniques

Two groups are critical in gathering requirements: the stakeholders and the users. The stakeholder's financial and professional investment in the project often dictates the visionary and deliverable requirements. "It has to do what we ask it to do (or it's worthless). Stakeholder requirements can be gathered in informal one-on-one sessions with the analyst or in formal group sessions. Ultimately, they will have approval or veto rights on what is delivered in the software.

Users define the finer, detailed requirements. Using the bank ATM example, the stakeholder is concerned with the customer's ability to withdraw cash and, ideally, being charged a fee for this service. The user worries about inserting the card into the ATM, entering some sort of code, picking the account to withdraw cash from, getting his/her cash and receiving a printed receipt. User requirements can be gathered using a variety of techniques, including focus groups where the analyst attempts to steer the requirement gathering to a specific area of the total project.

### Requirement Management Tools

Once the requirements are gathered, the analyst uses a variety of tools to document, track and manage them. For a small project, you can utilize word processing (like MS Word), database (like MS Access) and graphics (like MS Paint) software. The key is having established templates that different requirements-related documents such as the project's release charter, business model, analysis and design documents. You can also lump in change control documents, which typically describe additions or alterations to software functionality.

Large projects often use packages provided by vendors like Rational. Rational's strength is their package provides a well-documented process (based on Object Oriented Analysis), the supporting methodologies and robust progress reports. Evidently, their success in the marketplace was significant enough that the company was purchased by IBM. And if you look at Ford Motor Company's "One IT" unified solution delivery methodology, you'll see it is identical to Rational's approach. (IBM is the major software development vendor used by Ford.)

### Build the Requirement Models

"Requirement models" act as a blueprint for a software product. Each individual model takes the form of a diagram, a list, or a table often supplemented with text that depicts a user's needs. For example, context diagrams, data models, class models, business rules, actor maps or working prototypes are requirement models. These models are typically embedded in or attached to the critical software project development documents.

The fundamental purpose of a requirement model is to communicate effectively the intent of the software or an active part of the software. Requirement models communicate to the development team as well as the users. Models can have multiple views or perspectives providing a rich context for soliciting additional requirements. They can also alleviate concerns.

## Use Case Management

### Works in Harmony with Requirement Management

Years of experience in defining requirements have led to the development of a number of techniques and models to assist in the software development process. One of the most effective models is the "Use Case."

Frequently, functional requirements are typically written from the point of view of the software, but a use case is written to reflect the "voice of the customer". Building a software product without understanding their needs is a sure path to failure. Use cases are arguably among the best requirement modeling techniques, especially when managing large projects.

#### Defining and Understanding Use Cases

"A use case is a sequence of transactions in a system whose task is to yield a measurable value to an individual actor of the system." [Jacobson et al., 1995].

To better understand this definition, please note:

- A use case is "a specific flow of events through the system, that is, an instance" ([Jacobson et al., 1995]).
- A set of all items which share similar characteristics or similar courses of events can be grouped into a "use case" class.
- An actor is "a role that someone or something in the environment can play in relation to the business" ([Jacobson et al., 1995]). Alternatively, [Jacobson et al., 1992] defines an actor as representing "everything that needs to exchange information with the system," and [Christerson and Jonsson, 1995] defines actors as "everything that interacts with the system."
- An "individual actor" (sometimes referred to as a "user") is defined to be an instance of a class actor. Keep in mind the same person (or other item) can assume more than one role. For example, in the Ford design verification software known as "eFDVS" a user can serve the dual role of program administrator and engineer.
- A transaction is defined as "an atomic set of activities that are performed either fully or not at all. It is invoked by a stimulus from an actor to the system or by a point in time being reached in the system. A transaction consists of actions, decisions and transmission of stimuli to the invoking actor or to some other actor(s)." (See [Jacobson et al., 1995].) Jacobson provides the following advice for use case creation, and

an accompanying use case example: "If we try to describe a use case that contains a great many alternative courses of events, our text can easily become difficult to understand. Therefore, it is wise to use some form of structured writing approach."
- "Transactions in a system" implies that the system will make available to its actors a set of capabilities that will both allow the actors to communicate with the system and to accomplish some meaningful work (i.e., meaningful value). Like withdrawing cash using our ATM example.
- "A measurable value" implies that the performance of the task has some visible, quantifiable, and/or qualifiable impact on those things which lie outside of the system, and, in particular, the actor who initiated the task.

