Business Requirements

Business requirements collected from multiple sources might conflict. For example, consider a kiosk product with embedded software that will be sold to retail stores and used by the store's customers. The kiosk developer's business objectives include the following:

- leasing or selling the kiosk to the retailers
- selling consumables through the kiosk to the customer
- attracting customer to the brand
- modifying the nature of the historical developer-customer relationship

The retailer's business interest could include:

- making money from customer use of kiosk
- attracting more customers to the store
- saving money if the kiosk replaces manual operations

The developer might want to establish a high-tech and exciting new direction for customers, while the retailer wants a simple solution and the customer wants convenience and features. The tension among these three parties with their different goals, constraints, and cost factors can lead to conflicting business requirements, which must be resolved before the kiosk's software requirements are detailed.

You can also use the business requirements to set implementation priorities for use cases and their associated functional requirements. For example, a business requirement to generate maximum revenue from the kiosk would imply the early implementation of features directly associated with selling more products or services to the customer, rather than glitzy features that appeal to only a subset of customers.

The Vision Statement

The vision statement should reflect a balanced view that will satisfy the need of diverse customers. It can be somewhat idealistic but should be grounded in the realities of existing or anticipated customer markets, enterprise architectures, organizational strategic directions, and resource limitations.

Chemical Tracking System

The chemical tracking system will allow scientists to request containers of chemicals to be supplied by chemical stockroom or by vendors. The location of every chemical container within the company, the quantity of the material remaining in it, and the complete history of each container's location and usage will be known by the system at all times. The company will save 25% on chemical costs by fully exploiting chemicals already available within the company, by disposing of fewer partially used or expired containers, and by using a standard chemical purchasing process. The chemical tracking system will also generate all reports required to comply with federal and state government regulations that require the reporting of chemical usage, storage, and disposal.

Assumptions and Dependencies

All assumptions that were made when conceiving the project have to be recorded. For example, the management sponsor for the chemical tracking system assumed that it would replace the existing chemical stockroom inventory system and that it would interface to the appropriate purchasing department applications

Scope

Project scope defines the concept and range of the proposed solution, and limitations identify certain capabilities that the product will not include. Clarifying the scope and limitations helps to establish realistic stakeholder's expectations. Sometimes customers request features that are too expansive or do not lie within the intended project scope. Propose requirements that are out of scope must be rejected, unless they are so beneficial that the scope should be enlarged to accommodate them (with accompanying changes in budget, schedule, and staff). Keep a record of these requirements and why they were rejected, as they have a way of reappearing.

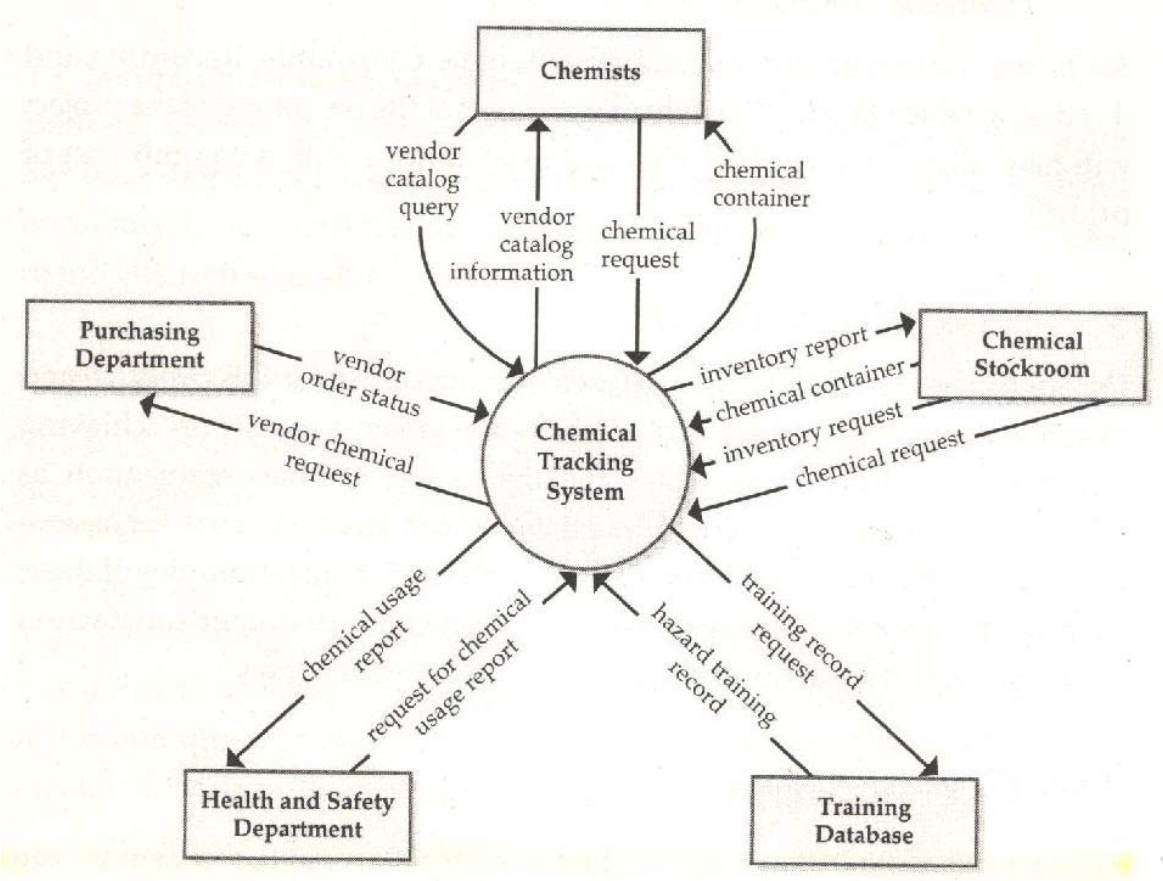
Scope and Initial Release

The major features that will be included in the initial release of the project should be summarized. Describe the quality characteristics that will enable the product to provide the intended benefits to its various customer communities.
Requirements need to be prioritized and a release schedule should be made.

