COLLECTION OF FORMAL DOCUMENTATION ON THE VARIOUS STAGES OF SOFTWARE DEVELOPMENT LIFE CYCLE

GROUP MEMBERS

SHAURYA RAJ VERMA	: 510519105
SHASHANK SRIVASTAV	: 5101519106
RISHITA DE	: 510519107
SAYAK RANA	: 510519108
ABHIROOP MUKHERJEE	: 510519109
NIKHIL JOSHI	: 51019110

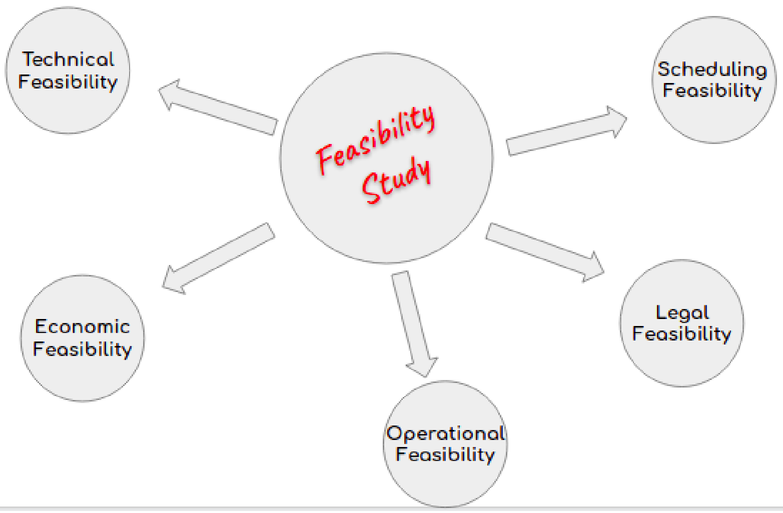
Feasibility Study

Feasibility Study is the first step of Software development life cycle. It is a management oriented activity which finds out if a project can be undertaken and assesses its operational , technical and economic merit.

There are five sorts of feasibility studies—individual topics that a feasibility study investigates :

1. Technical feasibility
2. Economic feasibility
3. Operational feasibility
4. Legal feasibility feasibility
5. Scheduling feasibility

Feasibility Study in addition to exploring all above areas generally provide some recommendation whether a project should be undertaken or not? It also gives opinions about other alternatives available. Management or client of the product , then on the basis of Feasibility Study, decide on the fate of the project.



All examples are given from the project OES web application feasibility study.

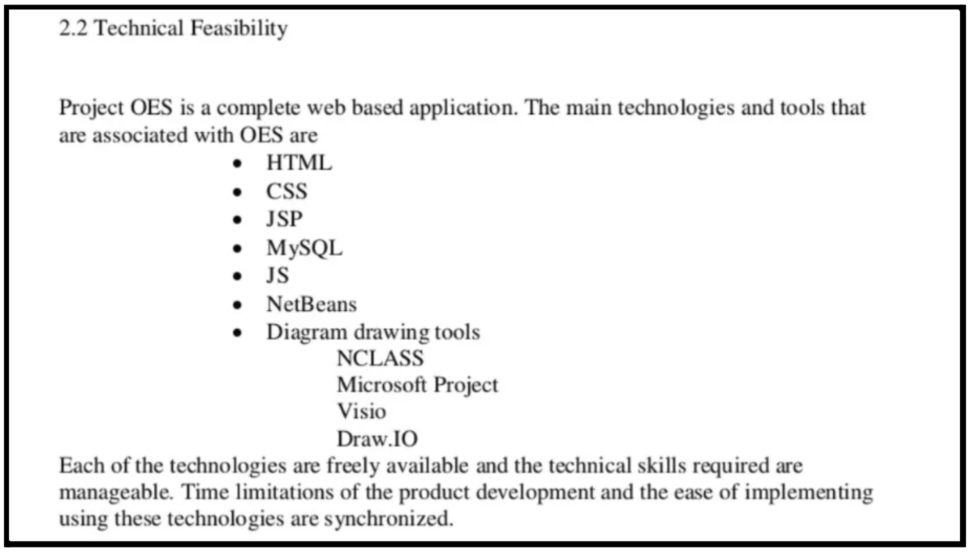
❖ Technical feasibility

A large part of determining resources has to do with assessing technical feasibility. The technical requirements of the proposed project are considered and then compared to the technical capability of the organization. The systems project is considered technically feasible if the internal technical capability is sufficient to support the project requirements.

It also determines if it is possible to upgrade the current system and what are other alternatives to the proposed project.

It answers the question like :

- Is it a practical proposition?
- Do we currently possess the necessary technology?
- Is the project feasible within the limits of current technology?
- Manpower- programmers, testers & debuggers requirement.

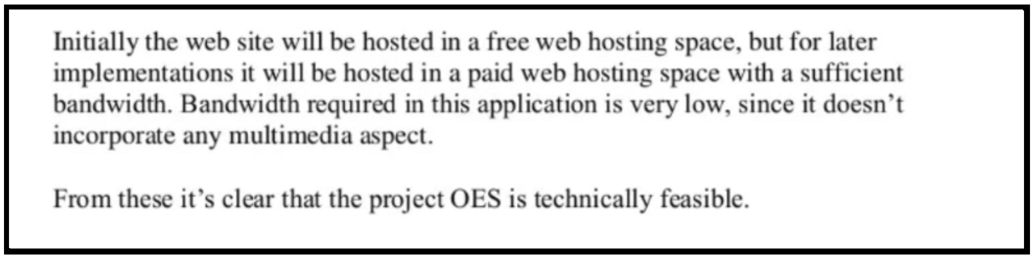


2.2 Technical Feasibility

Project OES is a complete web based application. The main technologies and tools that are associated with OES are

- HTML
- CSS
- JSP
- MySQL
- JS
- NetBeans
- Diagram drawing tools
  - NCLASS
  - Microsoft Project
  - Visio
  - Draw.IO

Each of the technologies are freely available and the technical skills required are manageable. Time limitations of the product development and the ease of implementing using these technologies are synchronized.



Initially the web site will be hosted in a free web hosting space, but for later implementations it will be hosted in a paid web hosting space with a sufficient bandwidth. Bandwidth required in this application is very low, since it doesn't incorporate any multimedia aspect.

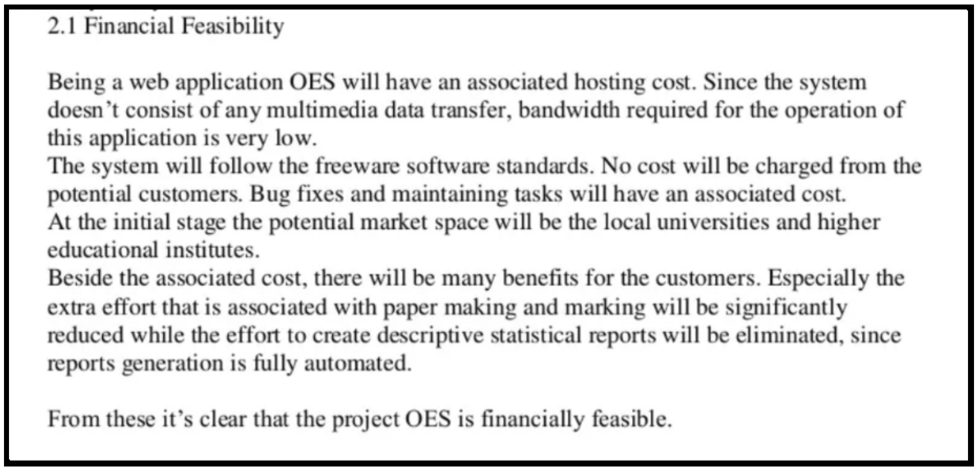
From these it's clear that the project OES is technically feasible.

❖ Economic feasibility

Economic analysis could also be referred to as cost/benefit analysis. It is the most frequently used method for evaluating the effectiveness of a new system. In economic analysis the procedure is to determine the benefits and savings that are expected from a candidate system and compare them with costs. If benefits outweigh costs, then the decision is made to design and implement the system.

Possible questions raised in economic analysis are:

- Estimated cost of hardware
- Estimated cost of software/software development·
- Is the system cost effective?
- Do benefits outweigh costs?
- Selection among alternative financing arrangements (rent/lease/purchase)


2.1 Financial Feasibility

Being a web application OES will have an associated hosting cost. Since the system doesn't consist of any multimedia data transfer, bandwidth required for the operation of this application is very low.
The system will follow the freeware software standards. No cost will be charged from the potential customers. Bug fixes and maintaining tasks will have an associated cost.
At the initial stage the potential market space will be the local universities and higher educational institutes.
Beside the associated cost, there will be many benefits for the customers. Especially the extra effort that is associated with paper making and marking will be significantly reduced while the effort to create descriptive statistical reports will be eliminated, since reports generation is fully automated.

