**Software Analysis Concepts and Principles**

The overall role of software in large system is identified during system engineering. However, it’s necessary to take a harder look at software’s role to understand the specific requirements that must be achieved to build high-quality software. That’s the job of software requirements analysis. To perform the job properly you should follow a set of underlying concepts and principles.

**1. Requirements Analysis**

Requirement analysis is a software engineering task that bridges the gap between system level requirements engineering and software design. Requirements engineering activities result in the specification of software’s operational characteristics, indicate software’s interface with other system elements, and establish constraints that software must meet. Requirement analysis allows the software engineer to refine domains that will be treated by software.

Software requirements analysis may be divided into five areas of effort:

(1)     problem recognition,

(2)     evaluation and synthesis,

(3)     modeling,

(4)     specification, and

(5)     review.

The analyst studies the *system specification* and the *software Project Plan*. It is important to understand software in a system context and to review the software scope that was used to generate planning estimates. Problem evaluation and solution synthesis is the next major area of effort for analysis. The analyst must define all externally observable data objects, evaluate the flow and content of information, define and elaborate all software functions, understand software behavior in the context of events that affect the system, establish system interface characteristics, and uncover additional design constraints.

Throughout evaluation and solution synthesis, the analyst’s primary focus is on “what” not “how”. What data does the system produce and consume, what functions must the system perform, what behavior does the system exhibit, what interfaces are defined and what constraints apply?

**2. Requirements Elicitation for Software**

Before requirements can be analyzed , modeled, or specified they must be gathered through an elicitation process.

**2.1 Initiating the Process**

The first meeting between a software engineer and the customer can be liked to the awkwardness of a first date between two adolescents. Communication must be initiated by asking *context-free questions*. That is a set of questions that will lead to basic understanding of the problem, the people who want a solution that will lead to basic understanding of the problem, the people who want a solution, the nature of the solution that is desired, and the effectiveness of the first encounter itself.

* Who is behind the request for this work?
* Who will use the solution?
* What will be the economic benefit of a successful solution?
* Is there another source for the solution that you need?

The next set of questions enables the analyst to gain a better understanding of the problem and the customer to voice his or her perceptions about a solution:

* How would you characterize “good” output that would be generated by a successful solution?
* What problem(s) will this solution address?
* Can you show me the environment in which the solution will be used?
* Will special performance issues or constrains affect the way the solution is approached?

**2.2 Facilitated Application Specification Techniques**

Customers and software engineers have an unconscious “us and them” mind-set. With these problems in the mind that a number of independent investigators have developed a team-oriented approach to requirements gathering that is applied during early stages of analysis and specification. Called*facilitated application technique (****FAST****)*. Basic guidelines for this technique are:

* A meeting is conducted at a neutral site and attended by both software engineers and customers.
* Rules for preparation and participation are established.
* An agenda is suggested that is formal enough to cover all important points but informal enough to encourage the free flow of ideas.
* A “facilitator” controls the meeting.
* A “definition mechanism” is used
* The goal is to identify the problem, propose elements of the solution, negotiate different approaches, and specify a preliminary set of solution requirements in an atmosphere that is conductive to the accomplishment of the goal.

Initial meeting between the developer and customer occur and basic questions and answers help to establish the scope of the problem and the over all perception of a solution. The product request distributed to all attendees before the meeting date. The FAST team is composed of representatives from marketing, software and hardware engineering, and manufacturing. As the FAST meeting begins, the first topic of discussion is the need and justification for the new product – everyone should agree that the product justified. Once agreement has been established, each participant his or her list for discussion.

After individual lists are presented in one topic area, a combined list is created by the group. The combined list eliminates redundant entries, adds any new ideas that come up during the discussion, but does not delete anything. The combined list is shortened, lengthened, or reworded to properly reflect the product or system to be developed. The objective is to develop a *consensus list* in each topic area. Each sub team presents its mini-specs to all FAST attendees for discussion. After the mini-specs are completed, each FAST attendee makes a list of *validation criteria*for the product or system and presents his or her to the team.

**2.3 Quality Function Deployment**

*Quality function deployment (*QFD) is a quality management technique that translates the needs of the customer into technical requirements for software. QFD identifies three types of requirements:

**Normal requirements.** The objectives and goals that are stated for a product or system during meeting with customer. If these requirements are present, the customer is satisfied.

**Expected requirements.** These requirements are implicit to the product or system and may be so fundamental that the customer does not explicitly state them. Their absence will be a cause for significant dissatisfaction.

**Exciting requirements.** These features go beyond the customer’s expectations and prove to be very satisfying when present.

*Functional deployment* is used to determine the value of each function that is required for the system. *Information deployment* identifies both the data objects and events that the system must consume and produce. These are tied to the functions. Finally, *task deployment*examines the behavior of the system or product within the context of its environment. *Value analysis* is conducted to determine the relative priority of requirements determined during each of the three deployments.

**2.4. Use-Cases**

As requirements are gathered as part of informal meetings, a software engineer can create a set of scenarios that identify a thread of usage for the system to be constructed. To create a use-case, the analyst must first identify the different types of people play as the system operates. Defined somewhat more formally an actor is anything that communicates with the system or product and that is external to the system itself.

