**SRS (SOFTWARE REQUIREMENT SPECIFICATIONS)**

**1. Introduction**

**1.1 Purpose**

* The objective of our project is to provide information to dog lovers about ideal and best products for their pet dogs
* We want to facilitate by providing them description about different products and their quality. • One of the biggest objective is to aware the doglovers about different products of different quality so they can choose the right product.
* Implement the various concepts of Android programming together within an application that becomes helpful for intended users.
* Developing an Android based application for the dog lovers and system having user friendly interface. • Read on for help with choosing a dog’s food, training, first aid and vet visits, walks, dog sitters, GPS trackers.
* To make life better. Even better is when they combine to make everything easier, because dog lovers don’t need to go to market for dog’s food,chains,calcium bones,vaxines.they get these all stuff online.

**1.2 Document Conventions**

* The synopsis shall be computer typed (English- British, Font -Times Roman, Size-12 point) and printed on A4 size paper
* The Synopsis shall be typed on one side only with double space with a margin 3.5 cm on the left, 2.5 cm on the top, and 1.25 cm on the right and at bottom.
* In the synopsis, the title page [Refer sample sheet (inner cover)] should be given first. This should be followed by index, notations/nomenclature.
* The diagrams should be printed on a light/white background, Tabular matter should be clearly arranged. Decimal point may be indicated by full stop(.)The caption for Figure must be given at the BOTTOM of the Fig. and Caption for the Table must be given at the TOP of the Table.

**1.3 Intended Audience and Reading Suggestions**

* Anyone with a Dog and needs the product to take care of their needs.
* Audience in the ecommerce industry works differently from that of real world. Merchants don’t interact with customers as they would at a physical shop. Hence, judgment on one’s behavior and personality varies at almost every encounter.

**1.4 Product Scope**

The app uses android technology which has evergreen scope. The app obviously has a bright future scope as there is test which includes different level and type of questions. Moreover in Future, one can see his/her earlier records regarding the payments and can checkout the product with the use of image processing, by simply placing the camera on previous buyed product. The platform used is android. Nowadays Android has become very popular which is an open-source, Linux-based operating system mainly designed by Google for smart-phones and tablets. Many mobile Apps development industries are considering Android Application Development as one of the best business opportunities, for this they need to hire a lot of knowledgeable mobile application developer in future. This adds a big sign of scope of mobile Apps in future. In the current job market of mobile application development, the need for inventive App developers is huge and still increasing. Android Apps development can also be taken up as a part time job. You can create your own applications at home and submit it to the Google Play store which can be downloaded by smart-phone users.

**1.5 References**

Tutorialspoint.com

IEEE Template for System Requirement Specification Documents:

https://goo.gl/nsUFwy

**2. Overall Description**

**2.1 Product Perspective**

Saphar.com was developed for everyone who is interested in travelling and wants either to just look among various packages available for travelling. It is an open source project and allows users to give back feedbacks. It was developed to run on Windows, Mac OS X and Linux.

**2.2 Product Functions**

HOME PAGE:

It describes the details of various travel agents and packages offered by them. Here, user can fill their destinations as well as departure and return date.

GALLERY:

This option illustrates the different images of various tourist destinations.

REGISTER AND LOGIN:

The register link allows new users to create username and password to access the information.

In the login page, user can login and choose from various packages available.

User can also save username and password.

FEEDBACK:

The logged user can provide feedback by filling a form o show the satisfaction.

CONTACT:

Here user can send text messages to the admin regarding any issue.

**2.3 User Classes and Characteristics**

* Typical Users, such as individuals, who want to use Saphar.com for travelling.
* Programmers who are interested in working on the project by further developing it or fix existing bugs.

**2.4 Operating Environment**

 Windows 2000

 Windows XP

 Windows Vista

 Windows 7

 Windows 8

 Windows 10

 Mac OS X

 Linux

**2.5 Design and Implementation Constraints**

Saphar.com is developed in PHP, it uses MySQL and Xampp for its visualization and data storage & retrieval. It uses a modular design where every feature is wrapped into a separate module and the modules depend on each other.

**2.6 Assumptions and Dependencies**

Saphar.com is developed in PHP & MySQL and therefore requires Xampp to be installed on the user’s system. The stable version of Saphar.com requires Xampp version 3.2.2 or higher. This applies to Windows and Linux users.

**3. EXTERNAL INTERFACE REQUIREMENTS**

**3.1 User Interfaces**

User interface design (UI) or user interface engineering is the design of userinterfaces for machines and software, such as computers, home appliances, mobiledevices, and other electronic devices, with the focus on maximizing usability and the userexperience. The goal of user interface design is to make the users interaction as simple andefficient as possible, in terms of accomplishing user goals (user-centered design).

Good user interface design facilitates finishing the task at hand without drawingunnecessary attention to itself. Graphic design and typography are utilized to support itsusability, influencing how the user performs certain interactions and improving theaesthetic appeal of the design; design aesthetics may enhance or detract from the ability ofusers to use the functions of the interface.

The design process must balance technical functionality and visual elements (e.g.,mental model) to create a system that is not only operational but also usable and adaptableto changing user needs.

In the proposed project we have used Php as well as bootstrap to design and develop anattractive as well as easy to use interface. The interface designed is a graphical userinterface.

**3.2 Hardware Interfaces**

The minimum hardware requirements of website are an 80 GB hard disk and 512 MB of RAM. Also, it require Network Interface Card of 32 bit and intel core i5 processor.

**3.3 Software Interfaces**

• Adaptive Project Framework

In this methodology, the project scope is a variable. Additionally, the time and the cost are constants for the project. Therefore, during the project execution, the projectscope is adjusted in order to get the maximum business value from the project.

• Agile Software Development

Agile software development methodology is for a project that needs extreme agilityin requirements. The key features of agile are its short-termed delivery cycles(sprints), agile requirements, dynamic team culture, less restrictive project controland emphasis on real-time communication.

• Crystal Methods

In crystal method, the project processes are given a low priority. Instead of theprocesses, this method focuses more on team communication, team member skills,people and interaction. Crystal methods come under agile category.

• Extreme Programming (XP)

Lowering the cost of requirement changes is the main objective of extremeprogramming. XP emphasizes on fine scale feedback, continuous process, sharedunderstanding and programmer welfare. In XP, there is no detailed requirementsspecification or software architecture built.

• Feature Driven Development (FDD)

This methodology is more focused on simple and well-defined processes, shortiterative and feature driven delivery cycles. All the planning and execution in thisproject type take place based on the features.

• Information Technology Infrastructure Library (ITIL)

This methodology is a collection of best practices in project management. ITILcovers a broad aspect of project management which starts from the organizationalmanagement level.

• Lean Development (LD)

Lean development focuses on developing change-tolerance software. In this method,satisfying the customer comes as the highest priority. The team is motivated toprovide the highest value for the money paid by the customer.

• Scrum

This is an agile methodology. The main goal of this methodology is to improve teamproductivity dramatically by removing every possible burden. Scrum projects aremanaged by a Scrum master.

• Spiral

Spiral methodology is the extended waterfall model with prototyping. This methodis used instead of using the waterfall model for large projects.

• Systems Development Life Cycle (SDLC)

This is a conceptual model used in software development projects. In this method,there is a possibility of combining two or more project management methodologiesfor the best outcome. SDLC also heavily emphasizes on the use of documentationand has strict guidelines on it.

• Waterfall (Traditional)

This is the legacy model for software development projects. This methodology hasbeen in practice for decades before the new methodologies were introduced. In thismodel, development life cycle has fixed phases and linear timelines. This model isnot capable of addressing the challenges in the modern software developmentdomain.

Saphar.com requires Xampp to be installed on the system, more specifically Xampp version 3.2.2 or more for its latest release. Additional information can be found on section 2.7 of this document.

It can be connected with a MySQL database to import any data.