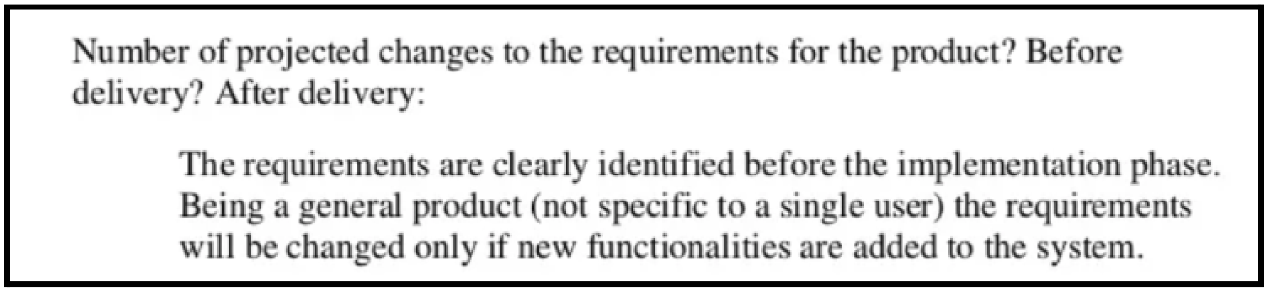
From these it's clear that the project OES is financially feasible.

- ❖ Operational feasibility

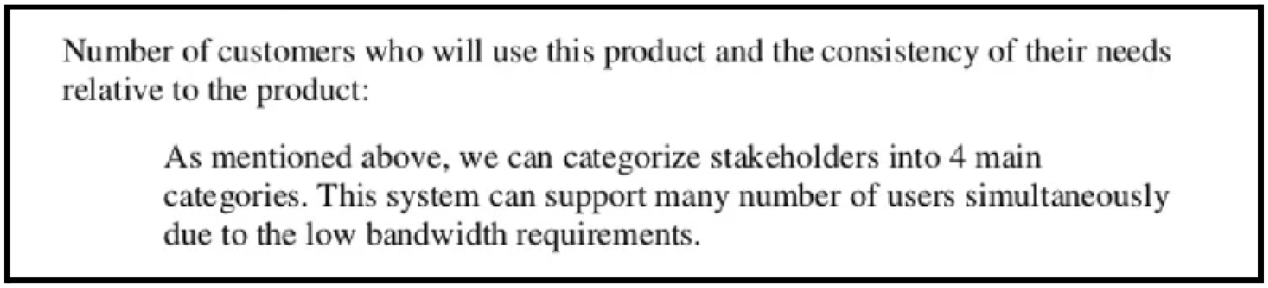
  Operational feasibility is a measure of how well a proposed system solves the problems, and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development. Operational feasibility reviews the willingness of the organization to support the proposed system.

This process answers the following questions:

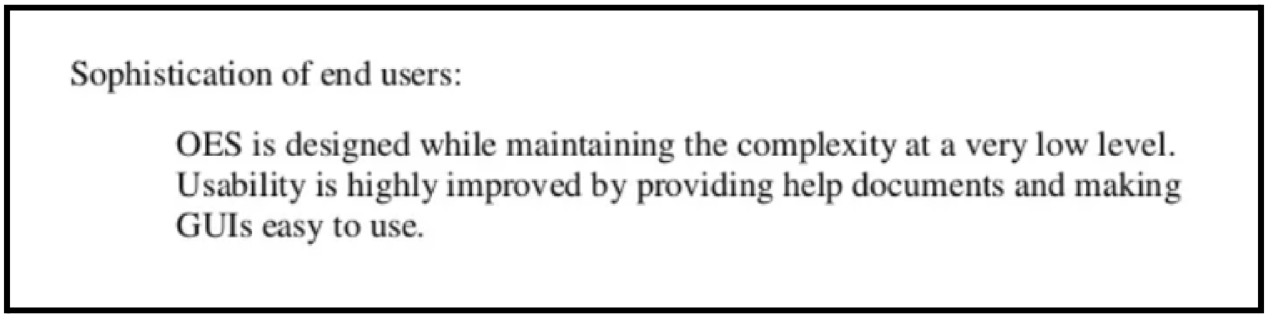
- Does current mode of operation provide adequate throughput and response time?
- If the new system is developed, will it be used?
- Will the proposed system really benefit the organization?
- How will the working environment of the end-user change?
- Can or will end-users and management adapt to the change?
- What end-users or managers may resist or not use the system?


Number of projected changes to the requirements for the product? Before delivery? After delivery:

The requirements are clearly identified before the implementation phase. Being a general product (not specific to a single user) the requirements will be changed only if new functionalities are added to the system.


Number of customers who will use this product and the consistency of their needs relative to the product:

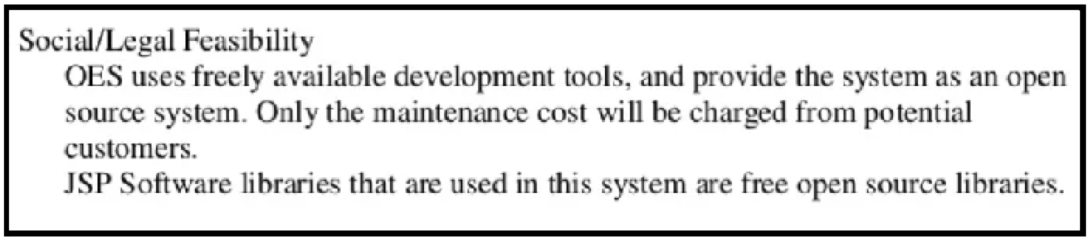
As mentioned above, we can categorize stakeholders into 4 main categories. This system can support many number of users simultaneously due to the low bandwidth requirements.


Sophistication of end users:

OES is designed while maintaining the complexity at a very low level. Usability is highly improved by providing help documents and making GUIs easy to use.

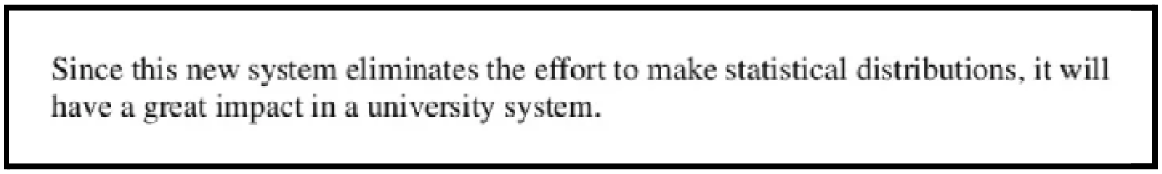
- Legal Feasibility

  This evaluation looks at if any component of the planned project violates any regulations, such as zoning regulations, data protection legislation, or social media legislation.

