Quest Journals
Journal of Software Engineering and Simulation
Volume1 ~ Issue1 (2013) pp: 13-18
ISSN(Online) :2321-3795 ISSN (Print):2321-3809
www.questjournals.org

Research Paper

Methods of Software Analysis and Design

Komal Sachdeva[1]

Received 21 September, 2013; **A**ccepted 05 October, 2013; © The author(s) 2013. Published with open access at **www.questjournals.org**

ABSTRACT : *The paper describes the process of the software analysis and design in detail. Why the analysis is required and what are the benefits of analysis, side by side how analysis and design processes are related. The various analysis methods along with their use in software architecture is explained. The design process along with the various design methodologies is well explained here. The Design Concepts with all the attributes required and the design concepts for the methods of the software design are explained here. The views in UML are also explained here. Structured method, Object Oriented Approach, Data Flow Diagram, Algorithmic Designs are explained here. SAAMER ESAAMER analysis approaches are also explained here.*

Keywords: *Algorithmic Design, Object Oriented Design, SAAMER, Software Architecture, UML*

I. INTRODUCTION

Software Analysis is the process by which customer needs are understood and documented. Software Analysis is the process of prediction of the quality of a system before it has been built and to identify the potential risks and the verification of the quality requirements that have been addressed in the design. Software Analysis process expresses "What" is to be built and not" How" is to be built. The Software Requirement Analysis depends upon C- and D- requirements [1] - Customer wants and needs, expressed in the language understood by the customer. D- For the developers, it may be formal. C- Requirements make use of number of processes like: Use Cases Diagrams, Data Flow Diagrams, State Transition Diagrams, and User Interfaces .D-Requirements organize the D- requirements, make Sequence diagrams for the use cases, and validate with the customer and then release. D- Requirements should have various properties like: Traceability, Testability, Categorization, Prioritization Strategy, Completeness and Error Conditions. Analysis is the investigation of the problem (what). Design is the conceptual solution to fulfill the requirements; how will the system do what it is intended to do. Object Oriented Analysis considers a problem from the perspective of objects (real world things, concepts etc). Object oriented design defines the solution as a collection of software objects (allocating responsibilities to the objects). Like Object Oriented Analysis, in the case of Library Information Systems, , one would *find* concepts like: *book, library.* In Object Oriented Design, , there is an emphasis on designing the software objects , finally, these objects are implemented in some programming language; *Book*, may have a method name *print.*

1.1 ANALYSIS AND DESIGN MODELS

Analysis model is related to the investigation of the domain and the problem space e.g. Use Case Diagrams, Design model is related to the solution. e.g.: Class Diagrams. UML is unified modeling language. Parts of UML are: Views: it shows the different aspects of the system that are modeled, and links the modeling language to the method chosen for the development. Diagrams: it graphs that describe the contents in a view. Model Elements: these are the concepts used in a diagram.
Views in UML: As shown in the Fig.1:
Use Case View: The view showing the functionality of the system as perceived by the external actors.
Logical View: This is the view showing how the functionality is designed inside the system, in terms of the static structure and dynamic behavior.
Component View: This is the view which shows the organization of the code components.
Concurrency View: This is the view which shows the concurrency of the system.
Deployment View: This is the view which shows the deployment of the system in terms of the physical architecture.

*Corresponding Author: Komal Sachdeva
Department of Computer Science and Engineering, Faculty of CSE,
Manav Rachna College of Engineering, Faridabad

UML can best be defined in terms of the following Model Elements:

- Class
- Object
- State
- Use Case
- Interface
- Association
- Link
- Package

UML diagrams can best be described with the following diagrams:

- Use Case Diagram: Deals with external interaction with the actors
- Class/Object Diagram: it captures static structural aspects, objects and relationships.
- State Diagram: it describes the dynamic state behavior.
- Sequence Diagram: it models object interaction over the time.
- Collaboration Diagram: it models components interaction and structural dependencies.
- Activity Diagram: it models the object activities.
- Deployment Diagram: it models the physical architecture
- Component Diagram: it models the software architecture.