**3.4 Communications Interfaces**

Saphar.com requires an internet connection to install new plugins, update already installed ones and update some of its components.

**4. SYSTEM FEATURES**

HOME PAGE:

It describes the details of various travel agents and packages offered by them. Here, user can fill their destinations as well as departure and return date.

GALLERY:

This option illustrates the different images of various tourist destinations.

REGISTER AND LOGIN:

The register link allows new users to create username and password to access the information.

In the login page, user can login and choose from various packages available.

User can also save username and password.

FEEDBACK:

The logged user can provide feedback by filling a form o show the satisfaction.

CONTACT:

Here user can send text messages to the admin regarding any issue.

**5. OTHER NON-FUNCTIONAL REQUIREMENTS**

**5.1 Performance Requirements**

The minimum hardware requirements of website are a 80 GB hard disk and 512 MB of RAM. Also, it require Network Interface Card of 32 bit and intel core i5 processor. Saphar.com requires Xampp to be installed on the system, more specifically Xampp version 3.2.2 or more for its latest release.

**5.2 Safety Requirements**

To ensure that no one of websites users loses any data while using it (due to a crash or a bug of some kind) the developer team updates Xampp regularly. There is a contact section available where users can report any bugs they have encountered so that the developers can fix it in the next release.

**5.3 Security Requirements**

Saphar.com has security requirements and thus users need to have authorized password to access and use any of the available packages.

**5.4 Software Quality Attributes**

Saphar.com provides the users with both simple and advanced features. Due to its well designed and easy to use interface it can be used by both experts and typical users. However, users must already have a basic knowledge of packages before using it.

To specify the functional and non-functional requirements with their validation process.

**1. GENERAL INFORMATION**

**1.1 Purpose**

* To make it convenient for users to book their tour online instead of visiting agency.
* Saves time and money.
* Provides a customized view of the tour to users.
* The website is easy and flexible to use.
* Automates the manual booking, payment and customization process.

**1.2 Scope**

The most important part of the online travel agency project is its database. The database is full of Buses, Trains, Airplane timings and availability.Online travel agency is a web based project where a user may search and apply for a travel service or package. The system allows the user to check various travel destinations and choose his destination accordingly. The software system checks for user choice and then queries the database for various available mediums to travel to that destination. The system then loads all that data and puts those choices in front of the user. The user can now choose various ways to reach his destination. When the user chooses the Bus, train or Airplane option, the system also allows the user to book tickets to the destination for the desire day and timings. Thus this software system automates the working of a travel agency and allows users to check and book his holidays online through this website.

**1.3 Project References**

• [https://it.gndec.ac.in/Report-format- and-Guidelines.pdf](https://it.gndec.ac.in/Report-format-%20and-Guidelines.pdf)

• <https://thecreatingexperts.com/wp-content/uploads/2014/06/frtemplate.doc>

**2. CURRENT SYSTEM SUMMARY**

* All Work are done Manually.
* In Manual Booking System Customer has to go to the Travelling office.
* Ask Inquiry for Travelling then Book ticket Finally Pay Payment & Collect Receipt.
* Difficult To Maintain the Customer Details of Package and Payment Receipt in Register.
* They Register Tour Package in the notebook.
* Add advertisement in Local newspaper or Local Market.
* Use Travelling Facility For the Limited Area or Person.

**2.1 Background**

This application is develop to provide best travelling services to the customers and travel agents. We have developed tours and travel management system to provide a search platform where a tourist can find their tour places according to their choices. This system also helps to promote responsible and interesting tourism so that people can enjoy their holidays at their favorable places. This system also helps to develop tourism with different cultures so that they enrich the tourism experience and build pride. We develop this system to create and promote forms of tourism that provide healthy interaction opportunities for tourists and locals and increase better understanding of different cultures, customs, lifestyles, traditional knowledge and believes. This system also provide a better way to connect with various events.

Also, the system should be user friendly and Window based so that no specialized and extensive learning is required. Simple knowledge of the computer is sufficient for operating the system successfully.

**2.2 System objectives and current functionality**

* To Create Web Based Application For our Organization.
* To Provide Search Facility For Customer.
* To Generate Different Types of Reports.
* To Provide the online Ticket Booking and online Payment Facility For Customer.
* To Provide package Details.

**2.3 Current methods and procedures**

There are five phases in this model and the first phase is the planning stage. The planning stage determines the objectives of the project and whether the project should be given the green light to proceed. This is where the proposal submission comes into picture. After obtaining the approval, the next phase is analysis. Gathering and analyzing the system and user requirements is essential for entry to the design step. With the user requirements gathering completed, there is a need to prepare the resources for the project. Be it software or hardware components, careful consideration and selection is to be taken care at this stage. The decision on the appropriate resources to be used is further elaborated under the subsections below. The next step is to design the system and database structure. Results from the analysis and preparation that were concluded from the previous stage are put into action. With the user requirements in mind, the flow of the system is planned and the user interface is designed to suit their easy navigation needs. In addition, the number of tables, attributes, primary and unique keys of the database is listed. After completing the design, actual coding begins. Database is created and codes are written. Some of the codes required amendments and improvement to it so these are being developed at this fourth stage of the waterfall model. With the development completed, testing will begin. The codes and database are tested to ensure the results obtained are as intended. More time is spent on both development and testing stages because it is inevitable to have errors and issues and buffer time is allocated for troubleshooting. This system will use both waterfall and spiral models.

**3. PROPOSED METHODS AND PROCEDURES**

The proposed system is a web based application and maintains a centralized repository of all related information. The system allows one to easily access the relevant information and make necessary travel arrangements. Users can decide about places they want to visit and make bookings online for travel and accommodation

**3.1 Summary of improvements**

**3.1.1 Functional improvements**

* It is very much faster than manual system.
* Easy and fastest record finding technique.
* It is very much flexible to work.
* Man power required is very less.
* Data can be stored for a longer period.

**3.1.2 Improvements to Existing Capabilities**

* There is no requirement of power.
* Easy to generate a report.
* It is very much flexible to work.
* Easy to search a destination.
* Data may not be perfect human error.

**3.2 Assumptions and Constraints**

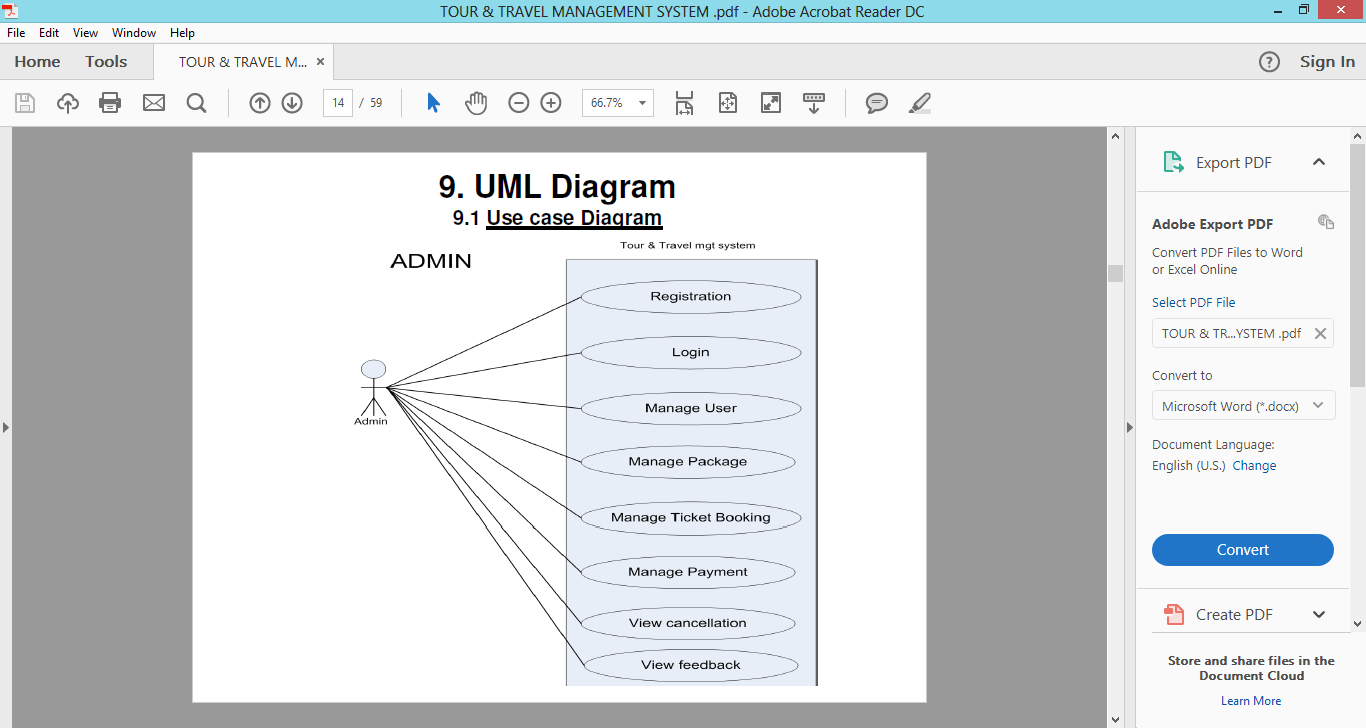
* This software must be run in the windows platform.
* This software will run on that pc which has PHP & MySQL programming and database languages.