The Context Diagram

The scope description establishes the boundary between the system we are developing and everything else in the universe. The context diagram graphically illustrates this boundary by showing the connections between the system being developed or the problem being addressed, and the outside world. The context diagram identifies the entities outside the system that interface to it in some way (called terminators or external entities), as well as the flow of data and material between each external entity and the system. The context diagram is used as the top level abstraction in a dataflow diagram developed according to principles of structured analysis. The context diagram can be included in the vision and scope document, in the SRS, or as part of a dataflow model of the system.

Following is a context diagram of the chemical tracking system.



Use Cases and Customer-Developer Relationship

It has been mentioned earlier on, excellent software products are the result of a well-executed design based on excellent requirements and high quality requirements result from effective communication and coordination between developers and customers. That is, good customer-developer relationship and effective communication between these two entities is a must for a successful software project. In order to build this relationship and capture the requirements properly, it is essential for the requirement engineer to learn about the business that is to be automated.

It is important to recognize that a software engineer is typically not hired to solve a computer science problem – most often than not, the problem lies in a different domain than computer science and the software engineer must understand it before it can be solved. In order to improve the communication level between the vendor and the client, the software engineer should learn the domain related terminology and use that terminology in documenting the requirements. Document should be structured and written in a way that the customer finds it easy to read and understand so that there are no ambiguities and false assumption.

One tool used to organize and structure the requirements is such a fashion is called use case modeling.

It is modeling technique developed by Ivar Jacobson to describe what a new system should do or what an existing system already does. It is now part of a standard software modeling language known as the Unified Modeling Language (UML). It captures a discussion process between the system developer and the customer. It is widely used because it is comparatively easy to understand intuitively – even without knowing the notation. Because of its intuitive nature, it can be easily discussed with the customer who may not be familiar with UML, resulting in a requirement specification on which all agree.

Use Case Model Components

A use case model has two components, use cases and actors.

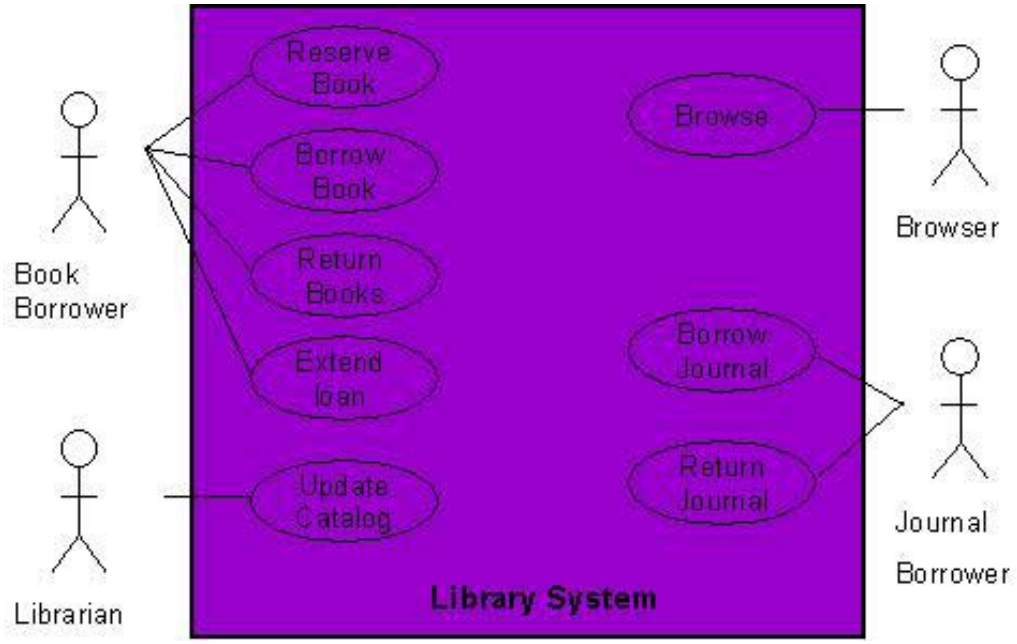
In a use case model, boundaries of the system are defined by functionality that is handled by the system. Each use case specifies a complete functionality from its initiation by an actor until it has performed the requested functionality. An actor is an entity that has an interest in interacting with the system. An actor can be a human or some other device or system.

A use case model represents a use case view of the system – how the system is going to be used. In this case system is treated as a black box and it only depicts the external interface of the system. From an end-user's perspective it and describes the functional requirements of the system. To a developer, it gives a clear and consistent description of what the system should do. This model is used and elaborated throughout the

development process. As an aid to the tester, it provides a basis for performing system tests to verify the system. It also provides the ability to trace functional requirements into actual classes and operations in the system and hence helps in identifying any gaps.

Use Diagram for a Library System

As an example, consider the following use case diagram for a library management system. In this diagram, there are four actors namely Book Borrower, Librarian, Browser, and Journal Borrower. In addition to these actors, there are 8 use cases. These use cases are represented by ovals and are enclosed within the system boundary, which is represented by a rectangle. It is important to note that every use case must always deliver some value to the actor.



With the help of this diagram, it can be clearly seen that a Book Borrower can reserve a book, borrow a book, return a book, or extend loan of a book. Similarly, functions performed by other users can also be examined easily.