1.2 Software Analysis Properties

The following are the properties of the Software Analysis which have to be fulfilled:

- The method should be able of being applied at all the stages of the system lifecycle
- The method should not results in an increase of the work required at the subsequent later stages of the design
- The Analysis should help in the design phase by doing all the comparisons of alternatives at the initial levels.
- Analysis should have a high degree of failure confidence
- The Analysis should allow the design to be checked incrementally as the process proceeds.
- The result of Analysis should be used at any of the another stages of the Software Development Life Cycle.

1.3 Software Analysis Methods

There are various methods available for doing the software analysis [2]:

a) ***Scenario-Based Architecture Analysis Method (SAAM)*****:** This method deals with the better understanding of the general architectural concepts. Against the documents which are used for describing the desired properties of an application, basic architectural and principles are verified. In this method, the scenarios represent the foundation for showing the properties of Software Architecture. Scenarios show two things, the kinds of activities that the system must support and, the kind of the predictable changes that will be made to the system. Based upon the requirement of modification by a scenario, they are categorized as direct scenarios, else as indirect scenarios. The activities of the method are shown in Fig.2 as the main inputs of SAAM. Here, the main inputs are: Problem Description, Requirements Statement, and Architecture Descriptions.

b) ***Extending SAAM by Integration in the Domain (ESAAMI):*** By integrating the SAAM in the domain-specific and reuse-based development process, ESAAMI is achieved. Fig.3 describes the main inputs of ESAAMI and the relationships between them.

c) ***Software Architecture Analysis Method for Evolution and Reusability (SAAMER):*** To support the quality objectives and risk levels, SAAMER is used. The framework of SAAMER consists of mainly four activities: Gathering Information about stakeholders, Software Architecture, Quality, and Scenarios. So as to cover each objective, the scenarios are identified and clustered. Quality Function Deployment (QFD) is used to validate the balance of scenarios with respect to the objective. A flow of matrices is generated to show the relational strength between the stakeholders and the architectural objectives. Now, quality attributes are translated to scenarios. And corresponding to each of the quality attribute, an imbalance factor is calculated. If the factor comes out to be 1, more scenarios will be required to address the attribute concurrent to the stakeholder, Software Architecture and quality importance.

d) ***The Architecture Trade-Off Analysis Method (ATAM):*** The result of individual attributes on architectural analysis gives ATAM. This method is considered as the spiral model of analysis and design. When there are multiple competing quality attributes, to understand the Software Architecture capability, ATAM is

*Corresponding Author: Komal Sachdeva

Department of Computer Science and Engineering, Faculty of CSE,
Manav Rachna College of Engineering, Faridabad

used. ATAM activities that consider scenarios are: Scenario brainstorming, Scenario coverage checking, Scenario grouping and prioritization, Map high priority scenario onto the architecture.

II SOFTWARE DESIGN

The process of implementation of software solutions to one or more set of problems is called as software design. Software requirement analysis is one of the most important parts of software design. Software Design may be platform independent or platform dependent, depending upon the technology used for the design. The main difference between the software analysis process and software design process is that **the** output of software analysis consists of smaller problems to solve. The analysis should remain same even if it is designed by different team members or the groups. The focus of the design is on the capabilities, and there can be multiple designs for the same problem.

2.1 Design Concepts

The software design is described with the help of the following concepts:

- *Abstraction* - The process of generalization by reducing the information content of a concept and to retain only relevant information is called as the abstraction.
- *Refinement* - The process of breakdown of large set of instructions into small instructions is called as the refinement. Abstraction and Refinement are complementary concepts.
- *Modularity* - Software architecture is divided into small components which are called as the modules.
- *Software Architecture* –Maintaining the conceptual integrity of a system,a structure of the software is developed.
- *Control Hierarchy* –Based on the hierarchy of control, the program structure is organized.
- *Structural Partitioning* - The program structure is divided intotwo partitions: Horizontal Partition and Vertical Partition In Horizontal partition,separate branches of modular hierarchy for each major program function is defined. In Vertical partitioning,control and work is distributed top down in the program structure.
- *Data Structure* – The representation of the logical relationship among individual elements of data.
- *Software Procedure* - It focuses on the processing of each modules individually
- *Information Hiding* – Modules of the information are specified and designed in such a way that information contained within a module is not accessible to other modules that have no need for such information