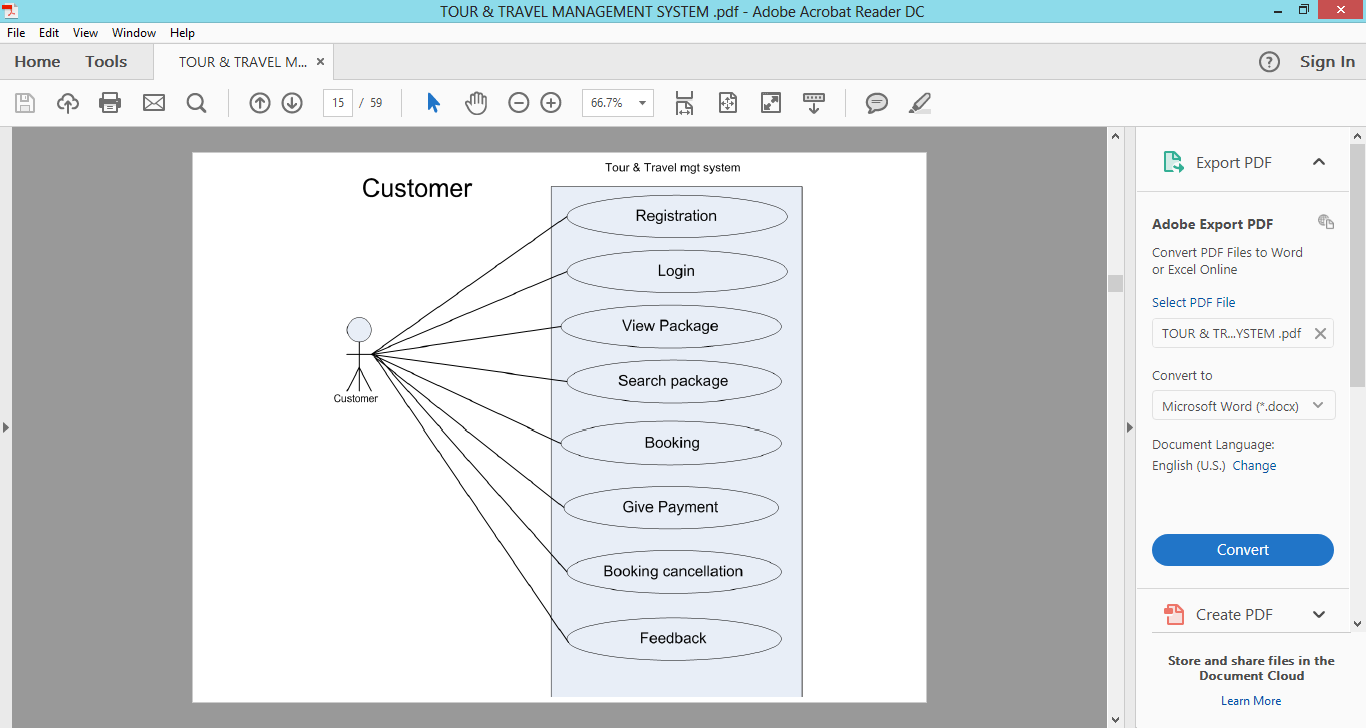
**4. DETAILED CHARACTERISTICS**

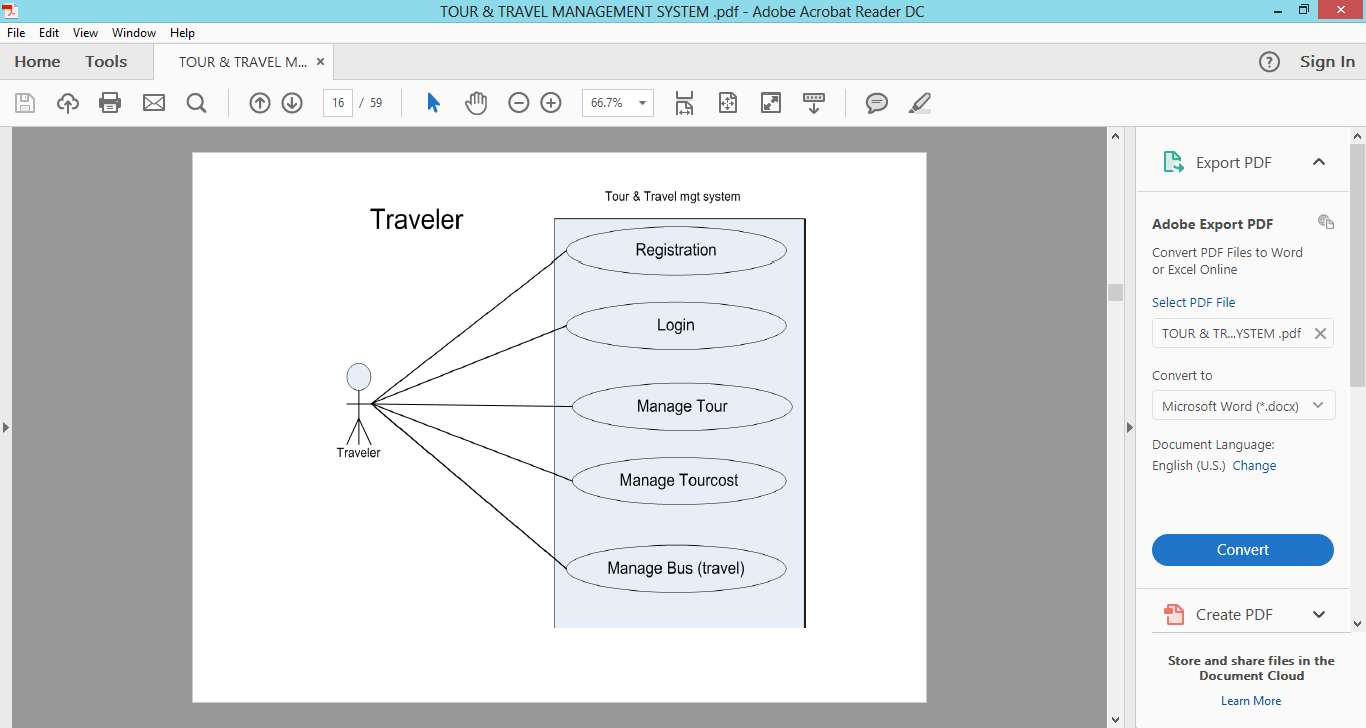
**4.1 Specific performance requirements**

* To computerize the manual work to increase the efficiency of the system.
* To reduce the paper work.
* It’s easy to maintain the report.
* One of the another powerful advantage of the computerized  software is that it will allow only authorized person to operate the system
* The system is portable so if you have windows PHP and MySQL installed in any pc then you can easily run the software.
* No need for any modification to run the software in another system.