## Effective Usage and Benefits of Use Cases

Most, if not all, references to Object Oriented methodologies stress that use cases should be used throughout the development part of the software life cycle, e.g., during analysis, design, and testing. [Jacobson et al., 1995] and [Christerson and Jonsson, 1995], among others, even point out that use cases can be an important tool in business process modeling and business process reengineering as well.

Put simply, a use case describes an interaction between an external actor (user) and the system, thereby documenting a major function that the system will perform. Use cases express the behavior of a system and its the functional requirements in a way that helps technical experts and non-technical people alike understand that behavior. Without use cases, functional requirements are often expressed in a series of disconnected shall statements, such as "the system shall ask users to login", "the system shall deliver a printed receipt", "the system shall allow users to withdraw money", etc. Using this format, it is often hard for people not familiar with software to understand its intent and its deliverables.

In contrast, when using use cases, functional requirements are expressed in text that clearly outlines the sequence of steps system users will follow to benefit from using the software. The sequences spell out clearly what the user and the system will do, in the correct order, to accomplish the goal of the system.

An effective use case technique is to provide a graphic diagram, which serves as a simple overview of the system functionality without the excessive details. This simple technique prevents confusion for the people with a non-technical background.

To summarize, use case benefits include:
- They communicate requirements in a way that is understandable and usable for technical as well as non-technical audiences using a single, common format.
- They include both user requirements and system requirements. This minimizes errors that can occur in translating user requirements (for non-technical audiences) into system requirements (for technical audiences).
- Because use cases are readable and easily understood by customers, there is a higher probability that misunderstandings between the customer and the software team are resolved earlier, before the development is well under way, avoiding costly re-work late in the development lifecycle.
- Use cases provide the basis for the whole object-oriented, software life cycle including architecture, design (including GUI design) and development.
- Use cases help testing efforts by facilitating the creation of test cases. All tests must contain a sequence of events, which will be followed to test a particular area of the system. Because use cases already provide requirements in a sequence, testers no longer have to guess at the order and content of user actions.

## Getting Started with Use Cases

Overview

Research indicates the following are the key best practices to use while modeling Use cases in a project:
- Scope the domain
- Scope the use cases
- Validate the use cases as they emerge
- Define the requirements model
- Determine the strategy you will use to validate requirements

- Use a standard format for use cases
- Develop a glossary

Scope the Domain
Use cases are an effective mechanism for scope management. You can define which cases are in and out of the scope of the project before investing too much time in the details. You can also use a top-down approach by creating several modeling constructs, list of stakeholders, a context diagram, a list of events and the definition of domain. This will help identify the domain of the scope and assist in determining use cases that fit the scope.

Scope the Use Cases
Use case scoping is as important as the project scope itself. To ensure that each use case stays in scope, make sure that it addresses a single actor, goal and is not overly complex. At the same time, however, avoid the temptation to create small, piecemeal use cases that handle partial functions or partial processes in the business.

One way to define a well-scoped use case is to frame each use case with triggering events and necessary event responses. In this way, you will know when the use case starts and ends. Events are what cause actors to initiate the use cases. The use case finishes when the actor's goals are satisfied and achieved..

Validate the Use Cases as They Emerge
Whether the use cases in scope are modeled using a top-down approach or simply by listing and associating them with actors (users), it is essential to validate the use cases to ensure that each one is necessary to meet the business objectives defined in the product or project's vision. Validation ensures that the right requirements are defined correctly.

Here are some questions the analyst should ask:
- How does the use case help us achieve our goals and vision?
- Does the use case address some aspect of the problem in our problem statement?
- Does this use case differentiate the product in some way?
- Do we have use cases to address all the stakeholder and users we identified in our vision statement?
- When will use cases be implemented in our project plan to support the initial release?

Build the Requirement Models
We mentioned "requirement models" earlier in this paper. Just to recap, a requirement model is a diagram, a list or a table often supplemented with text that depicts a user's needs. For example, context diagrams, data models, class models, business rules, actor maps or working prototypes are requirement models.

