UNIT II

Introduction To Requirements Analysis And Specification

Before we start to develop our software, it becomes quite essential for us to understand and document the exact requirement of the customer. Experienced members of the development team carry out this job. They are called as system analysts.

The analyst starts requirements gathering and analysis activity by collecting all information from the customer which could be used to develop the requirements of the system. He then analyzes the collected information to obtain a clear and thorough understanding of the product to be developed, with a view to remove all ambiguities and inconsistencies from the initial customer perception of the problem. The following basic questions pertaining to the project should be clearly understood by the analyst in order to obtain a good grasp of the problem:

• What is the problem?

• Why is it important to solve the problem?

• What are the possible solutions to the problem?

• What exactly are the data input to the system and what exactly are the data output by the system?

• What are the likely complexities that might arise while solving the problem?

• If there are external software or hardware with which the developed software has to interface, then what exactly would the data interchange formats with the external system be

After the analyst has understood the exact customer requirements, he proceeds to identify and resolve the various requirements problems. The

most important requirements problems that the analyst has to identify and eliminate are the problems of anomalies, inconsistencies, and incompleteness. When the analyst detects any inconsistencies, anomalies or incompleteness in the gathered requirements, he resolves them by carrying out further discussions with the end users and the customers.

Software prototype and Specification

Software prototyping refers to the activity of creating prototypes of software applications, i.e., incomplete versions of the software program being developed. It is an activity that can occur in software development. Prototypes are Mockups of an application. Mockups allow users to visualize an application that hasn't yet been constructed. Prototypes help users get an idea of what the system will look like, and make it easier for users to make design decisions without waiting for the system to be built. Major improvements in communication between users and developers were often seen with the introduction of prototypes. Early views of applications led to fewer changes later and hence reduced overall costs considerably.

Prototypes can be flat diagrams (often referred to as wireframes) or working applications using synthesized functionality. Wireframes are made in a variety of graphic design documents, and often remove all color from the design (i.e. use a grayscale color palette) in instances where the final software is expected to have graphic design applied to it. This helps to prevent confusion over the final visual look and feel of the application.

Prototyping process

The process of prototyping involves the following steps:

1. Identify basic requirements: Determine basic requirements including the input and output information desired. Details, such as security, can typically be ignored.

2. Develop Initial Prototype: The initial prototype is developed that includes only user interfaces.

3. Review: The customers, including end-users, examine the prototype and provide feedback on additions or changes.

4. Revise and Enhance the Prototype: Using the feedback both the specifications and the prototype can be improved.

Negotiation about what is within the scope of the contract/product may be necessary. If changes are introduced then a repeat of steps #3 and #4 may be needed.

Types of prototyping: Software prototyping has many variants. However, all the methods are in some way based on two major types of prototyping: Throwaway Prototyping and Evolutionary Prototyping.

Prototyping Methods and Tools

For software prototyping to be effective, a prototype must be developed rapidly so that the customer may assess results and recommend changes. To conduct rapid prototyping, three generic classes of methods and tools are available:

Fourth generation techniques: Fourth generation techniques (4GT) encompass a broad array of database query and reporting languages, program and application generators, and other very high-level nonprocedural languages. Because 4GT enable the software engineer to generate executable code quickly, they are ideal for rapid prototyping.

Reusable software components: Another approach to rapid prototyping is to assemble, rather than build, the prototype by using a set of existing software components. Melding prototyping and program component reuse will work only if a library system is developed so that components that do exist can be cataloged and then retrieved. It should be noted that an existing software product can be used as a prototype for a "new, improved" competitive product. In a way, this is a form of reusability for software prototyping.

Formal specification and prototyping environments: Over the past two decades, a number of formal specification languages and tools have been developed as a replacement for natural language specification techniques. Today, developers of these formal languages are in the process of developing interactive environments that (1) enable an analyst to interactively create language-based specifications of a system or software, (2) invoke automated tools that translate the language-based specifications into executable code, and (3) enable the customer to use the prototype executable code to refine formal requirements.

Advantages of prototyping

1. Reduced time and costs

2. Improved and increased user involvement

Disadvantages of prototyping

1. Insufficient analysis

2. User confusion of prototype and finished system

3. Developer misunderstanding of user objective

4. Developer attachment to prototype

5. Excessive development time of the prototype

6. Expense of implementing prototyping

Best projects to use prototyping: Prototyping is most beneficial in systems that will have many interactions with the users. It has been found that

prototyping is very effective in the analysis and design of on-line systems, especially for transaction processing, where the use of screen dialogs is much more in evidence. The greater the interaction between the computer and the user, the greater the benefit is that can be obtained from building a quick system and letting the user play with it. Systems with little user interaction, such as batch processing or systems that mostly do calculations benefit little from prototyping. Sometimes, the coding needed to perform the system functions may be too intensive and the potential gains that prototyping could provide are too small. Prototyping is especially good for designing good human-computer interfaces.

Throwaway prototyping

It is also called close-ended prototyping or Rapid prototyping. Throwaway refers to the creation of a model that will eventually be discarded rather than becoming part of the final delivered software. After preliminary requirements gathering is accomplished, a simple working model of the system is constructed to visually show the users what their requirements may look like when they are implemented into a finished system. In this approach the prototype is constructed with the idea that it will be discarded and the final system will be built from scratch. The steps in this approach are:

1. Write preliminary requirements
2. Design the prototype
3. User experiences/uses the prototype, specifies new requirements
4. Repeat if necessary
5. Write the final requirements
6. Develop the real products

Evolutionary prototyping

It is also known as breadboard prototyping. The main goal when using Evolutionary Prototyping is to build a very robust prototype in a structured manner and constantly refine it. "The reason for this is that the Evolutionary prototype, when built, forms the heart of the new system, and the improvements and further requirements will be built. When developing a system using Evolutionary Prototyping, the system is continually refined and rebuilt.

This technique allows the development team to add features, or make changes that couldn't be conceived during the requirements and design phase. Evolutionary Prototypes have an advantage over Throwaway Prototypes in that they are functional systems. Although they may not have all the features the users have planned, they may be used on an interim basis until the final system is delivered. In Evolutionary Prototyping, developers can focus themselves to develop parts of the system that they understand instead of working on developing a whole system.

To minimize risk, the developer does not implement poorly understood features. The partial system is sent to customer sites. As users work with the system, they detect opportunities for new features and give requests for these features to developers.

Incremental prototyping

The final product is built as separate prototypes. At the end the separate prototypes are merged in an overall design.

Extreme prototyping

Extreme Prototyping as a development process is used especially for developing web applications. Basically, it breaks down web development into three phases, each one based on the preceding one. The first phase is a static prototype that consists mainly of HTML pages. In the second phase, the screens are programmed and fully functional using a simulated services layer. In the third phase the services are implemented. The process is called Extreme Prototyping to draw attention to the second phase of the process, where a fully functional UI is developed with very little regard to the services other than their contract.