**4.2 Functional Area System Functions**







**5. DESIGN CONSIDERATIONS**

The proposed system follows function oriented design approach. A function-oriented design strategy relies on decomposing the system into a set of interacting functions with a centralized system state shared by these functions. Functions may also maintain local state information but only for the duration of their execution. Detailed design follows a process which entails conceptual design, embodiment design and detail design and, when performed professionally, eventually results in a well designed solution.

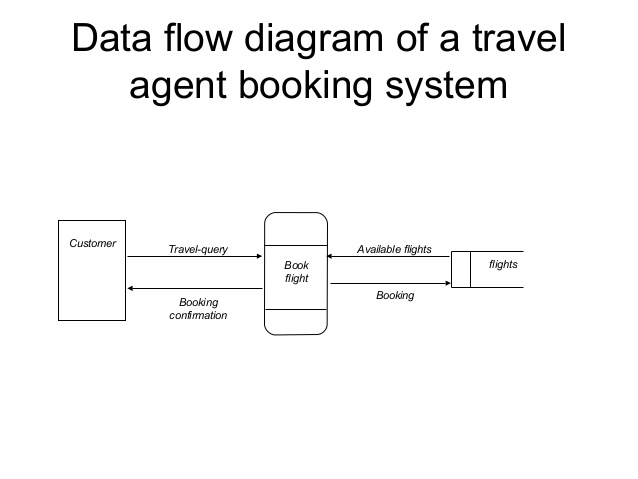
Conceptual design is Phase One of detailed design and engineering in which drawings are the main output. The drawings produced are often quite simple ideas with little detail, but the aim of the conceptual phase is to commit ideas to paper.

The Embodiment phase of the detailed design and engineering process starts with the concept and develops it into a workable system that can be further developed. During this phase, engineers will typically follow a framework of clarity, simplicity and safety in achieving the design goal.

Detailed design is the phase where the design is refined and plans, specifications and estimates are created. Detailed design will include outputs such as 2D and 3D models, P & IDs, cost build up estimates, procurement plans etc. This phase is where the full cost of the project is identified.

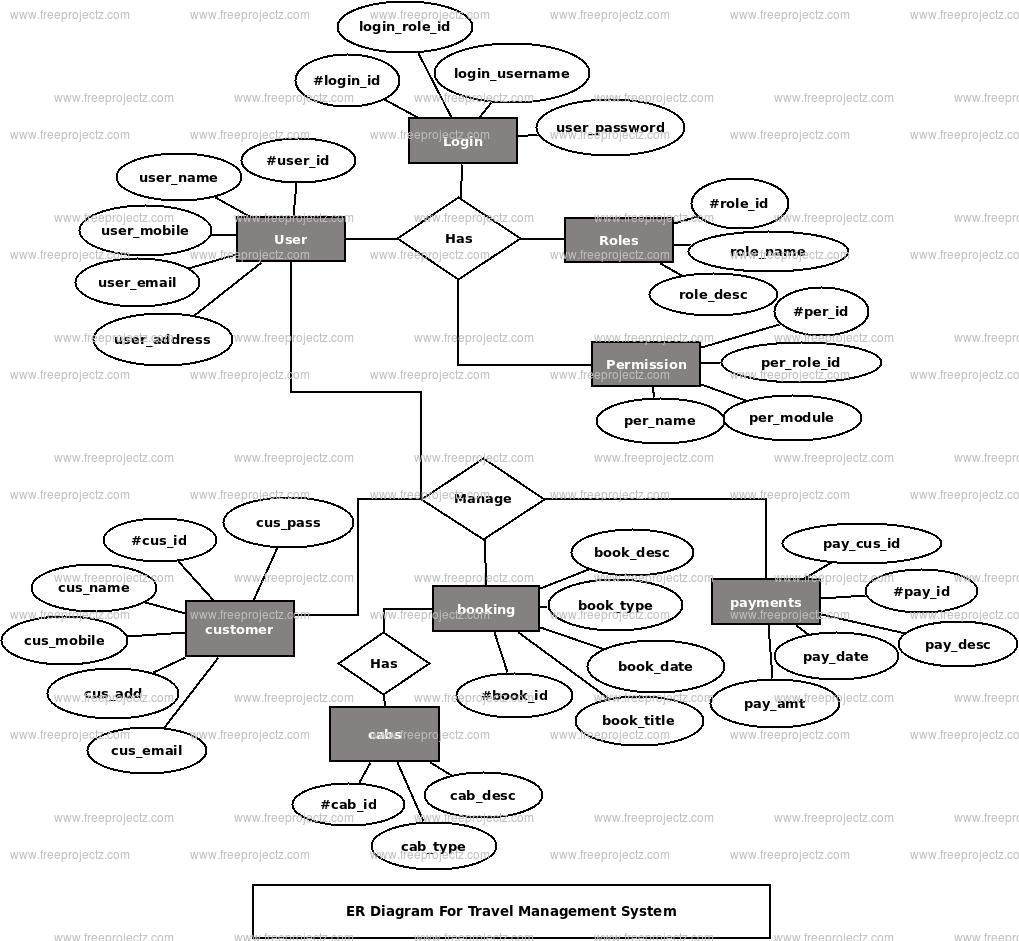
**5.1 System description**

System design is a highly creative process. It considers how the problem must be solved. The termdesign describes a final system and process by which it is developed. It includes the construction of program and program testing. A systematic method has to achieve the beneficial result. It involves starting with a vague idea and developing it into a series of steps. The series of steps for successful system development or successful system design are given below: - First step is to study the problem completely because first of all we should know the goal, which we have to achieve. Second we should see what kind of input we require and what kind of output we give so that we can get desired output from the system. It is very challenging step of the system development. According to input requirement of the system the structures of various databases should be designed. Next we should know that what kind of web pages should develop to reach the final goal.

**5.1.1 Flowchart**

.

**5.1.2Entity Relationship Diagram**



**DESIGN DOCUMENTATION**

**1. Overview**

**1.1 Scope**

This is a standard for software design descriptions (SDD). An SDD is a representation of a software design to be used for recording design information and communicating that design information to key design stakeholders. This standard is intended for use in design situations in which an explicit software design description is to be prepared. These situations include traditional software construction activities, when design leads to code, and “reverse engineering” situations where a design description is to be recovered from an existing implementation. The standard can be applied to commercial, scientific, or military software that runs on digital computers.

Applicability is not restricted by the size, complexity, or criticality of the software. This standard can be applied to the description of high-level and detailed designs. This standard is not limited to use with specific methodologies for design, configuration management, or quality assurance. This standard does not require the use of any particular design languages, but establishes requirements on the selection of design languages for use in an SDD. The standard can be applied to the preparation of SDDs captured as paper documents, automated databases, software development tools or other media.

**1.2 Purpose**

This standard specifies requirements on the information content and metadata organization of SDDs. This standard specifies requirements for the selection of design languages to be used for software design description, and requirements for documenting design viewpoints to be used in organizing a software design description.

**1.3 Intended Audience**

This standard is intended for technical and managerial stakeholders who prepare and use SDDs. It will guide a designer in the selection, organization, and presentation of design information. For an organization developing its own design description practices, the use of this standard will help to ensure that design descriptions are complete, concise, consistent, interchangeable, appropriate for recording design experiences and lessons learned, well organized and easy to communicate.