From the point of view of a person or system interacting with the software, a use case describes an aspect of the behavior. But, no single use case will fully describe all the software's functional requirements: its behavior, structure, dynamics and control mechanisms. These are interrelated views of the functional requirements, and they give you complementary mechanisms for analyzing a business domain and modeling it accurately and completely (Exhibit 1)

Determine the Strategy You Will Use to Validate Requirements
We covered requirement gathering earlier in this paper. Here are some additional techniques for gathering use case information and related requirements:
- Generating a list of scenarios
- Reusing existing requirements
- Prototyping a user interface
- Observing users in their work environment
- Mining customer complaints and help desk logs

Settle on a Standard Format for your Use Cases
Using a standard format helps in a variety of ways. It allows the analyst to look into the issues from the user's perspective. It helps in covering both the functional and non-functional requirements. It also ensures all the

business rules are captured in the use case making the job of the software developer easier. (See the example in Exhibit 3.)

Develop a Glossary
If you are dealing with complex use cases and terminology that is unfamiliar to your user base, it may be beneficial to include a use case glossary. The glossary provides a central location in a standard format for reviewers and users of the use cases to read and understand.

## Summary

### Concluding Statements

Requirement and Use Case Management are a new variation on an old theme. They are used to describe the system in terms of how the user will see it and in terms of the delivered, measurable value to the user. (See, for example, the discussions of process descriptions also known as "mini specs" in [DeMarco, 1979].) Even within the object-oriented community, there are other very similar approaches.

It is the very simplicity of requirement and use case management that makes them so powerful. Yet, this simplicity can lead to problems if the creation, integration and maintenance of requirements and use cases is not carefully controlled. The larger and/or more critical the end product, the greater will be the need for rigor in the creation and handling of requirements and use cases.

One final caution in using use cases, object-oriented software engineers should be on guard for problems of conflicts in localization strategies, violations of information hiding, and sloppiness in the creation and configuration management of use cases.

We have included an example of a "Use Case Application" to show the advantages of this software development method.

### Use Case Application

Description: Use Case modeling was used for managing a set of change requests for Ford's Scheduling & Optimization (S & O) application. This example contains one Change Request and the use case model, diagram and related material used to deliver the project two days ahead of schedule.

The project deliverables were very tight and had to go through the Software Development Methodology (SDM) process followed at Ford. The SDM process basically calls for a five-stage process (Analysis, ID & Assess, Design, Build, Test) and then user acceptance testing (UAT) and the roll to production. Due to the holidays in December, the project was planned for delivery before Christmas, 2002. Post-Christmas closure would not be permitted, because the project funding for later work did not exist.

There were a total of six change requests that were planned in the 1.2 release, though our focus is only one change.

One of the fundamental problem I noticed was the lack of understanding of the business process by the IT developers. Based on my earlier experiences, I realized that using a use case would help in bringing the IT community and the users on the same page. Therefore, I used the use case methodology to write a requirement specification document.
This document essentially adds functionality to the existing system based on the users feed back.

A use case model (Exhibit 3) helps basically to identify the users of the system, which helps in identifying what each individual's needs might be. Also, it assists in designing the security and other aspects of the system to capture the user requirements

Based on the interaction and transaction that takes place, I developed one use case model. However, depending on the functionality that is being added, there could be more than one use case for a specific functionality development.