2.2 Design considerations

Various design considerations are described below:

- *Compatibility* –Compatibilty is the ability of the software to operate with other products that are designed for interoperability with another product. For example, a piece of software may be backward-compatible with an older version of itself.
- *Extensibility* –Addition of new capabilities to the software without major changes to the underlying architecture.
- *Fault-tolerance* – In case of failure,the software is resistant to and able to recover.
- *Maintainability* – The ease of bug fixing or functional modifications to the existing software.
- *Modularity* – All the components of the software should be well defined.. That leads to better maintainability. The components are implemented and tested individually before being integrated to form a desired software system.
- *Reliability* – The ability of the software to perform a required function under stated conditions for a specified period of time.
- *Reusability* – The ability ofthe software to add further features and modification with slight or no modification.
- *Robustness* - The software is able to operate under stress or tolerate unpredictable or invalid input.
- *Security* - The software is able to withstand hostile acts and influences.
- *Usability* - The software user interface must be usable for its target user/audience. Default values for the parameters must be chosen so that they are a good choice for the majority of the users.
- *Performance* - The software performs its tasks within a user-acceptable time. The software does not consume too much memory.
- *Scalability* - The software adapts well to increasing data or number of users.

*Corresponding Author: Komal Sachdeva

Department of Computer Science and Engineering, Faculty of CSE,
Manav Rachna College of Engineering, Faridabad

2.2 Modeling language

- A modeling language is any artificial language that can be used to express information or knowledge or systems in a structure that is defined by a consistent set of rules. The rules are used for interpretation of the meaning of components in the structure. A modeling language can be graphical or textual. Examples of graphical modeling languages for software design are:
- *Business Process Modeling Notation* (BPMN) is an example of a Process Modeling language.
- *EXPRESS and EXPRESS-G (ISO 10303-11)* is an international standard general-purpose data modeling language.
- *Extended Enterprise Modeling Language (EEML)* is commonly used for business process modeling across a number of layers.
- *Flowchart* is a schematic representation of an algorithm or a step-wise process,
- *Fundamental Modeling Concepts (FMC)* modeling language for software-intensive systems.
- *IDEF* is a family of modeling languages, the most notable of which include IDEF0 for functional modeling, IDEF1X for information modeling, and IDEF5 for modeling ontologies.
- *Jackson Structured Programming (JSP)* is a method for structured programming based on correspondences between data stream structure and program structure
- *LePUS3* is an object-oriented visual Design Description Language and a formal specification language that is suitable primarily for modelling large object-oriented (Java, C++,C#) programs and design patterns.
- *Unified Modeling Language (UML)* is a general modeling language to describe software both structurally and behaviorally. It has a graphical notation and allows for extension with a Profile (UML).
- *Alloy (specification language)* is a general purpose specification language for expressing complex structural constraints and behavior in a software system. It provides a concise language based on first-order relational logic.
- *Systems Modeling Language (SysML)* is a new general-purpose modeling language for systems engineering.

III. DESIGN METHODOLOGIES

There are various design methodologies available. Some of which are:

- Structured Methods
- Data Flow Model
- Entity Relationship Notation
- Object Oriented Design
- Algorithmic Design

***Structured Methods*:** Here program is broken into functions and subroutines.There is only a single entry point and a single exit point for any function or routine.Here. program structure is derived from the data structure.Design is dependent upon temporal ordering of processing phases e.g initialize, process,cleanup.Changes in data representationsripple through entire structuredue to the lack of information hiding.

***Data Flow Model*:** Diagramatically representation of the data and the transfer of the information between the inputs and the outputs are represented as: Rounded rectangles represent functions which transforms inputs to outputs, the transformation name indicates its functions, Rectangles represent data stores, Circles represent user interactions with the system which provides input and receive output, Arrows show the direction of the data flow.