**1.4 Conformance**

A Software Design Description conforms to this standard if it satisfies all of the requirements of this standard.

**2. References**

Tutorialspoint.com

IEEE Template for System Requirement Specification Documents:

<https://goo.gl/nsUFwy>

**3. Definitions**

For the purposes of this standard, the following terms and definitions apply. The IEEE Standard Dictionary of Electrical and Electronics Terms [IEEE Std 100], and Industry Implementation of International Standard ISO/IEC 12207:1995 Software life cycle processes [IEEE/EIA Std 12207.0–1996] should bereferenced for terms not defined in this clause.

**design concern:** an area of interest with respect to a software design.

**design constraint**: an element of a design view which names and specifies a rule or restriction on a design entity, design attribute or design relationship.

**design element**: an item occurring in a design view which may be any of the following: design entity, design relationship, design attribute, or design constraint.

**design attribute:** an element of a design view which names a characteristic or property of a design entity, design relationship or design constraint

**design entity**: an element of a design view which is structurally, functionally or otherwise distinct from other elements, or plays a different role relative to other design entities.

**design overlay**: a representation of additional, detailed or derived design information organized with reference to a previously-defined design view.

**design rationale**: information capturing the reasoning of the designer which led to the system as designed, including design options, tradeoffs considered, decisions made, and the justifications of those decisions.

**design relationship**: an element of a design view which names a connection or correspondence between design entities.

**design stakeholder:** an individual, organization or group (or classes thereof playing the same role) having an interest in, or design concerns relative to, the design of some software item.

**design subject**: any software item or system which is to be constructed or which already exists and is to be analyzed, for which a software design description will be prepared.

**Alternate terms to consider**: softwareunder design or system under design.

**designer**: the stakeholder responsible for devising and documenting the software design.

**design view:** a representation comprised of one or more design elements to address a set of design concerns from a specified design viewpoint.

**design viewpoint:** a specification of the elements and conventions available for constructing and using a design view.

**diagram (type)**: a logically coherent fragment of a design view, using selected graphical icons and conventions for visual representation from an associated design language, to be used for representing selected design elements of interest for a system under design from a single viewpoint.

**4. Conceptual Framework for Software Design Descriptions**

This clause establishes a conceptual framework for Software Design Descriptions. The conceptual framework includes basic terms and concepts of software design description, the context in which SDDs are prepared and used, the stakeholders who use them, and how they are used.

**4.1 Software Design in Context**

A design is a framework which demonstrates a means to fulfill the requirements for some software item and to guide the implementation of that software item. A design subject is any software item to be constructed or which already exists and is to be analyzed, without loss of generality we will also refer to a design subject as the system under design or software under design. This standard does not establish what a design subject may be. Examples of design subjects include systems, subsystems, applications, components, libraries, application frameworks, application program interfaces (APIs) and design pattern catalogs.

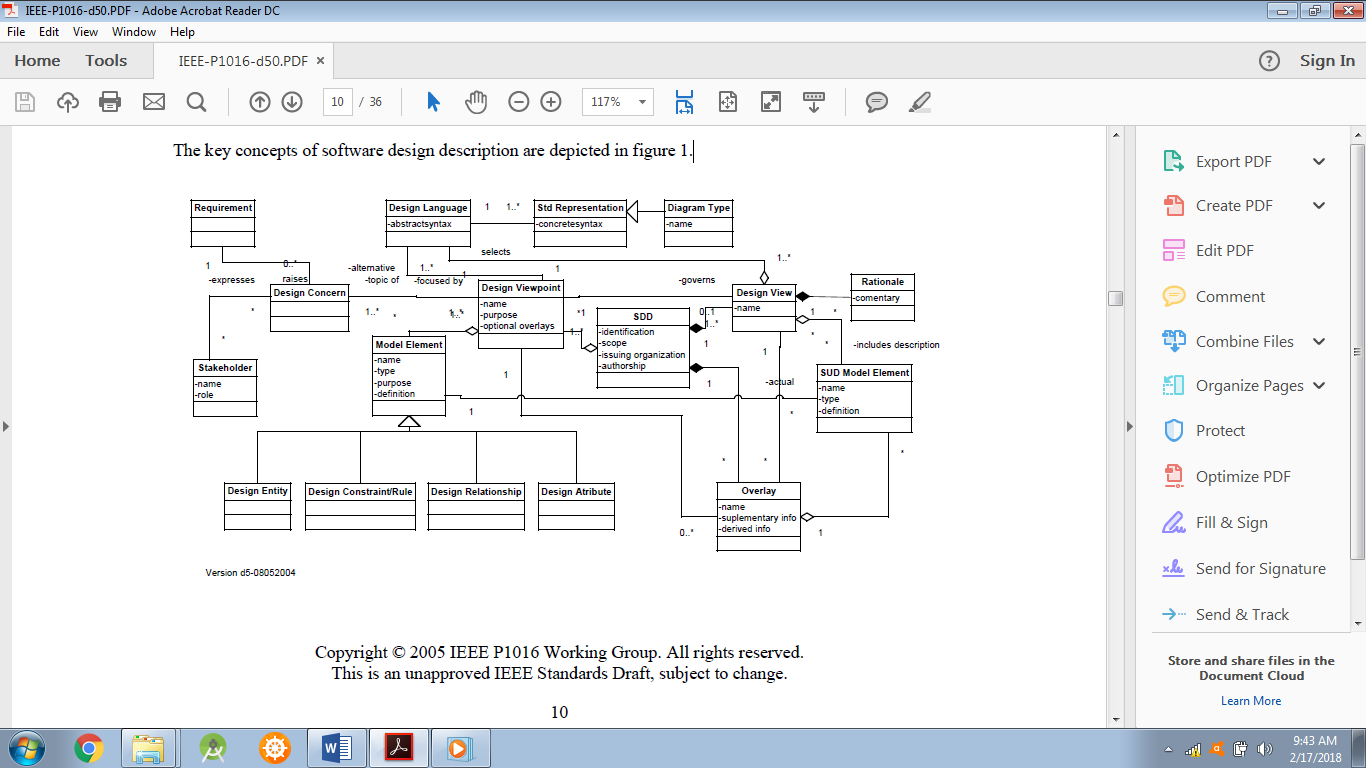
A software design description (SDD) is a representation of some design subject of interest. An SDD is prepared to represent exactly one design subject. An SDD can be produced to capture one or more levels or layers of concern with respect to its design subject. These levels or layers are usually defined by the design methods in use or the life cycle context; they have names such as architectural design, logical design, or physical design. An SDD can be prepared and used in a variety of design situations. Typically, an SDD is prepared to support the development of a software item to solve a problem, where this problem has been expressed in terms of a set of requirements. The contents of the SDD can then be traced to these requirements. In other cases, an SDD can be prepared to understand an existing system lacking any design documentation. Typically, there is a system under development or under review for which an SDD is to be

described such that information of interest is to be captured, organized, presented and disseminated to all interested parties. This information of interest can be used for planning, analysis, implementation and evolution of the software system, by identifying and addressing essential design concerns. A designconcern is any area of interest in the design, pertaining to its development, implementation, or operation. Design concerns are expressed by design stakeholders—those parties which may be individuals, groups and organizations with an interest in the design of the system. Frequently design concerns arise from specific requirements on the software, others arise from contextual constraints. Typical design concerns include functionality, reliability, performance, and maintainability. Typical design stakeholders include users, developers, software designers, system integrators, maintainers, acquirers, and project managers.

An SDD is organized using design views. A design view addresses one or more of the design concerns. Each design view is governed by a design viewpoint. Each design viewpoint focuses on a set of the design concerns and introduces a set of descriptive resources (or view elements) that are used to construct and interpret the design view. E.g., a viewpoint may introduce familiar elements such as functions, input and outputs; these elements are used to construct a functional view.