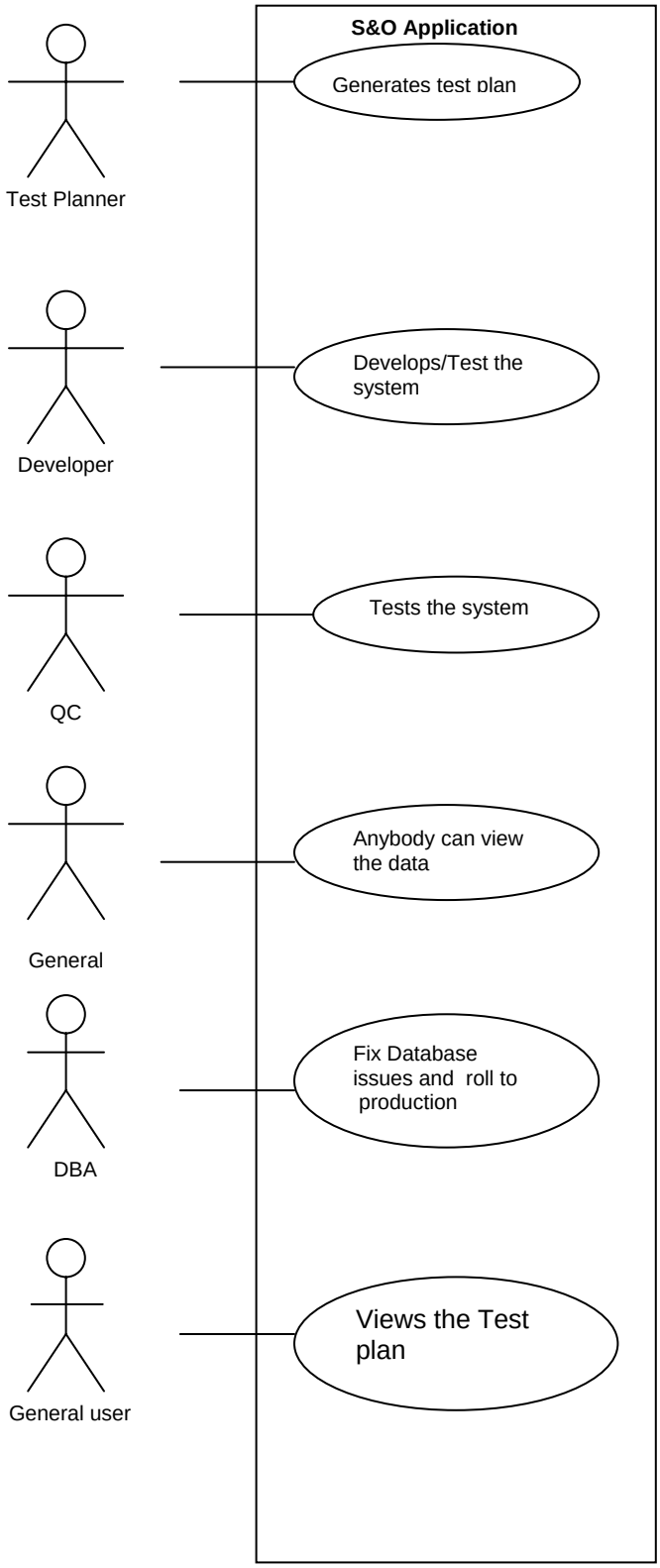
Then a sequence diagram was developed to understand how the user would be using the functionality (see Exhibit 2).

Please note that each diagram has its own benefit, but still does not explain to the user if project will provide a product which will meets their needs. As explained above, a format in plain English is used to explain the actors, the business rules etc. which everybody understands. However, the picture explains the business processes more effectively to the software developers.