Entity Relationship Notation: The data modelling technique that creates a graphical representation of the entities, and the relatioship between those entities is called as an Entity Relationship Notation.

The three main components of an ERN are:

- The *entity* is a person, object, place or event for which data is collected. For example, if you consider the information system for a business, entities would include not only customers, but the customer's address, and orders as well. The entity is represented by a rectangle and labeled with a singular noun.
- The *relationship* is the interaction between the entities. In the example above, the customer *places* an order, so the word "places" defines the relationship between that instance of a customer and the order or orders that they place. A relationship may be represented by a diamond shape, or more simply, by the line connecting the entities. In either case, verbs are used to label the relationships.
- The *cardinality* defines the relationship between the entities in terms of numbers. An entity may be *optional*: for example, a sales rep could have no customers or could have one or many customers;

or *mandatory*: for example, there must be at least one product listed in an order. There are several different types of cardinality notation; *crow's foot notation*, used here, is a common one. In crow's foot notation, a single bar indicates *one*, a double bar indicates *one and only one* (for example, a single instance of a product can only be stored in one warehouse), a *circle* indicates zero, and a *crow's foot* indicates many. The three main cardinal relationships are: one-to-one, expressed as 1:1; one-to-many, expressed as 1: M; and many-to-many, expressed as M: N.

Object Oriented Design: It is based on the idea of information hiding. Here, System is viewed as a set of interacting objects, with their own private state. In this, objects communicate by calling on services offered by other objects rather than sharing variables, and this reduces the overall system coupling, message passing model allows objects to be implemented as concurrent processes either as Passive Objects or the Active Objects.

Algorithmic Design: This method is based on the Top-Design approach, which is based on the functions performed by the system. It generally follows a "divide and conquer" strategy based on functions i.e. more general functions are iteratively/recursively decomposed into more specific ones.