There are four kinds of view elements: design entities, design relationships among entities, design attributes and design constraints on those elements. A design viewpoint will define the view element typesto be used in any design views it governs. Each design view used to represent a software system isexpressed as a collection of instances of design entities, attributes, the relationships among design entitiesand constraints on those elements. The design information needs of stakeholders of the system under design are to be satisfied through use of these elements. It is sometimes useful to gather and present information which does not strictly following the partition ofinformation by viewpoints. A design overlay is a mechanism intended to organize and present designadditional, detailed or derived information with respect to an already-defined design viewpoint for thispurpose.

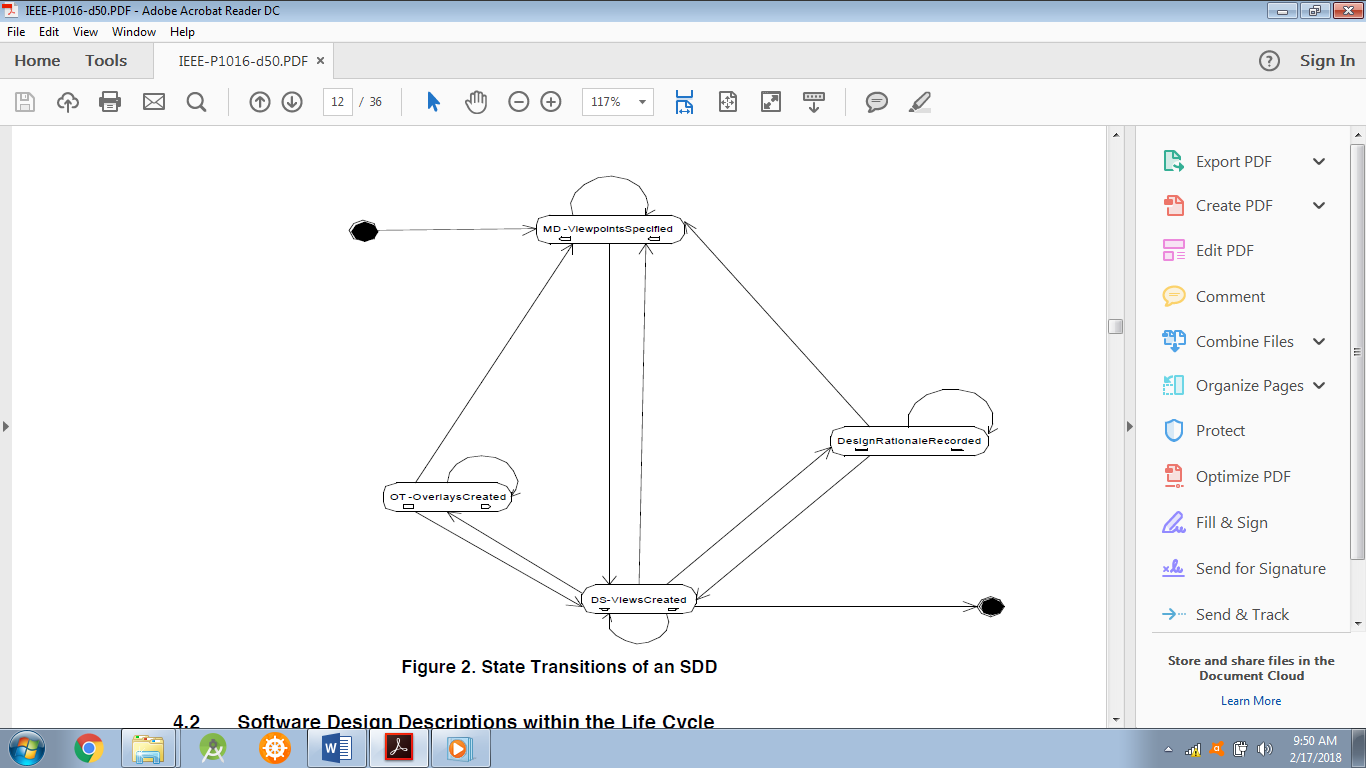
It is not sufficient to document only the actual design; it is also useful to capture the design rationale including alternative designs and the justifications for choices which have been made, whenever experiencesuggests long-term relevance and value of such information for current and future stakeholders.

The key concepts of software design description are depicted in figure 1.

**Figure 1. Conceptual Framework of Software Design Description**

To facilitate automation, exchange and long-term relevance of SDDs, the design state and design rationale information is accompanied by metadata describing both design state and design rationale. Metadata are organized around viewpoints and design state around design views to include instances for design entities, attributes and relationships. Figure 2 depicts the state transitions of an SDD in this respect.

Most importantly, this standard assumes use of models in software design. Models and their representations can be used in different modes: as sketches or rough drafts; as blueprints suitable for implementation; and as executable specifications. The use of models as sketches, while highly recommended in practice, is not governed by this standard; the intended modes are blueprints and executable specifications, as formal engineering documents. The primary use of a sketch is as an aid for thinking, and in conversation about ideas before those ideas can be systematized into designs as either blueprints or executable specifications. Blueprints are developed under general requirements of consistency and reasonable completeness and intended to communicate designs to humans such as to implementers or to maintainers trying to understand the design in order to change it. Executable specifications further restrict descriptions to those that can be automatically translated into implementations on real machines, without the intervention of human intelligence. Software design descriptions covered by this standard are not only formalized using defined languages, but are also intended to be precise i.e., rigorous irrespective of the medium to be used to record them. It is the intent to communicate specific ideas only and not to present complete designs to be implemented as is, that distinguishes sketches from blueprints and executable specifications.

There is no restriction by this standard to the use of any design language in sketches or to the use of sketches in documentation, but the expectation is to use the P1016 standard to govern blueprints and/or executable specifications in the full scope of design responsibility.

**Figure 2. State Transitions of an SDD**

**4.2 Software Design Descriptions within the Life Cycle**

In this standard, a typical cycle will be used to describe the various design situations in which an SDD can be created and used. This life cycle is based on IEEE/EIA 12207.

**4.2.1 Influences on SDD Preparation**

The key software life cycle product that drives a software design is typically the software requirements description (SRD). An SRD captures the software requirements which will drive the design, and may contain design constraints that must be considered or observed.

**4.2.2 SDD Influences on Software Life Cycle Products**

The SDD influences the content of several major software life cycle products. Developers of these products will be recognized among the SDD’s intended audience.

− Software Requirements Description. Design decisions, or design constraints discovered during the preparation of the SDD, may lead to requirements changes. Often traceability between requirements and design is maintained to manage these changes.

− Test Documentation. Test planning can be influenced by the SDD, but any white-box testing activities at the level of unit, integration, and system testing, are directly influenced by the SDD. Developers of any test specifications and test cases that relate to this type of testing should cover the design functionality, relationships, objects, and data descriptions contained in the SDD.

**PROTOTYPING MODEL:**

The Prototyping Model is a systems development method (SDM) in which a [prototype](http://searchcio-midmarket.techtarget.com/definition/prototype)(an early approximation of a final system or product) is built, tested, and then reworked as necessary until an acceptable prototype is finally achieved from which the complete system or product can now be developed. This model works best in scenarios where not all of the project requirements are known in detail ahead of time. It is an iterative, trial-and-error process that takes place between the developers and the users.

There are several steps in the Prototyping Model:

1. The new system requirements are defined in as much detail as possible. This usually involves interviewing a number of users representing all the departments or aspects of the existing system.
2. A preliminary design is created for the new system.
3. A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.
4. The users thoroughly evaluate the first prototype, noting its strengths and weaknesses, what needs to be added, and what should to be removed. The developer collects and analyzes the remarks from the users.
5. The first prototype is modified, based on the comments supplied by the users, and a second prototype of the new system is constructed.
6. The second prototype is evaluated in the same manner as was the first prototype.
7. The preceding steps are iterated as many times as necessary, until the users are satisfied that the prototype represents the final product desired.
8. The final system is constructed, based on the final prototype.
9. The final system is thoroughly evaluated and tested. Routine maintenance is carried out on a continuing basis to prevent large-scale failures and to minimize downtime.