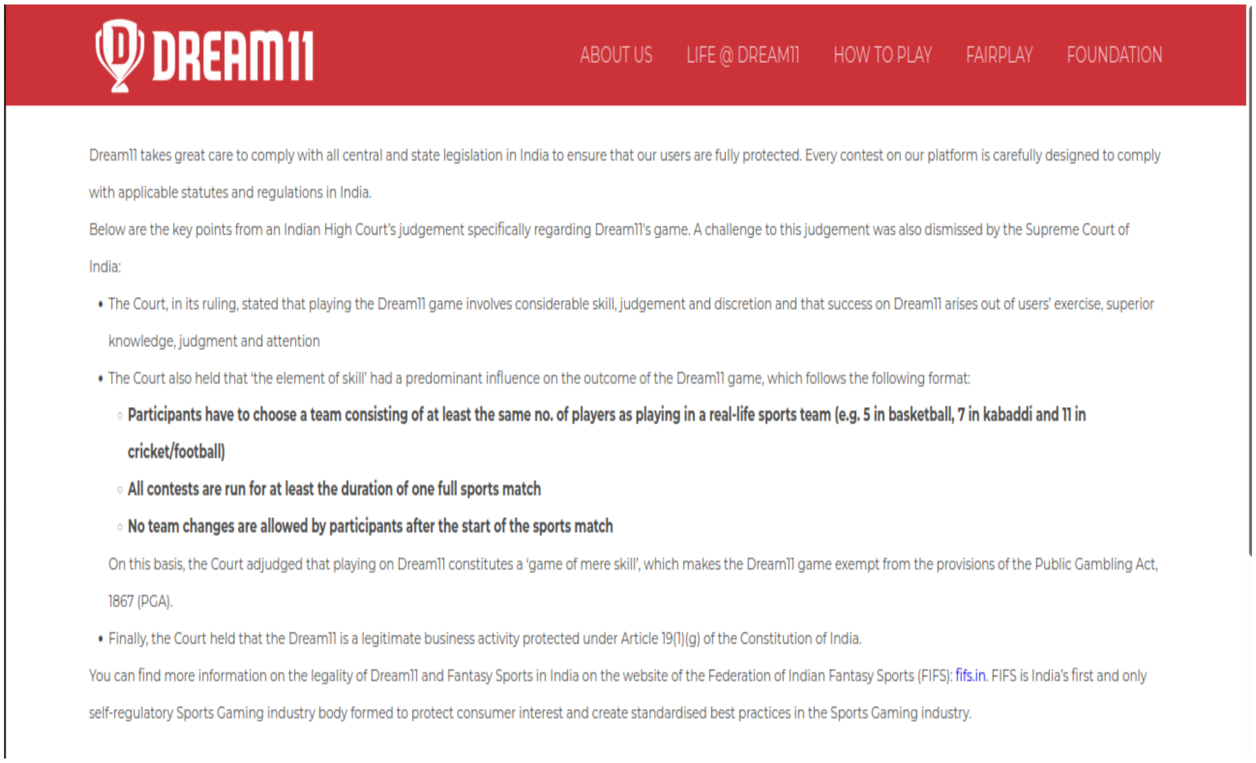

Social/Legal Feasibility

OES uses freely available development tools, and provide the system as an open source system. Only the maintenance cost will be charged from potential customers.

JSP Software libraries that are used in this system are free open source libraries.


Since this new system eliminates the effort to make statistical distributions, it will have a great impact in a university system.

Another example of Legal Feasibility : Court ruling in case of Dream11 allows more firms to develop and promote their fantasy gaming app.



Dream11 takes great care to comply with all central and state legislation in India to ensure that our users are fully protected. Every contest on our platform is carefully designed to comply with applicable statutes and regulations in India.

Below are the key points from an Indian High Court's judgement specifically regarding Dream11's game. A challenge to this judgement was also dismissed by the Supreme Court of India:

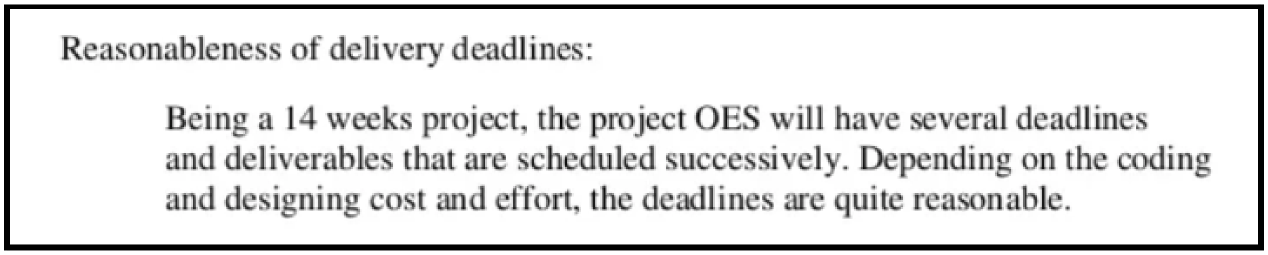
- The Court, in its ruling, stated that playing the Dream11 game involves considerable skill, judgement and discretion and that success on Dream11 arises out of users' exercise, superior knowledge, judgment and attention
- The Court also held that 'the element of skill' had a predominant influence on the outcome of the Dream11 game, which follows the following format:
  - **Participants have to choose a team consisting of at least the same no. of players as playing in a real-life sports team (e.g. 5 in basketball, 7 in kabaddi and 11 in cricket/football)**
  - **All contests are run for at least the duration of one full sports match**
  - **No team changes are allowed by participants after the start of the sports match**

  On this basis, the Court adjudged that playing on Dream11 constitutes a 'game of mere skill', which makes the Dream11 game exempt from the provisions of the Public Gambling Act, 1867 (PGA).
- Finally, the Court held that the Dream11 is a legitimate business activity protected under Article 19(1)(g) of the Constitution of India.

You can find more information on the legality of Dream11 and Fantasy Sports in India on the website of the Federation of Indian Fantasy Sports (FIFS): fifs.in. FIFS is India's first and only self-regulatory Sports Gaming industry body formed to protect consumer interest and create standardised best practices in the Sports Gaming industry.

❖ Scheduling Feasibility

An organization predicts the length of time it will take to finish a project in scheduling feasibility. Generally , projects are also divided into different phases and deadlines of each phase are provided separately so that resources can be mapped to them efficiently.



Reasonableness of delivery deadlines:

Being a 14 weeks project, the project OES will have several deadlines and deliverables that are scheduled successively. Depending on the coding and designing cost and effort, the deadlines are quite reasonable.

Requirement Analysis

Requirements analysis or requirements engineering is a process used to determine the needs and expectations of a new product.

The requirement analysis process involves the following steps

1. **Problem recognition and identification of users requirements:**
   The main aim of requirement analysis is to fully understand the main objective of requirement that includes why it is needed, does it meet the minimum requirements , add value to the product, will it be beneficial, does it increase quality of the project, does it will have any other effect.

2. **Interpret and analyse tools required:**
   Requirements analysis can be performed successfully by using the right set of tools.So choosing the right set of tools is a necessity.Tools include :
   - ➔ Github
   - ➔ VS Code,Sublime Text,IntelliJ IDEA.
   - ➔ Node JS,Angular,jQuery.
   - ➔ OpenCV, Keras, Tensorflow and SciPy.

3. **Evaluation and Synthesis:**
   Evaluation means judgement about something whether it is worth or not and synthesis means to create or form something. Here are some tasks are that are important in the evaluation and synthesis of software requirement :

- ➔ To define all functions of software that are necessary.
- ➔ To define all data objects that are present externally and are easily observable.
- ➔ To evaluate the flow of data is worth or not.
- ➔ To fully understand the overall behavior of a system that means overall working of the system.
- ➔ To identify and discover constraints that are designed.
- ➔ To define and establish character of the system interface to fully understand how the system interacts with two or more components or with one another.

4. **User-friendly and attractive:**
   The provided platform of the software should be user-friendly and highly interactive. A portal should be created where one can put his/her honest feedback for further improvements of the software.

Various models, charts and techniques are used to perform the analysis. Some are described below.

❖ Business Process Model and Notation (BPMN)

Business Process Model and Notation is used to create graphs that simplify the understanding of the business process. It is a popular technique used by business analysts to coordinate the sequence of messages between different participants in a related set of activities.

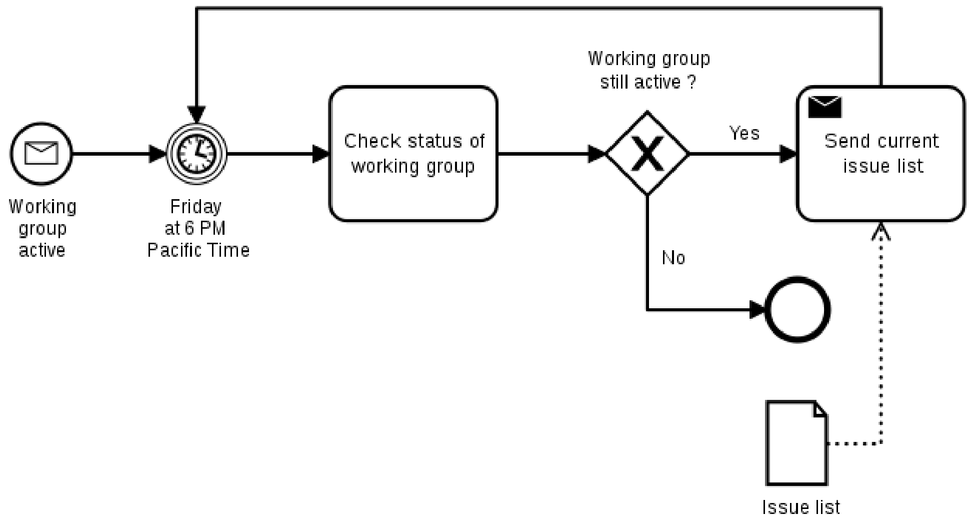

Working group active
Friday at 6 PM Pacific Time
Check status of working group
Working group still active ?
Yes
No
Send current issue list
Issue list

FIG : BPMN example

❖ Flowcharts

Flowcharts depict sequential flow and control logic of a related set of activities. They are useful for both technical and non-technical members.