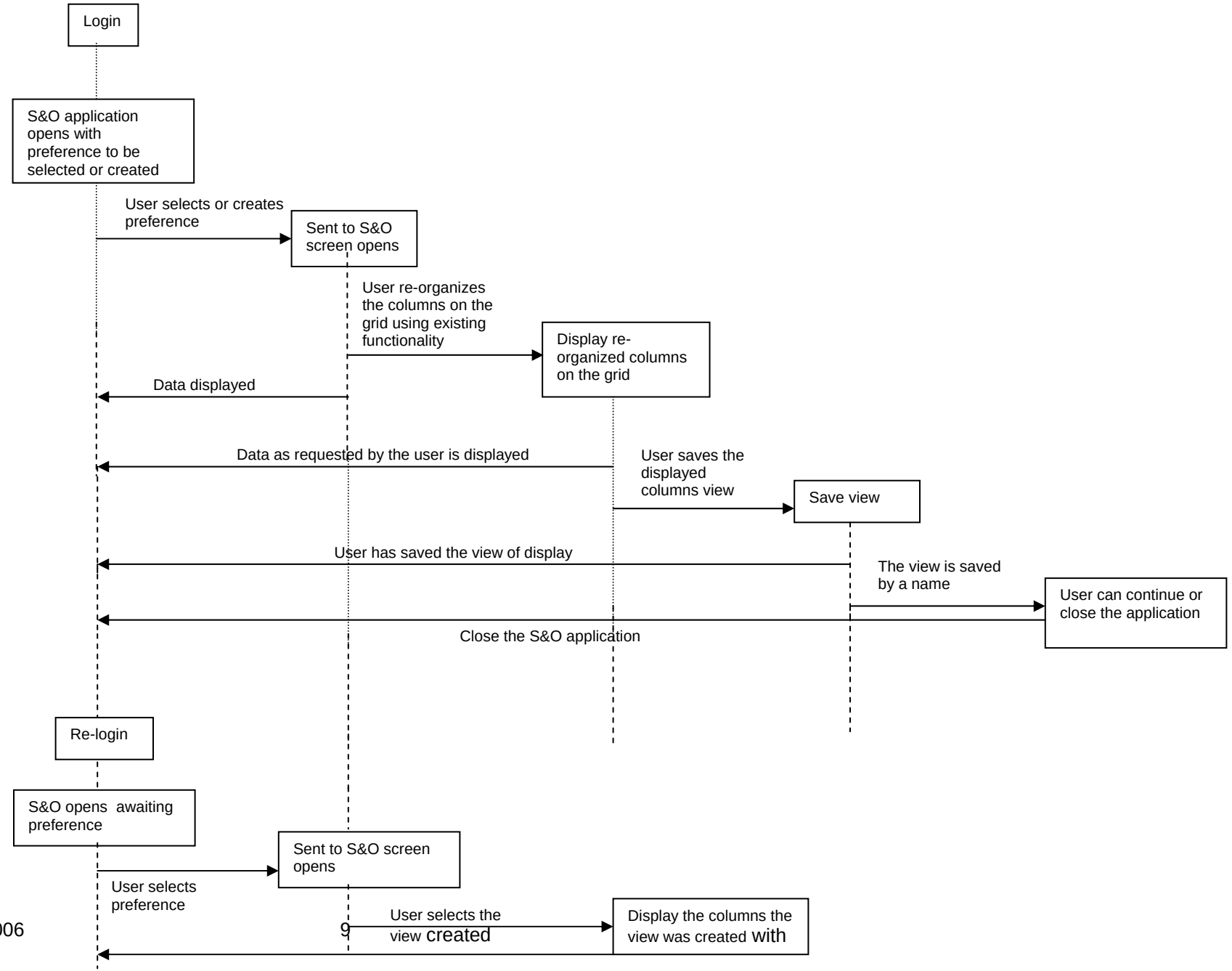
References:


1. [Gibson, 1991]. E. Gibson, "Flattening the Learning Curve: Educating Object-Oriented Developers," Journal of Object-Oriented Programming, Vol. 3, No. 6, February 1991, pp. 24 - 29.
2. [Jacobson et al., 1992]. I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard, Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, Reading, Massachusetts, 1992.
3. [Jacobson et al., 1995]. I. Jacobson, M. Ericsson, and A. Jacobson, The Object Advantage: Business Process Reengineering With Object Technology, Addison-Wesley, Reading, Massachusetts, 1995
4. Improving software development capability – Use Cases - Best Practices: Ellen Gottesdiener (Online article)
5. Bittner, kurt and Ian Spence, use Case Modeling. Addison Wesley 2003
6. Applying Requirements Management with Use cases Roger Oberg, Leslee Probasco, and Maria Ericsson (Rational Software White Paper – Online article)