**4.2.3 Design Verification and Design Role in Validation**

Verification is a process for determining whether the software products of an activity fulfill the

requirements or conditions imposed on them in the previous activities. [IEEE/EIA 12207.0] An SDD can be subject to design verification to ascertain whether the design: is consistent with stated requirements; implements intended design decisions (such as those pertaining to interfaces, inputs, outputs, algorithms, resource allocation, and error handling); achieves intended qualities (such as safety, security, maintainability); and conforms to an imposed architecture. Verification therefore raises a set of design concerns which can be dealt with in the SDD and subjected to inspection or analysis.

Validation is a process for determining whether the requirements and the final, as-built system or software product fulfills its specific intended use. [IEEE/EIA 12207.0] The SDD can play a role in this process mainly by providing: an overview necessary for understanding the implementation; the rationale justifying design decisions made; and traceability back to the requirements on the software item.

**5. Design description information content**

**5.1 Introduction**

The required elements of an SDD are:

− an identification of the SDD,

− its identified stakeholders,

− its identified design concerns,

− its selected design viewpoints, each with type definitions of its allowed design elements and design languages,

− its design views,

− its design overlays, and

− its design rationale.

These are described in the remainder of this clause.

**5.2 SDD Identification**

An SDD includes the following descriptive information:

− date of issue and status;

− scope;

− issuing organization;

− authorship (responsibility or copyright information);

− references;

− context;

− one or more design languages for each design viewpoint used;

− body;

− summary;

− glossary;

− change history.

**5.3 Design Stakeholders and Concerns**

An SDD identifies the stakeholders for the design subject.

An SDD identifies the design concerns for each stakeholder.

An SDD addresses each identified design concern. In addition, an SDD shall address the following design concerns when applicable to the system under design:

− purpose: describe the design of a software item. (the software design description and the architecture description provides the detailed design needed to implement the software.) may be supplemented by software item interface design and database design.

− description of how the software item satisfies the software requirements, including algorithms and data structures;

− software item input/output description;

− static relationships of software units;

− concept of execution, including data flow and control flow;

− requirements traceability: 1) software component-level requirements traceability; 2) software unit-level requirements traceability;

− rationale for software item design;

− reuse element identification.

**5.4 Design views**

A software design description is organized into one or more design views. A design view is a

representation consisting of design entities, design entity attributes, design relationships and design constraints to address an identified set of design concerns from a specific viewpoint.

The purpose of a design view is to address design concerns pertaining to the design subject, to allow a design stakeholder to focus on design details from a different perspective or design viewpoint, and effectively address relevant requirements.

Design views are the means of organizing an SDD to satisfy the needs of each design stakeholder and to promote separation of concerns. Each design view addresses one or more design concerns. Together, these views provide a comprehensive description of the design in a concise and usable form that simplifies information access and assimilation. Each software design stakeholder can have a distinct perspective on what are the essential aspects of a software design. Other design information may be extraneous to that stakeholder.

An SDD is complete when each identified design concern is the topic of at least one design view, all design attributes refined from each design concern by some viewpoint have been specified for all of the design entities and relationships in its associated view and all design constraints have been applied.

An SDD is consistent if there are no known conflicts between the elements of its design views.

**5.5 Design viewpoints**

A design viewpoint is a specification of the conventions for constructing and using a design view. It identifies the resources from which to develop individual design views. For each design view in an SDD, there is a design viewpoint governing it.

Each design viewpoint is specified by:

− the viewpoint name;

− the concerns which are the topics of the viewpoint;

− the resources, or view elements, provided by that viewpoint, specifically the types of design entities, attributes, relationships and constraints introduced by that viewpoint or used by that viewpoint (which may have been defined elsewhere). These elements may be realized by one or more design languages;

− analytical methods or other operations to be used in constructing the view based upon the viewpoint, and criteria for evaluating the design based upon the viewpoint; and

− the viewpoint source (e.g., authorship or citation) when applicable.

In addition, a design viewpoint specification provides the following information on using the

viewpoint:

− formal or informal consistency and completeness tests to be applied to the models making up an associated view;

− evaluation or analysis techniques to be applied to the models; and

− heuristics, patterns, or other guidelines to assist in construction or synthesis of an associated view.

An SDD includes a rationale for the selection of each selected viewpoint.

Each design concern identified in an SDD is addressed by at least one viewpoint selected for use. A design concern may be the focus of more than one viewpoint in an SDD.

**5.6 Design elements**

A design element (or model element) is any item occurring in a design view. A design element may be any of the following: design entity, design relationship, design attribute, or design constraint.

Each design element have the following attributes: a name, a type and an expression.

The type of each design element is introduced within exactly one design viewpoint definition. A design element is referenced in one or more design views.

**5.6.1 Design entities**

Design entities capture key elements of a software design. Each design entity bears a unique name and may be referenced by that name throughout the SDD. The intent of design entities is to divide the design subject into separate elements that can be considered, implemented, changed, and verified with minimal effect on other entities.

Entities can represent systems, subsystems, libraries, frameworks, abstract collaboration patterns, generic templates, components, classes, data stores, modules, programs, and processes.

The number and type of entities needed to express a design view are dependent on a number of factors, such as the complexity of the system, the design technique used, and the programming environment.

Although entities are different in nature, they possess common characteristics. Each design entity have a name, type, and purpose. The common characteristics of entities are described by design entity attributes. There are common relationships among entities such as interfaces or shared data.

**5.6.2 Design attributes**

A design attribute is a named characteristic or property of a design entity, design constraint, or a design relationship. It provides a statement of fact about the design element. Design attributes can be thought of as questions about design elements. The answers to those questions are the values of the attributes. All the questions can be answered, but the content of the answer will depend upon the nature of the entity. The collection of answers provides a complete description of an entity.

All attributes declared by a design viewpoint is specified. Attribute descriptions should include references and design considerations such as tradeoffs and assumptions when appropriate. In some cases, attribute descriptions may have the value none.

− The design attributes defined in 5.7.1 through 5.7.3 is applied to all design entities used in an SDD.

**5.6.2.1 Unique naming attribute**

The name of the element. All design elements have a name. Each element shall have an unambiguous reference name. The names for the elements may be selected to characterize their nature. This simplifies referencing and tracking in addition to providing identification.

**5.6.2.2 Entity type attribute**

A description of the kind of element. The type attribute describes the nature of the element. It may simply name the kind of element, such as subsystem, component, framework, library, class, subprogram, module, function, procedure, process, object, persistent object, class, or data store. Alternatively, design elements may be grouped into major classes to assist in locating an element dealing with a particular type of information. For a given design description, the chosen element types is applied consistently.

**5.6.2.3 Purpose attribute**

A description of why the element exists. The purpose attribute provides the rationale for the creation of the element.

**5.6.2.4 Author attribute**

Identification of designer. The author attribute identifies the author of the element, as an individual, or the organization responsible for design description.

**5.6.3 Design relationships**

A design relationship is a named association or correspondence among two or more design entities. It provides a statement of fact about those design entities.

**5.6.4 Design constraints**

A design constraint is an element of a design view which names a rule or restriction imposed on another design element which may be a design entity, design attribute or design relationship.

**5.7 Design overlays**

A design overlay is a mechanism for presenting additional, detailed or derived information with respect to an already-defined design view. It is frequently convenient to capture such information, as an alternative to introducing a new viewpoint, using overlays upon a subset of the information in the diagrams selected in existing relevant viewpoints.

Each design overlay shall be clearly marked. Each design overlay shall be clearly associated with a single viewpoint.