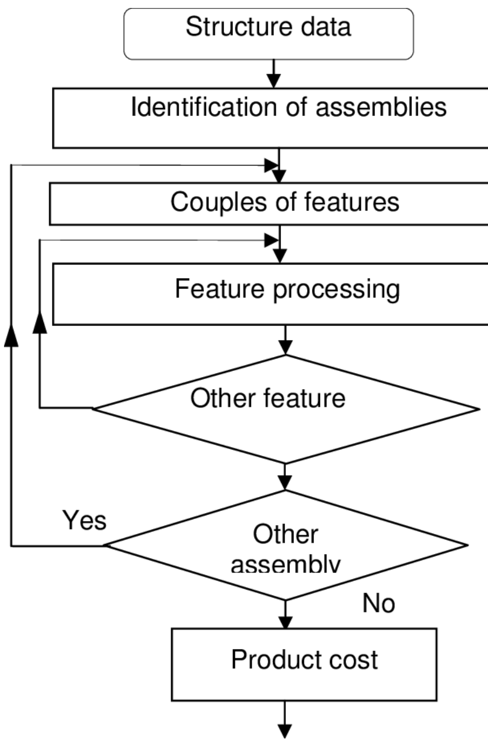


Fig: Flowchart example

❖ Gantt Charts

Gantt Charts provide a visual representation of tasks along with their scheduled timelines. They help business analysts visualize the start and end dates of all the tasks in a project.

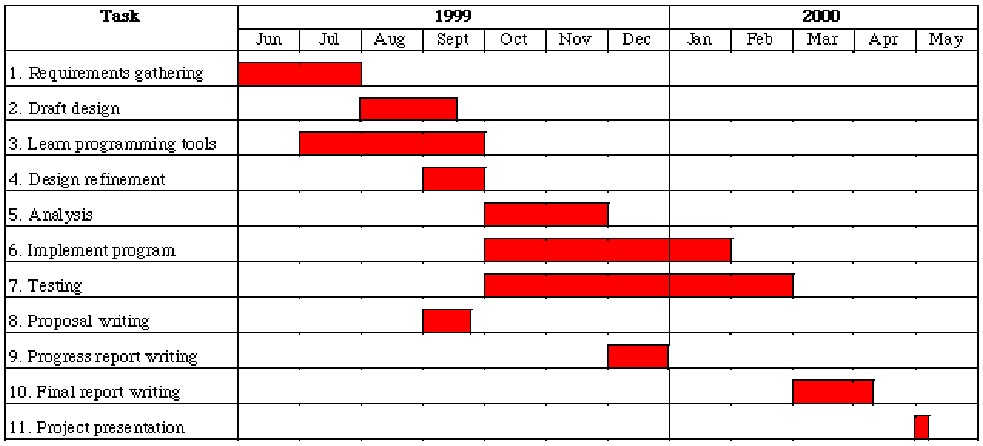
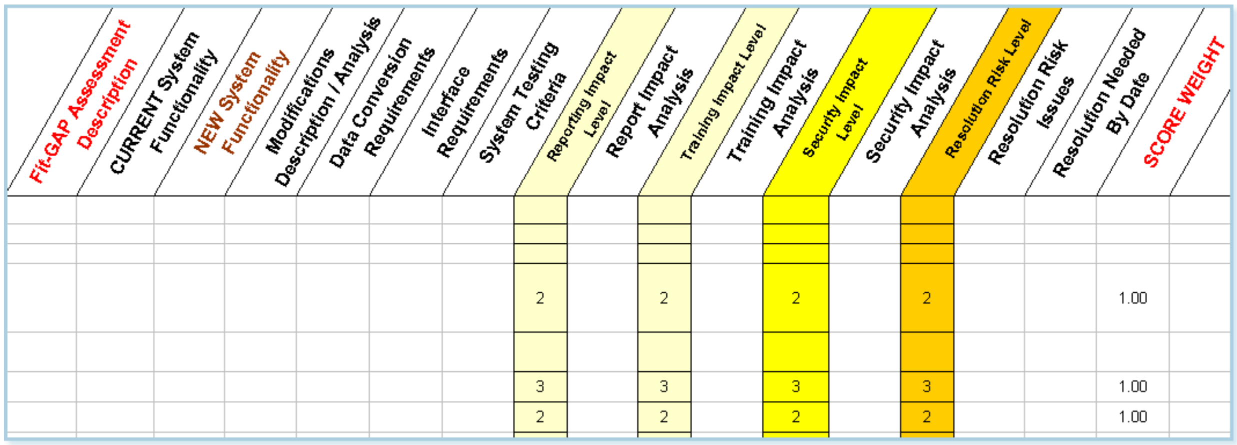


Fig: Gantt charts example

❖ Gap Analysis

Gap analysis evaluates the gaps in a product's performance to determine whether the requirements are met or not. They help business analysts determine the present state and target state of a product.



Fit-GAP Assessment Description	CURRENT System Functionality	NEW System Functionality	Modifications Description / Analysis	Data Conversion Requirements	Interface Requirements	System Testing Criteria	Reporting Impact Level	Report Impact Analysis	Training Impact Level	Training Impact Analysis	Security Impact Level	Security Impact Analysis	Resolution Risk Level	Resolution Risk Issues	Resolution Needed By Date	SCORE WEIGHT	
							2		2		2		2			1.00	
							3		3		3		3			1.00	
							2		2		2		2			1.00	

Fig: Gap analysis example

Design

“The purpose of the Software Design Document is to provide a description of the design of a system fully enough to allow for software development to proceed with an understanding of what is to be built and how it is expected to be built.” - [1]

[1] is a documentation of a design process for building a utility that interacts with and/or produces legal documents. It boasts interaction ability with a wide range of users. This specific version, dated 2007-04-20, is an update on the features to be offered by the utility and contains modifications that are built upon the previous version, dated 2007-01-31, of the design document

Below is a summary of the entire document which is readily available for viewing through [1].

1. **Introduction**
   This contains a short paragraph on - purpose and scope of the project

2. **Glossary**
   This contains short descriptions on various terms used in the document.

3. **Use Cases**
   - A use case diagram is a visual representation of the functionalities that are to be offered by the system and are in direct relation to the requirement analysis done prior to design.[2]
   - Initially the actors and their acceptable actions are listed out.
   - Then the use cases for each actor are specified.
   - A use case diagram is pictorially represented, one for the current modifications to the software as well as one for future modifications.

3.3.1 Document Manager- Essential Use Cases (“Enter Document into Workflow” for future update)

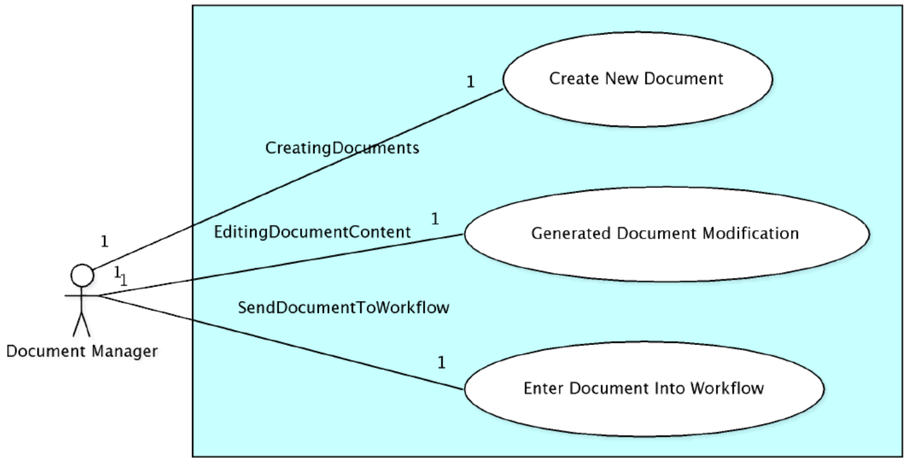


FIG : Shows the essential use cases of *Document Manager* user [1]