**Use Case Model For Developing "Save View" functionality (Exhibit 1)**



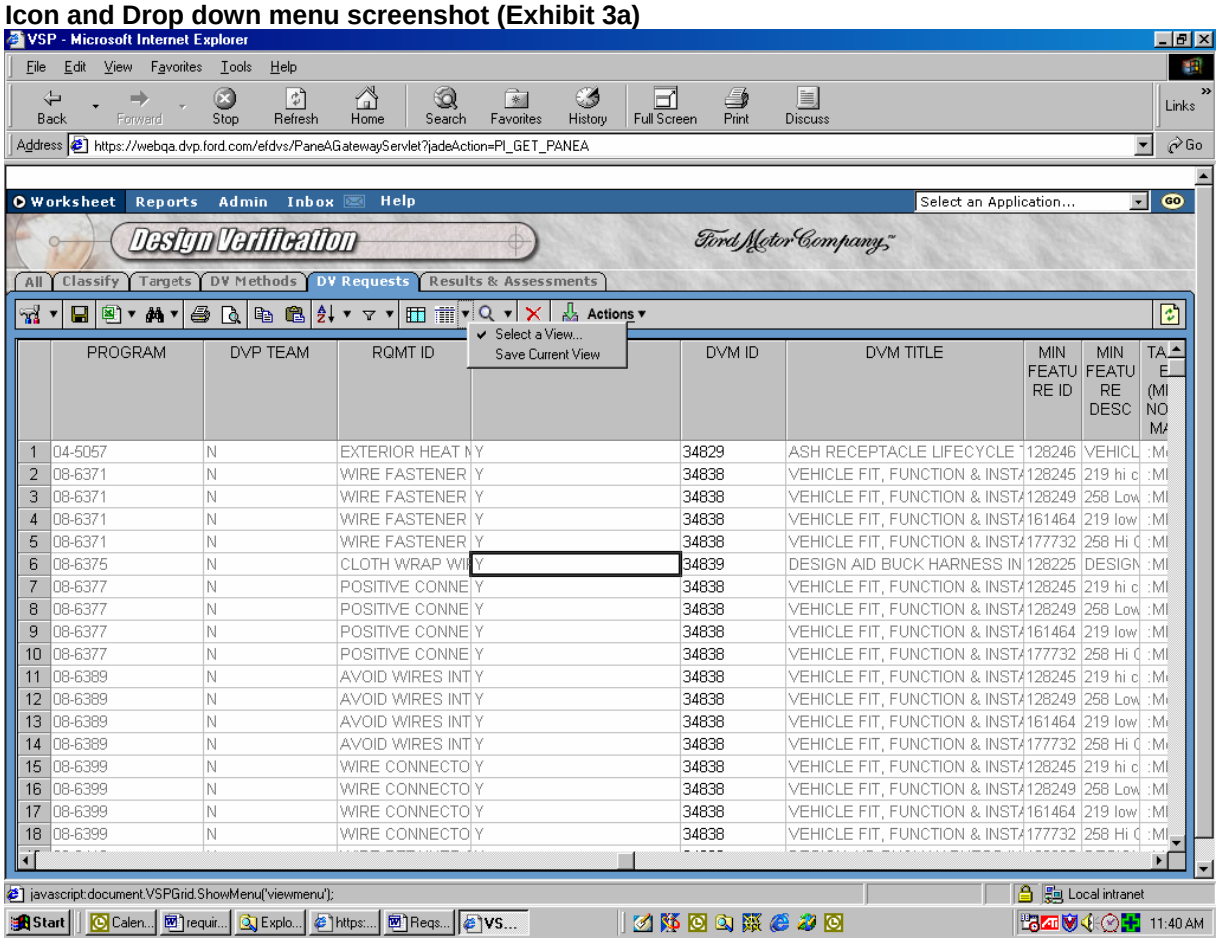
**Sequence Diagram for Save Views Functionality into S&O (Exhibit 2)**