**5.8 Design rationale**

Design rationale is information capturing the reasoning of the designer which led to the system as designed, including design options, tradeoffs considered, decisions made, and the justifications of those decisions.

Design rationale takes the form of commentary, made throughout the decision process and associated with collections of design elements. It captures the reasoning that led to the system as it has been designed.

Design rationale includes: design issues raised and addressed in response to design concerns; design options considered; tradeoffs evaluated; decisions made; criteria used to guide design decisions; and arguments and justifications made to reach decisions.

**5.9 Design languages**

A design language is a notation, representational scheme or other modeling technique used to develop, analyze, and document a software design. There are many design languages used to describe software designs. Design languages are selected as a part of design viewpoint declaration (5.5).

A design language may be selected for a design viewpoint only if it supports all modeling elements defined by that viewpoint.

For use in SDDs, design languages shall be selected which have:

− a well-defined syntax and semantics; and

− the status of an available standard or equivalent defining document.

In a conforming SDD, only standardized and established\* (defined and convenient) design languages shall be used. In the case of a newly-invented design language, the language definition must be provided as a part of the viewpoint declaration.

NOTE—Standardized design languages in common use are preferable to established one without a formal definition.

Examples of standardized languages include: UML

Examples of established languages include: State Diagram.

**6. Design Viewpoints**

**6.1 Introduction**

This clause defines several design viewpoints for use in SDDs. It illustrates the specification of viewpoints in terms of design language selections, relates design concerns with viewpoints and establishes language- (notation-, method-, and process-) neutral names for selected viewpoints. For each viewpoint, its name, design concerns, and appropriate design languages, are listed.

Short descriptions relating a minimal set of design entities, design relationships, design entity attributes, and design constraints are provided for each viewpoint. Additional references pertinent to the use of each viewpoint are also listed.

These viewpoints are required with a caveat, a qualified designer judgment is necessary to tailor out viewpoints not of interest in a particular situation, or to refine viewpoints

**Definition:** A viewpoint is a collection of patterns, templates, and conventions for constructing one type of view. It defines the stakeholders whose concerns are reflected in the viewpoint and the guidelines, principles, and template models for constructing its views.

**6.2 Context Viewpoint**

The Context Viewpoint is used to depict the services provided by a design subject with reference to an explicit context. That context is defined by reference to actors which include users and other stakeholders which interact with the design subject in its environment. The Context Viewpoint provides a “black box” perspective on the design subject.

Services depict an inherently functional aspect or anticipated cases of use of the design subject (hence “use cases” in UML). Stratification of services and their descriptions in the form of scenarios of actors’ interactions with the system provide a mechanism for adding detail. Services may also be associated with actors through information flows. The content and manner of information exchange with the environment implies additional design information and the need for additional viewpoints (e.g., Interaction Viewpoint).

A Deployment Overlay to a Context view can be transformed into a Deployment view whenever the execution hardware platform is part of the design subject; for stand-alone software design, a Deployment Overlay maps software entities onto externally available entities not subject of the current design effort. Similarly, work allocation to teams and other management perspectives are overlays in the design.

**6.2.1 Design Concerns**

The purpose of the Context Viewpoint is to identify a design subject’s offered services, its actors (users and other interacting stakeholders), to establish the system boundary, to effectively delineate the design subject’s scope of use, operation.

Drawing a boundary separating a design subject—whether system, subsystem, or component—from its environment, determining a set of services to be provided, and the information flows between design subject and its environment, is typically a key design decision; making this viewpoint applicable to most design efforts.

As the system is portrayed as a black box, with internal decisions hidden, the Context view is often a starting point of design, showing what is to be designed functionally as the only available information about the design subject: a name and an associated set of externally identifiable services. Requirements analysis may identify these services with a specification of Quality of Service attributes, henceforth invoking many non-functional requirements. Frequently incomplete a context view is begun in requirements analysis and the work persists in completing this view during the design process.

**6.2.2 View Elements**

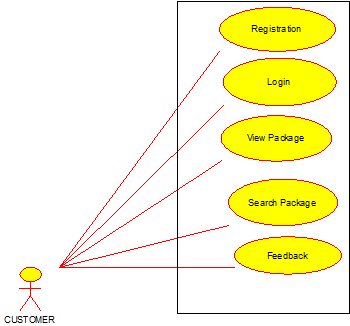
**Entities**: Actors: external active elements interacting with the design subject, including users, other stakeholders and external systems or other items. Services: also called use cases. Directed information Flowsbetween the design subject, treated as a black box, and its actors associate actors with services. Flows capture the expected information content exchanged.

**Relationships**: receive generated output and provide received input (between actors and the design subject). All design entities of this viewpoint are recursively decomposable into like entities to support hierarchical description. Therefore composition and generalization relationships are needed.

**Constraints**: Qualities of service, formats and media of interaction (provided to and received from) with environment as required by the environment are design constraints for this viewpoint.

**USE CASE DIAGRAMS:**

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different [use cases](https://en.wikipedia.org/wiki/Use_case) in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.



**6.3 Structure Viewpoint**

The Structure Viewpoint is used to document the internal structure of coarse-grained components and classes in terms of like elements (recursively).

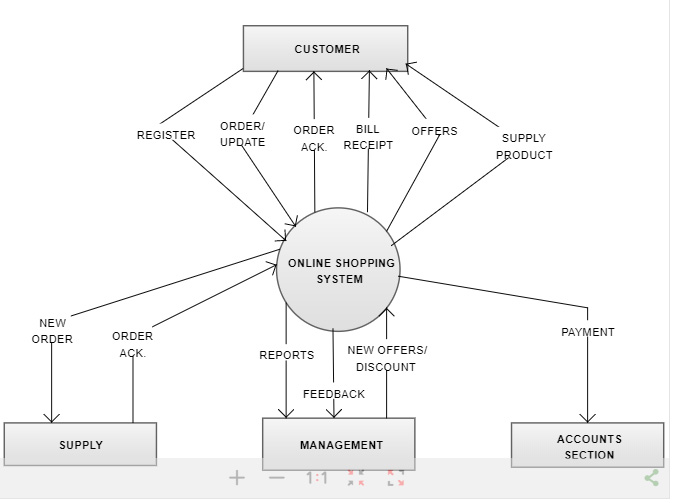
**6.3.1 Design Concerns**

Internal structure in terms of interaction between the various modules upto a detailed level with the help of DFDs.

**6.3.2 View elements**

Entities: actors.

Relationships:association.



**6.3.3 Example languages**

UML Internal (composite) Structure Diagram, UML Class Diagram and UML Package Diagram.

**6.4 State Dynamics Viewpoint**

Reactive systems and systems whose objects may have interested states require this dynamic viewpoint.

**6.4.1 Design concerns**

Modes, states, transitions, and constrains in time ordered events- systems dynamic

**6.4.2 View elements**

**Entities**: event, condition, state, transition, activity, composite state, submachine state, region, trigger.

**Relationships**: part-of, internal, effect, entry, exit, attached-to. Attributes: name, completion, active, initial, final.

**Constraints**: guard conditions, concurrency, synchronization, state invariant, transitionconstraint, protocol.

**6.5 Algorithm Viewpoint**

The detailed design description of operations (methods, functions), the internal details, logic, of each design entity; this applies to components, classes, and individual methods as design entities.

**6.51 Design concerns**

This description contains the details needed by programmers, analysts of algorithms re time-space performance and specifically coders prior to implementation. The detailed design description can also be used to aid in producing unit test plans.

**6.5.2 View elements**

These details include the attribute descriptions for identification, processing, and data. This attribute information should be provided for all design entities.

**ALGORITHM:**

Begin;

I New User creates user name and password to login.

II Admin can change and add new packages available for travelling.

III Customer can search for required packages.

IV Details of all the packages available for travelling.

End.