3.3.3 Administrative User – Use Cases (Future)

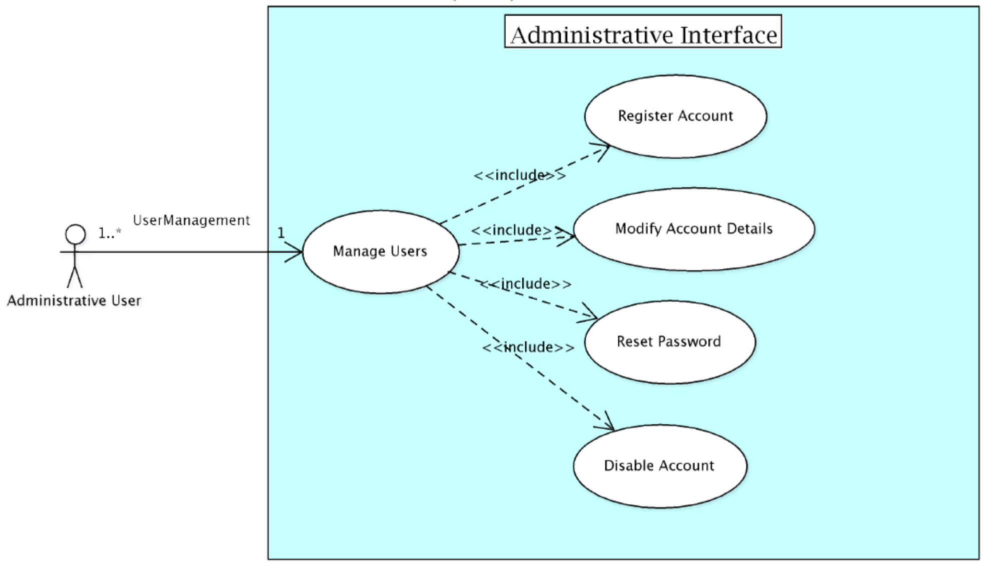
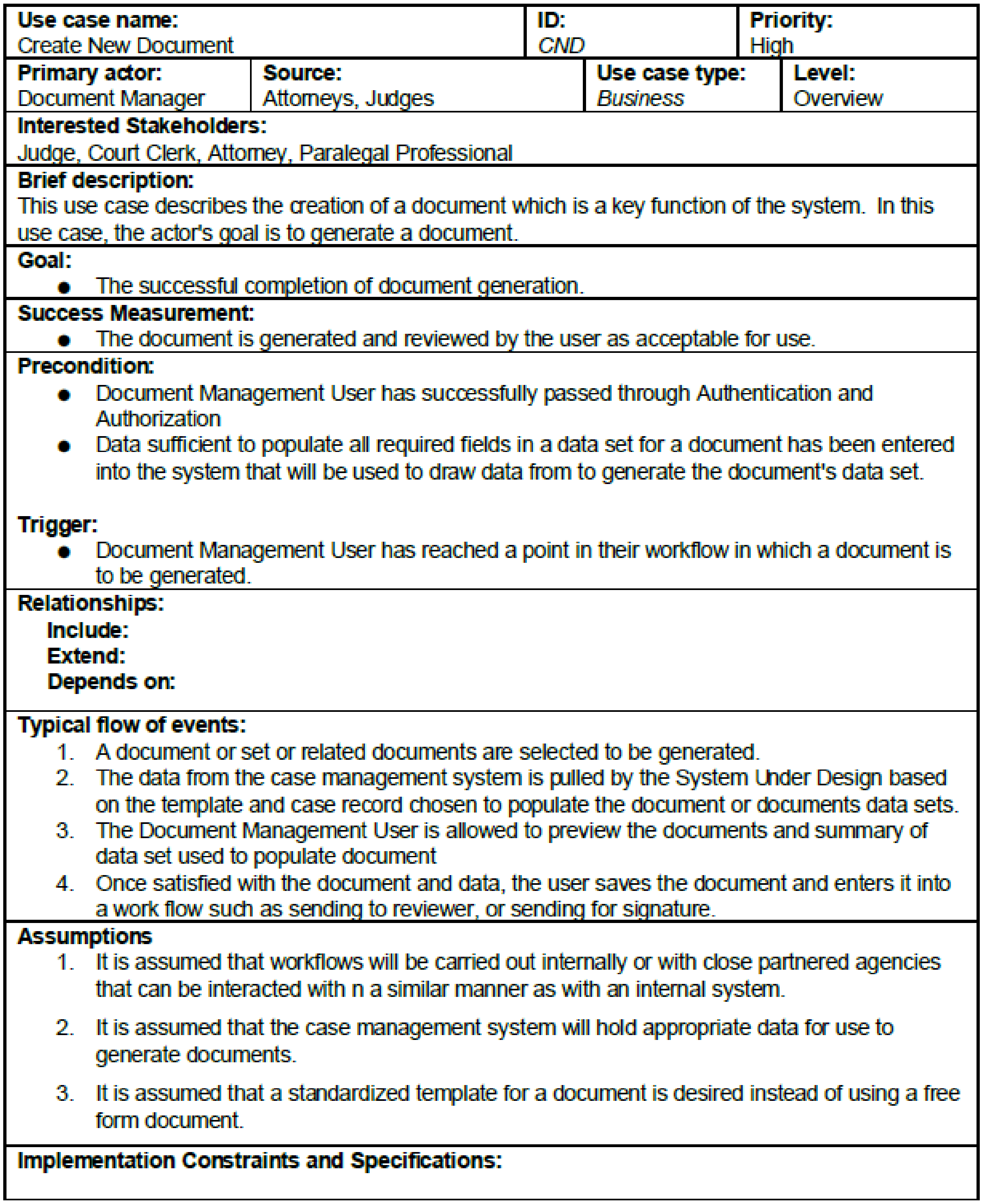


FIG : Shows the use cases that may arise when an *administrative* user uses the document utility [1]

- Furthermore a more detailed written description of the use cases is presented as below



Use case name: Create New Document		**ID:** *CND*	**Priority:** High
Primary actor: Document Manager	**Source:** Attorneys, Judges	**Use case type:** *Business*	**Level:** Overview
Interested Stakeholders: Judge, Court Clerk, Attorney, Paralegal Professional			
Brief description: This use case describes the creation of a document which is a key function of the system. In this use case, the actor's goal is to generate a document.			
Goal: ● The successful completion of document generation.			
Success Measurement: ● The document is generated and reviewed by the user as acceptable for use.			
Precondition: ● Document Management User has successfully passed through Authentication and Authorization ● Data sufficient to populate all required fields in a data set for a document has been entered into the system that will be used to draw data from to generate the document's data set. **Trigger:** ● Document Management User has reached a point in their workflow in which a document is to be generated.			
Relationships: **Include:** **Extend:** **Depends on:**			
Typical flow of events: 1. A document or set or related documents are selected to be generated. 2. The data from the case management system is pulled by the System Under Design based on the template and case record chosen to populate the document or documents data sets. 3. The Document Management User is allowed to preview the documents and summary of data set used to populate document 4. Once satisfied with the document and data, the user saves the document and enters it into a work flow such as sending to reviewer, or sending for signature.			
Assumptions 1. It is assumed that workflows will be carried out internally or with close partnered agencies that can be interacted with n a similar manner as with an internal system. 2. It is assumed that the case management system will hold appropriate data for use to generate documents. 3. It is assumed that a standardized template for a document is desired instead of using a free form document.			
Implementation Constraints and Specifications:			

FIG : Detailed description of the use case of *Document Manager* [1]

4. Design Overview

- This section begins by presenting the system architecture to be taken into consideration in the coding phase. It lists interface requirements, constraints and assumptions.