It’s most important to note that an actor and a user are not the same thing. An actor represents a class of external entities that play just one role. Once actors have been identified, use-case can be developed. The use-case describes the manner in which an actor interacts with the system. The use-case should be answer below questions:

* What main tasks or functions are performed by an actor?
* What system information will the actor acquire, produce, or change?
* Will the actor have to inform the system about changes in the external environment?
* What information does the actor desire from the system?
* Does the actor wish to be informed about unexpected changes?

In general, use-case is simply a written narrative that describes the role of an actor as interaction with the system occurs.

**3. Analysis Principles**

Over the past two decades, a large number of analysis modeling methods have been developed. Investigators have identified analysis problems and their causes and have developed a variety of notations and corresponding sets of heuristics to overcome them. Each analysis method has a unique point of view.

* The information domain of a problem must be represented and understood.
* The functions that the software is to perform must be defined.
* The behavior of the software must be represented.
* The models that depict information, function, and
* The models that depict information function and behavior must be partitioned in a manner that uncovers details in a layered fashion.
* The analysis process should move from essential information toward implementation detail.

In addition to these operational analysis principles for requirements engineering:

* *Understand the problem before you begin to create the analysis model.*
* *Develop prototype that enable a user to understand how human/machine interaction will occur.*
* *Record the origin of and the reason for every requirement.*
* *Use multiple views of requirements.*
* *Rank requirements*.
* *Work to eliminate ambiguity*

**SOFTWARE PROTOTYPING**

Rapid software development to validate requirements

Objectives

* + To describe the use of prototypes in different types of development project
  + To discuss evolutionary and throw-away prototyping
  + To introduce three rapid prototyping techniques - high-level language development, database programming and component reuse
  + To explain the need for user interface prototyping

Prototyping is the rapid development of a system

The principal use is to help customers and developers understand the requirements for the system

* + Requirements elicitation – Users can experiment with a prototype to see how the system supports their work
  + Requirements validation – The prototype can reveal errors and omissions in the requirements

Prototyping can be considered as a risk reduction activity

**Prototyping benefits**

* Misunderstandings between software users and developers are exposed
* Missing services may be detected and confusing services may be identified
* A working system is available early in the process
* The prototype may serve as a basis for deriving a system specification
* The system can support user training and system testing

**Prototyping in the software process**

* Evolutionary prototyping
  + An initial prototype is produced and refined through a number of stages to the final system
* Throw-away prototyping
  + A prototype is produced to help discover requirements problems and then discarded
  + The system is then developed using some other development process

Prototyping objectives

* The objective of *evolutionary prototyping* is to deliver a working system to end-users
  + The development starts with those requirements which are best understood.
* The objective of *throw-away prototyping* is to validate or derive the system requirements
  + The prototyping process starts with those requirements which are poorly understood

**Approaches to prototyping**



**Evolutionary prototyping**

* Must be used for systems where the specification cannot be developed in advance
  + E.g., AI systems and user interface systems
* Based on techniques which allow rapid system iterations
* Verification is impossible as there is no specification
* Validation means demonstrating the adequacy of the system



**Evolutionary prototyping advantages**

Accelerated delivery of the system

* + Rapid delivery and deployment are sometimes more important than functionality or long-term software maintainability

User engagement with the system

* + Not only is the system more likely to meet user requirements, they are more likely to commit to the use of the system
* Specification, design and implementation are inter-twined
* The system is developed as a series of increments that are delivered to the customer
* Techniques for rapid system development are used such as CASE tools and 4GLs
* User interfaces are usually developed using a GUI development toolkit

**Evolutionary prototyping problems**

Management problems

* + Existing management processes assume a waterfall model of development
  + Specialist skills are required which may not be available in all development teams

Maintenance problems

* + Continual change tends to corrupt system structure so long-term maintenance is expensive

Contractual problems

**Prototypes as specifications**

* Some parts of the requirements may be impossible to prototype
  + E.g., safety-critical functions
* An implementation has no legal standing as a contract
* Non-functional requirements cannot be adequately tested in a system prototype
* Incremental development
* System is developed and delivered in increments after establishing an overall architecture
* Requirements and specifications for each increment may be developed
* Users may experiment with delivered increments while others are being developed
  + These serve as a form of prototype system
* Intended to combine some of the advantages of prototyping
  + More manageable process
  + Better system structure

**Incremental development process**

****

**Throw-away prototyping**

* Used to reduce requirements risk
* The prototype is developed from an initial specification, delivered for experiment then discarded
* The throw-away prototype should NOT be considered as a final system
  + Some system characteristics may have been left out
  + There is no specification for long-term maintenance
  + The system will be poorly structured and difficult to maintain



**Rapid prototyping techniques**

* Various techniques may be used for rapid development
  + Dynamic high-level language development
  + Database programming
  + Component and application assembly
* These techniques are often used together
* Visual programming is an inherent part of most prototype development systems

**Dynamic high-level languages**

* Languages which include powerful data management facilities
* Need a large run-time support system. Not normally used for large system development
* Some languages offer excellent UI development facilities
* Some languages have an integrated support environment whose facilities may be used in the prototype