IV. FIGURES

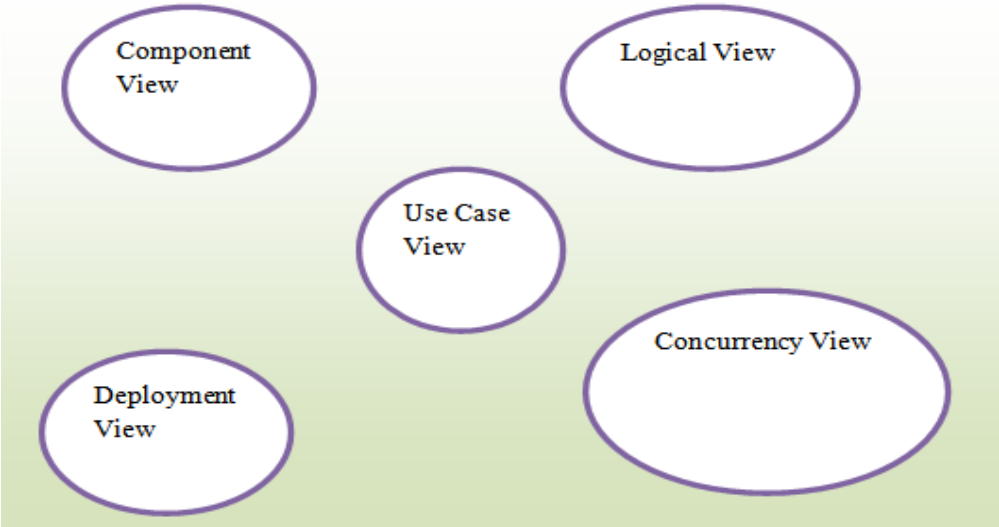

Fig.1 UML Views

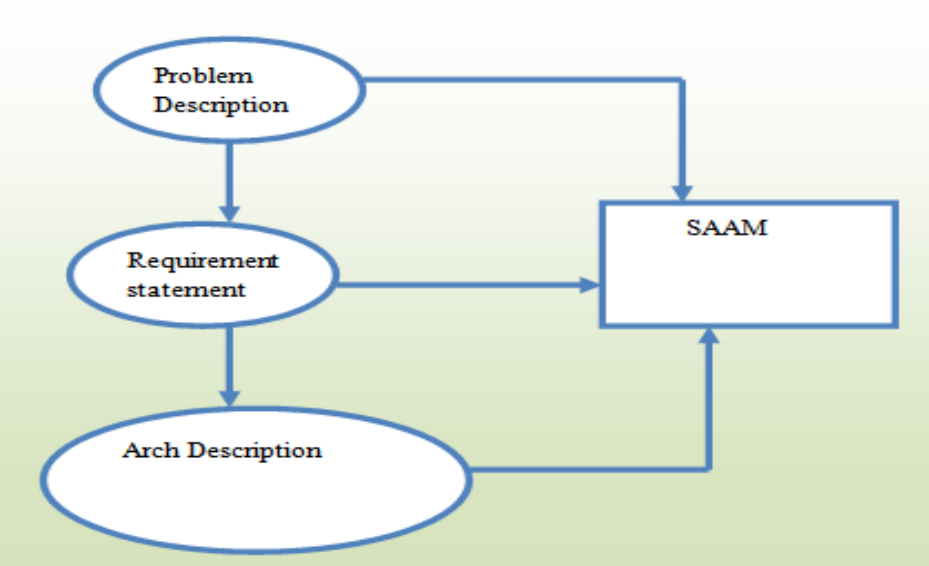

Fig.2: SAAM Inputs and Activities

*Corresponding Author: Komal Sachdeva
Department of Computer Science and Engineering, Faculty of CSE,
Manav Rachna College of Engineering, Faridabad

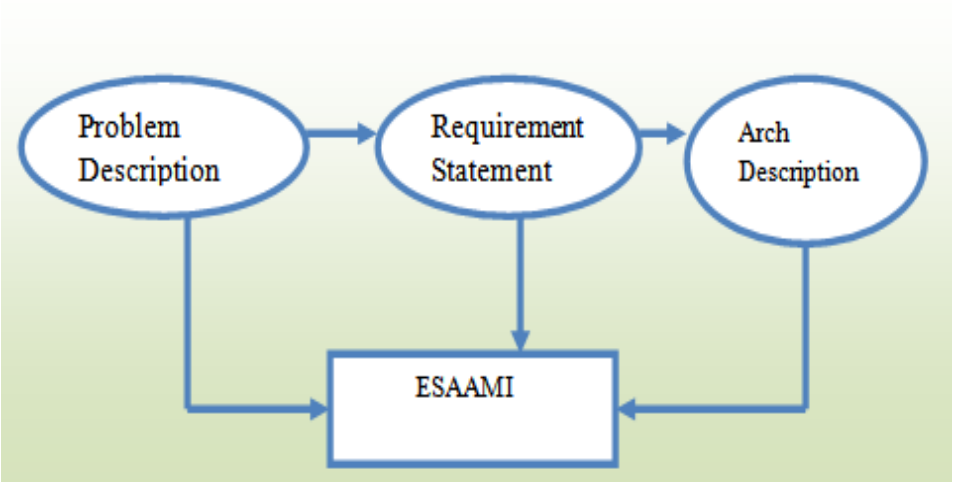

Fig.3 ESAAMI Inputs

V. CONCLUSION

The paper concludes the importance of analysis and design in the case of software architecture. The various analysis and design methods are explained here. The dependencies of analysis and design are explained here, the paper results as an software analysis methods and the design methods. The future work of the paper will be the strategies about where to use which design method.

REFERENCES

Theses:

[1]. BITS/C461 IS/C341 Software Engineering

Journal Papers:

[1]. Liliana Dobrica and Eila Niemela ,A Survey on Software Architecture Analysis Methods, *IEEE Transactions on Software Engineering, Vol 28, No. 7, July 2002*.

*Corresponding Author: Komal Sachdeva
Department of Computer Science and Engineering, Faculty of CSE,
Manav Rachna College of Engineering, Faridabad