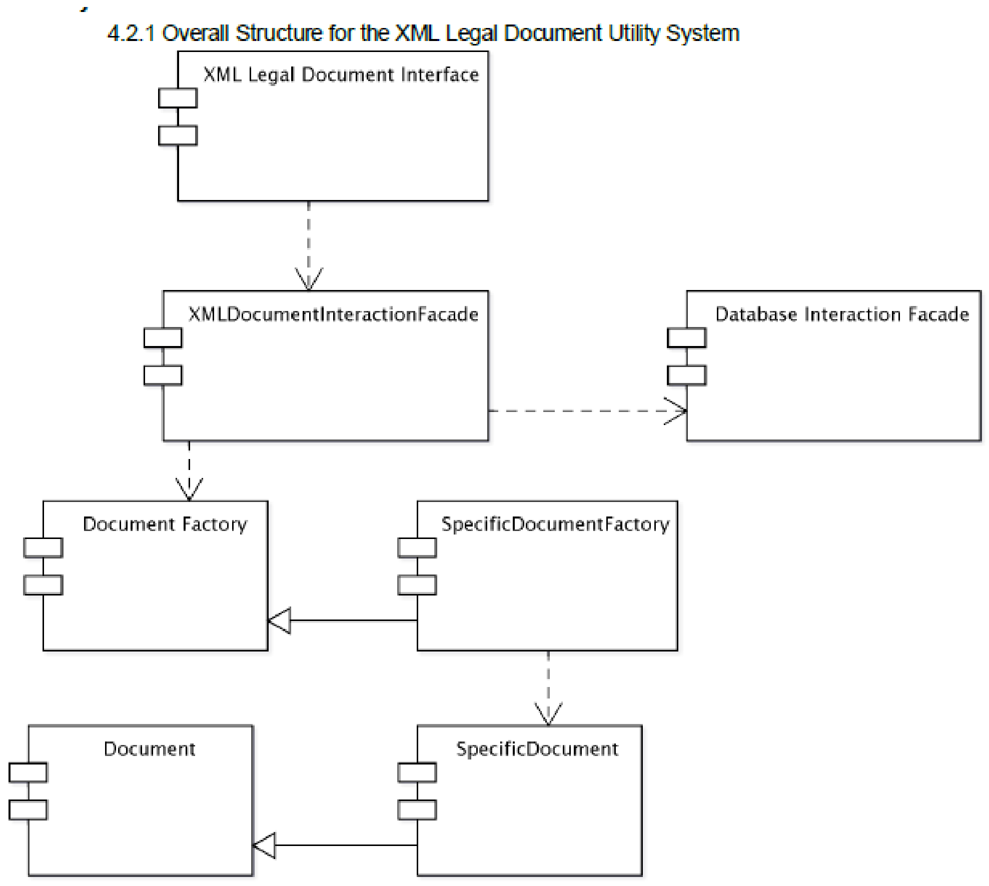

4.2.1 Overall Structure for the XML Legal Document Utility System
XML Legal Document Interface
XMLDocumentInteractionFacade
Database Interaction Facade
Document Factory
SpecificDocumentFactory
Document
SpecificDocument

FIG : Overall Structure for the XML Legal Document Utility [1]

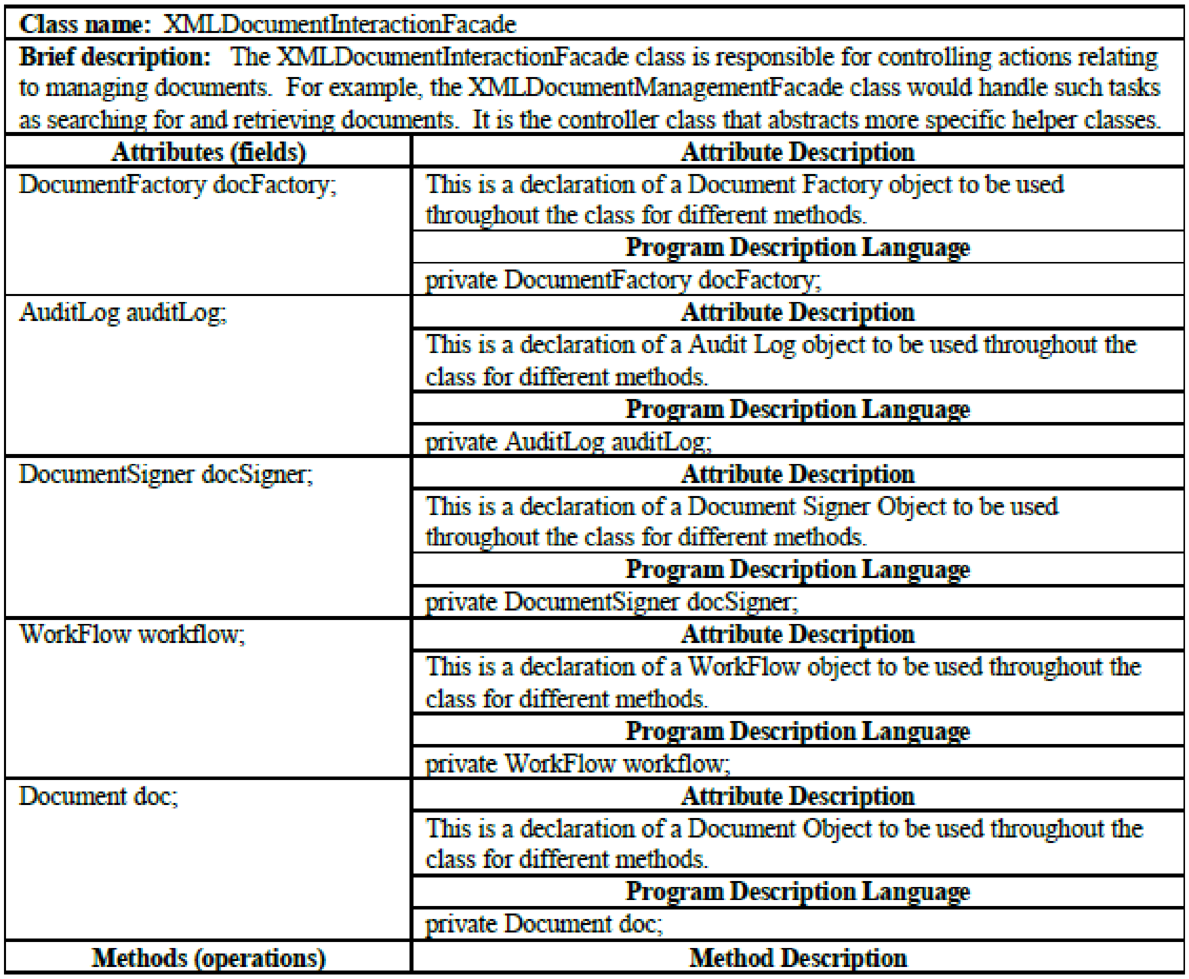
5. System Object Model

- This allows for a description of subsystems in use. However the modifications that this document documents are under one system and no sub systems are necessary.

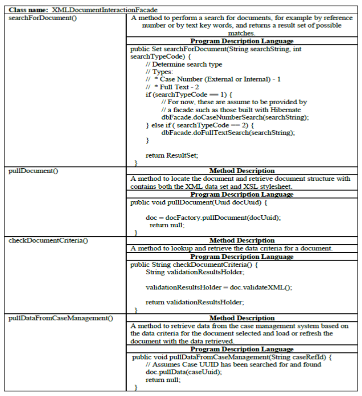
6. Object Descriptions

- This section quite simply contains a written version of the UML diagrams that may have been drawn for this project. It contains in detail all the classes and it's methods, lucidly written out for ready reference in the coding phase
-

6.1.1 XMLDocumentInteractionFacade

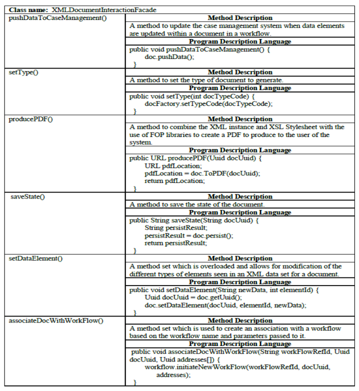


Class name: XMLDocumentInteractionFacade	
Brief description: The XMLDocumentInteractionFacade class is responsible for controlling actions relating to managing documents. For example, the XMLDocumentManagementFacade class would handle such tasks as searching for and retrieving documents. It is the controller class that abstracts more specific helper classes.	
Attributes (fields)	**Attribute Description**
DocumentFactory docFactory;	This is a declaration of a Document Factory object to be used throughout the class for different methods.
	Program Description Language
	private DocumentFactory docFactory;
AuditLog auditLog;	**Attribute Description**
	This is a declaration of a Audit Log object to be used throughout the class for different methods.
	Program Description Language
	private AuditLog auditLog;
DocumentSigner docSigner;	**Attribute Description**
	This is a declaration of a Document Signer Object to be used throughout the class for different methods.
	Program Description Language
	private DocumentSigner docSigner;
WorkFlow workflow;	**Attribute Description**
	This is a declaration of a WorkFlow object to be used throughout the class for different methods.
	Program Description Language
	private WorkFlow workflow;
Document doc;	**Attribute Description**
	This is a declaration of a Document Object to be used throughout the class for different methods.
	Program Description Language
	private Document doc;
Methods (operations)	**Method Description**