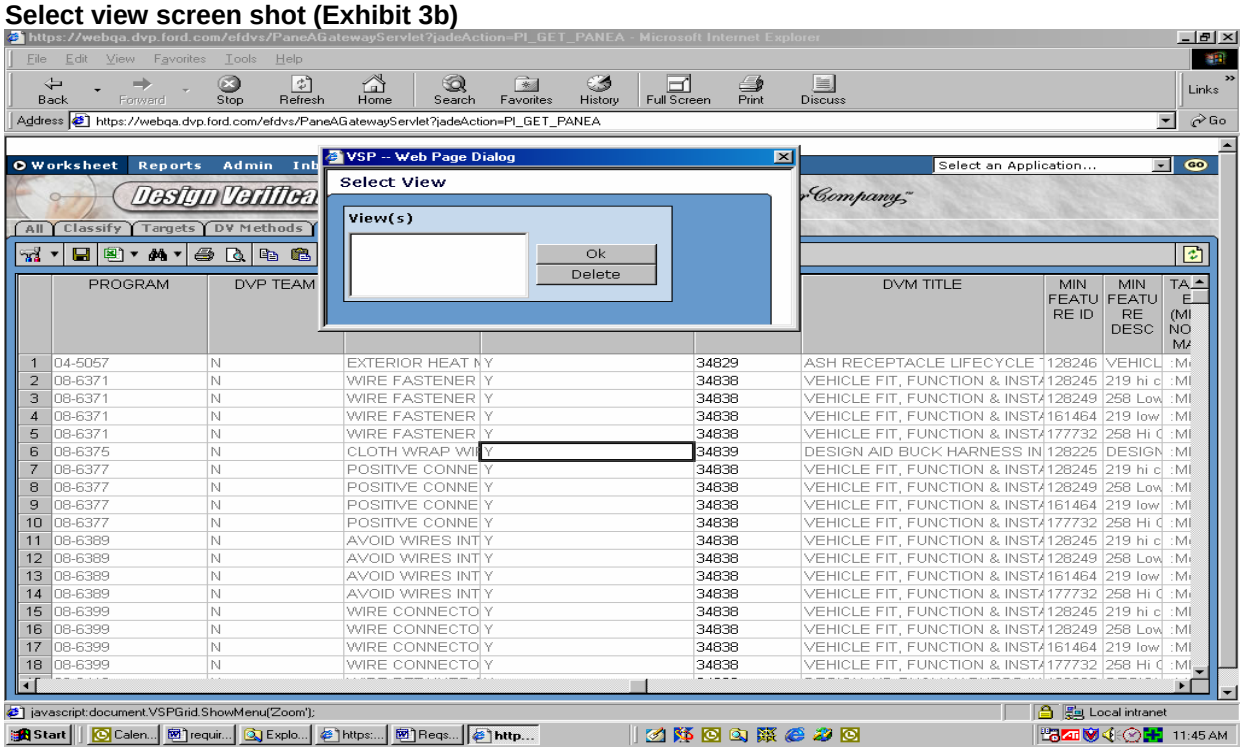
**Use Case:** Save view (Exhibit 3)

| | |
|---|---|
| **Use Case Name:** | **Save Views** |
| **Use Case ID#:** | **CRID: 95005** |
| **Actor:** | Test Planner, Business Analysts, IT developers/QC testers |
| **Summary:** | This use case assists the developer to develop "Save Views" feature. All the 4 grids allow the user to move the columns, however it does not allow saving the view preferred by the user. **The user should be able to save the view, thus help the user to avoid doing this every time he leaves and comes back to the screen. The system should also take into consideration to hide the column, sorted, filter etc. This is similar to feature already provided in eFDVS** |
| **Trigger:** | When a user enters the URL on the browser http://web.dvp.ford.com/so/soLogin in Production environment or http://webqa.dvp.ford.com/so/soLogin in Test environment or http://dev.ford.com/so/soLogin |
| **Preconditions:** | 1. The user must have a PC with either IE 5 or above browser installed with an Internet connection |
| **Basic Course of Events:** | • User opens up S&O after authentication<br>• User could choose an existing preference or create a new preference or edits an existing preference<br>• Selects the preference based on the above condition<br>• Sent to S&O screen opens up<br>• User re-organizes the columns and would like to save this view<br>• **Provide an icon to save the views (refer to the screenshot below)**<br>• **Further, the icon should have a drop down to offer the choice the following options: 1. Select a view 2. Save current view as shown in the screen shot below**<br>• **If the user chooses option1 (select a view option), a window should pop up as per screen shot below, and allow the user to select the view required from the drop down showing the listing of various views**<br>• **If the user chooses option 2 (save current view), a window should pop up as per the screen shot below, and allows to save the name input by the user**<br>• **This feature with the icon should be provided on all the 4 grids** |
| **Alternative Paths:** | User could be on another screen of the S&O application. In this case the user can navigate to any of the 4 grids from wherever he or she is |
| **Exception Paths:** | . |
| **Nonfunctional Requirements:** | Refer to 3.1.1 |
| **Assumptions** | |
| **Post conditions:** | |
| **Author:** | Vithal Kadarmandalgi |
| **Date** | 07/12/02 |

## Icon and Drop down menu screenshot (Exhibit 3a)




## Select view screen shot (Exhibit 3b)



## Save View screenshot