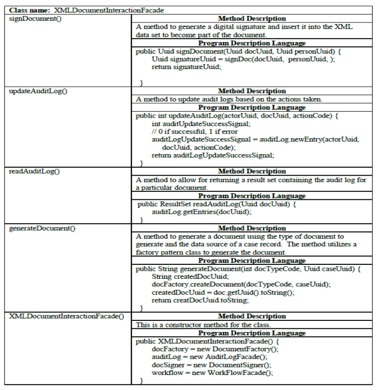
Class name: XMLDocumentInteractionFacade

	Method Description	Program Description Language
searchForDocument()	A method to perform a search for documents, for example by reference number or by text key words, and returns a result set of possible matches.	`public Set searchForDocument(String searchString, int searchTypeCode) {` `// Determine search type` `// Types:` `//  * Case Number (External or Internal) - 1` `//  * Full Text - 2` `if (searchTypeCode == 1) {` `// For now, these are assume to be provided by` `// a facade such as those built with Hibernate` `dbFacade.doCaseNumberSearch(searchString);` `} else if ( searchTypeCode == 2) {` `dbFacade.doFullTextSearch(searchString);` `}` `return ResultSet;` `}`
pullDocument()	A method to locate the document and retrieve document structure with contains both the XML data set and XSL stylesheet.	`public void pullDocument(Uuid docUuid) {` `doc = docFactory.pullDocument(docUuid);` `return null;` `}`
checkDocumentCriteria()	A method to lookup and retrieve the data criteria for a document.	`public String checkDocumentCriteria() {` `String validationResultsHolder;` `validationResultsHolder = doc.validateXML();` `return validationResultsHolder;` `}`
pullDataFromCaseManagement()	A method to retrieve data from the case management system based on the data criteria for the document selected and load or refresh the document with the data retrieved.	`public void pullDataFromCaseManagement(String caseRefId) {` `// Assumes Case UUID has been searched for and found` `doc.pullData(caseUuid);` `return null;` `}`



Class name: XMLDocumentInteractionFacade

	Method Description	Program Description Language
pushDataToCaseManagement()	A method to update the case management system when data elements are updated within a document in a workflow.	`public void pushDataToCaseManagement() {` `doc.pushData();` `}`
setType()	A method to set the type of document to generate.	`public void setType(int docTypeCode) {` `docFactory.setTypeCode(docTypeCode);` `}`
producePDF()	A method to combine the XML instance and XSL Stylesheet with the use of FOP libraries to create a PDF to produce to the user of the system.	`public URL producePDF(Uuid docUuid) {` `URL pdfLocation;` `pdfLocation = doc.ToPDF(docUuid);` `return pdfLocation;` `}`
saveState()	A method to save the state of the document.	`public String saveState(String docUuid) {` `String persistResult;` `persistResult = doc.persist();` `return persistResult;` `}`
setDataElement()	A method set which is overloaded and allows for modification of the different types of elements seen in an XML data set for a document.	`public void setDataElement(String newData, int elementId) {` `Uuid docUuid = doc.getUuid();` `doc.setDataElement(docUuid, elementId, newData);` `}`
associateDocWithWorkFlow()	A method set which is used to create an association with a workflow based on the workflow name and parameters passed to it.	`public void associateDocWithWorkFlow(String workFlowRefId, Uuid docUuid, Uuid addresses[]) {` `workflow.initiateNewWorkFlow(workFlowRefId, docUuid, addresses);` `}`



Class name: XMLDocumentInteractionFacade

	Method Description	Program Description Language
signDocument()	A method to generate a digital signature and insert it into the XML data set to become part of the document.	`public Uuid signDocument(Uuid docUuid, Uuid personUuid) {` `Uuid signatureUuid = signDoc(docUuid,  personUuid, );` `return signatureUuid;` `}`
updateAuditLog()	A method to update audit logs based on the actions taken.	`public int updateAuditLog(actorUuid, docUuid, actionCode) {` `int auditUpdateSuccessSignal;` `// 0 if successful, 1 if error` `auditLogUpdateSuccessSignal = auditLog.newEntry(actorUuid, docUuid, actionCode);` `return auditLogUpdateSuccessSignal;` `}`
readAuditLog()	A method to allow for returning a result set containing the audit log for a particular document.	`public ResultSet readAuditLog(Uuid docUuid) {` `auditLog.getEntries(docUuid);` `}`
generateDocument()	A method to generate a document using the type of document to generate and the data source of a case record. The method utilizes a factory pattern class to generate the document.	`public String generateDocument(int docTypeCode, Uuid caseUuid) {` `String createdDocUuid;` `docFactory.createDocument(docTypeCode, caseUuid);` `createdDocUuid = doc.getUuid().toString();` `return creatDocUuid.toString;` `}`
XMLDocumentInteractionFacade()	This is a constructor method for the class.	`public XMLDocumentInteractionFacade() {` `docFactory = new DocumentFactory();` `auditLog = new AuditLogFacade();` `docSigner = new DocumentSigner();` `workflow = new WorkFlowFacade();` `}`

FIG : Class diagram details [1]

7. Object Collaborations

- The object collaboration UML diagram is a representation of the architecture of the object residing in the system based on OOP. It is best used for analyzing use cases. The important components of this diagram are actors, links, messages and objects.[3]
- This contains the UML diagram representing not only the classes but their interactions and dependencies.

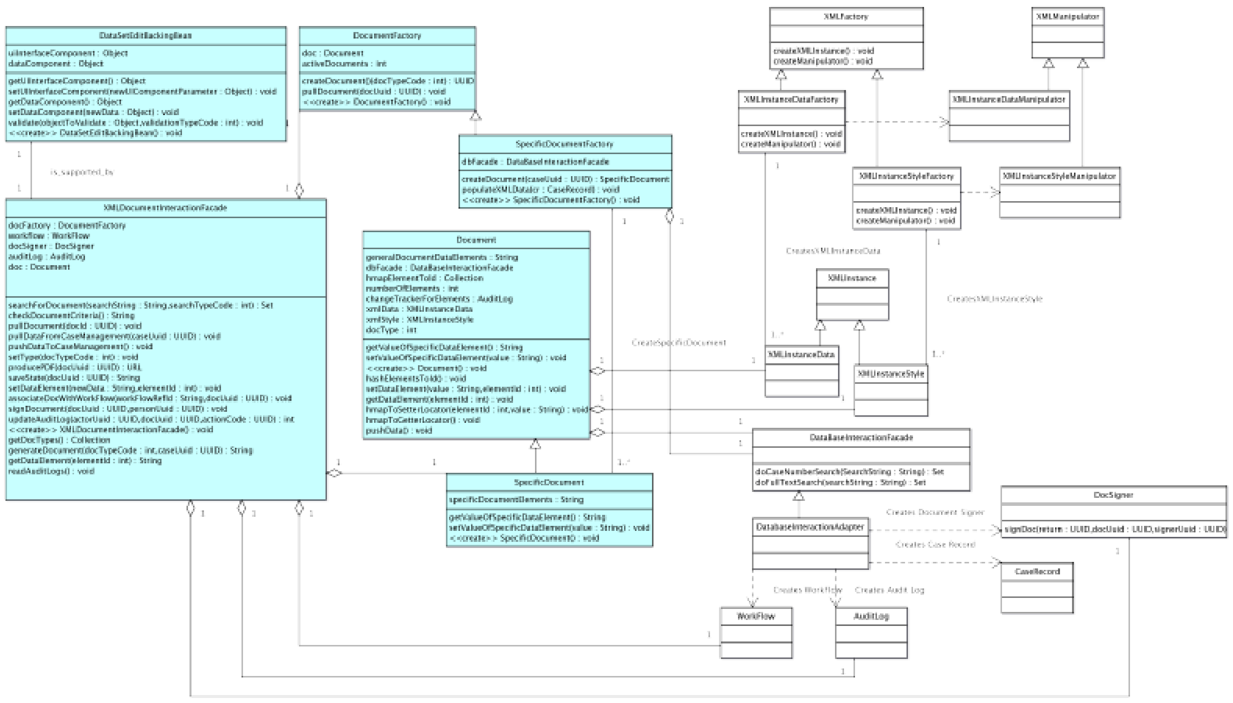

DataSetEditBackingBean
uiInterfaceComponent : Object
dataComponent : Object
getUiInterfaceComponent() : Object
setUiInterfaceComponent(newUIComponentParameter : Object) : void
getDataComponent() : Object
setDataComponent(newData : Object) : void
validate(objectToValidate : Object,validationTypeCode : int) : void
<<create>> DataSetEditBackingBean() : void
DocumentFactory
doc : Document
activeDocuments : int
createDocument()(docTypeCode : int) : UUID
pullDocument(docUuid : UUID) : void
<<create>> DocumentFactory() : void
is_supported_by
XMLDocumentInteractionFacade
docFactory : DocumentFactory
workflow : WorkFlow
docSigner : DocSigner
auditLog : AuditLog
doc : Document
searchForDocument(searchString : String,searchTypeCode : int) : Set
checkDocumentCriteria() : String
pullDocument(docId : UUID) : void
pullDataFromCaseManagement(caseUuid : UUID) : void
pushDataToCaseManagement() : void
setType(docTypeCode : int) : void
producePDF(docUuid : UUID) : URL
saveState(docUuid : UUID) : String
setDataElement(newData : String,elementId : int) : void
associateDocWithWorkFlow(workFlowRefId : String,docUuid : UUID) : void
signDocument(docUuid : UUID,personUuid : UUID) : void
updateAuditLog(actorUuid : UUID,docUuid : UUID,actionCode : UUID) : int
<<create>> XMLDocumentInteractionFacade() : void
getDocTypes() : Collection
generateDocument(docTypeCode : int,caseUuid : UUID) : String
getDataElement(elementId : int) : String
readAuditLogs() : void
SpecificDocumentFactory
dbFacade : DataBaseInteractionFacade
createDocument(caseUuid : UUID) : SpecificDocument
populateXMLData(cr : CaseRecord) : void
<<create>> SpecificDocumentFactory() : void
Document
generalDocumentDataElements : String
dbFacade : DataBaseInteractionFacade
hmapElementToId : Collection
numberOfElements : int
changeTrackerForElements : AuditLog
xmlData : XMLInstanceData
xmlStyle : XMLInstanceStyle
docType : int
getValueOfSpecificDataElement() : String
setValueOfSpecificDataElement(value : String) : void
<<create>> Document() : void
hashElementsToId() : void
setDataElement(value : String,elementId : int) : void
getDataElement(elementId : int) : void
hmapToSetterLocator(elementId : int,value : String) : void
hmapToGetterLocator() : void
pushData() : void
SpecificDocument
specificDocumentElements : String
getValueOfSpecificDataElement() : String
setValueOfSpecificDataElement(value : String) : void
<<create>> SpecificDocument() : void
CreateSpecificDocument
XMLFactory
createXMLInstance() : void
createManipulator() : void
XMLManipulator
XMLInstanceDataFactory
createXMLInstance() : void
createManipulator() : void
XMLInstanceDataManipulator
XMLInstanceStyleFactory
createXMLInstance() : void
createManipulator() : void
XMLInstanceStyleManipulator
CreatesXMLInstanceData
XMLInstance
CreatesXMLInstanceStyle
XMLInstanceData
XMLInstanceStyle
DataBaseInteractionFacade
doCaseNumberSearch(SearchString : String) : Set
doFullTextSearch(searchString : String) : Set
DocSigner
signDoc(return : UUID,docUuid : UUID,signerUuid : UUID)
DatabaseInteractionAdapter
Creates Document Signer
Creates Case Record
CaseRecord
Creates WorkFlow
Creates Audit Log
WorkFlow
AuditLog

FIG : UML diagram of the various objects interacting in the document utility [1]

8. Data Design

- The utility in design stores, accesses and modifies data and hence an Entity Relationship Diagram is in order.

8.1.1 Basic Entity Relationship Diagram

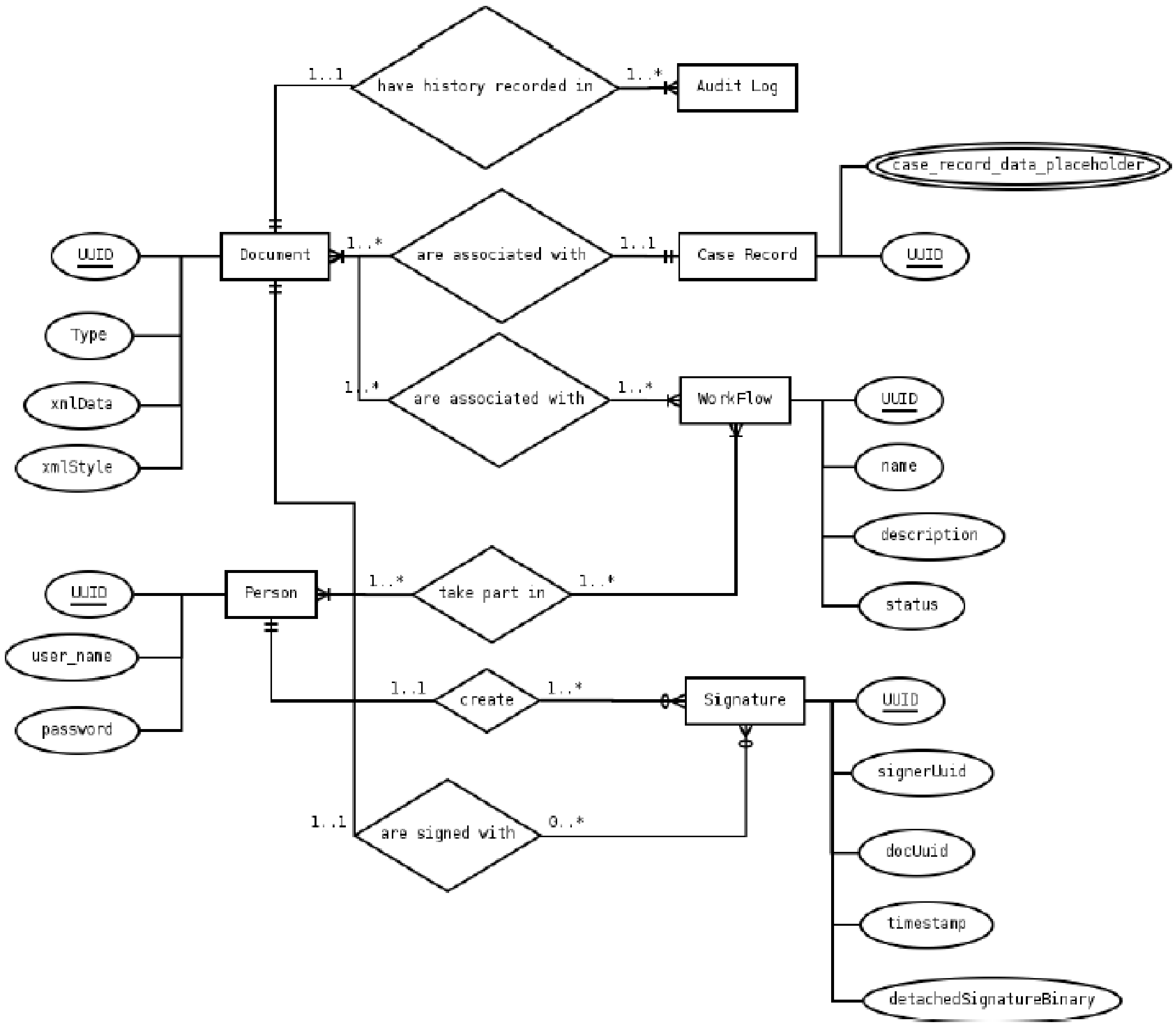


FIG : Entity Relationship Diagram [1]

9. Dynamic Model

- Sequence/event diagrams represent the flow of messages in a system. It is used to portray communication between any two life lines (participants in the diagram) as a time-ordered sequence of events that should take place at run-time.[4]
- A state diagram is a modeling of the various dynamic/static states that a system goes through. It shows the various states that an object goes through while moving through the system, thus capturing the software system’s behavior. [5]

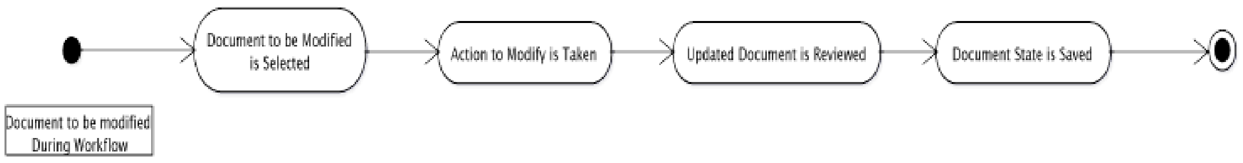

Document to be Modified is Selected
Action to Modify is Taken
Updated Document is Reviewed
Document State is Saved
Document to be modified During Workflow

FIG : Sequence diagram of moving from an unmodified original document to a modified document

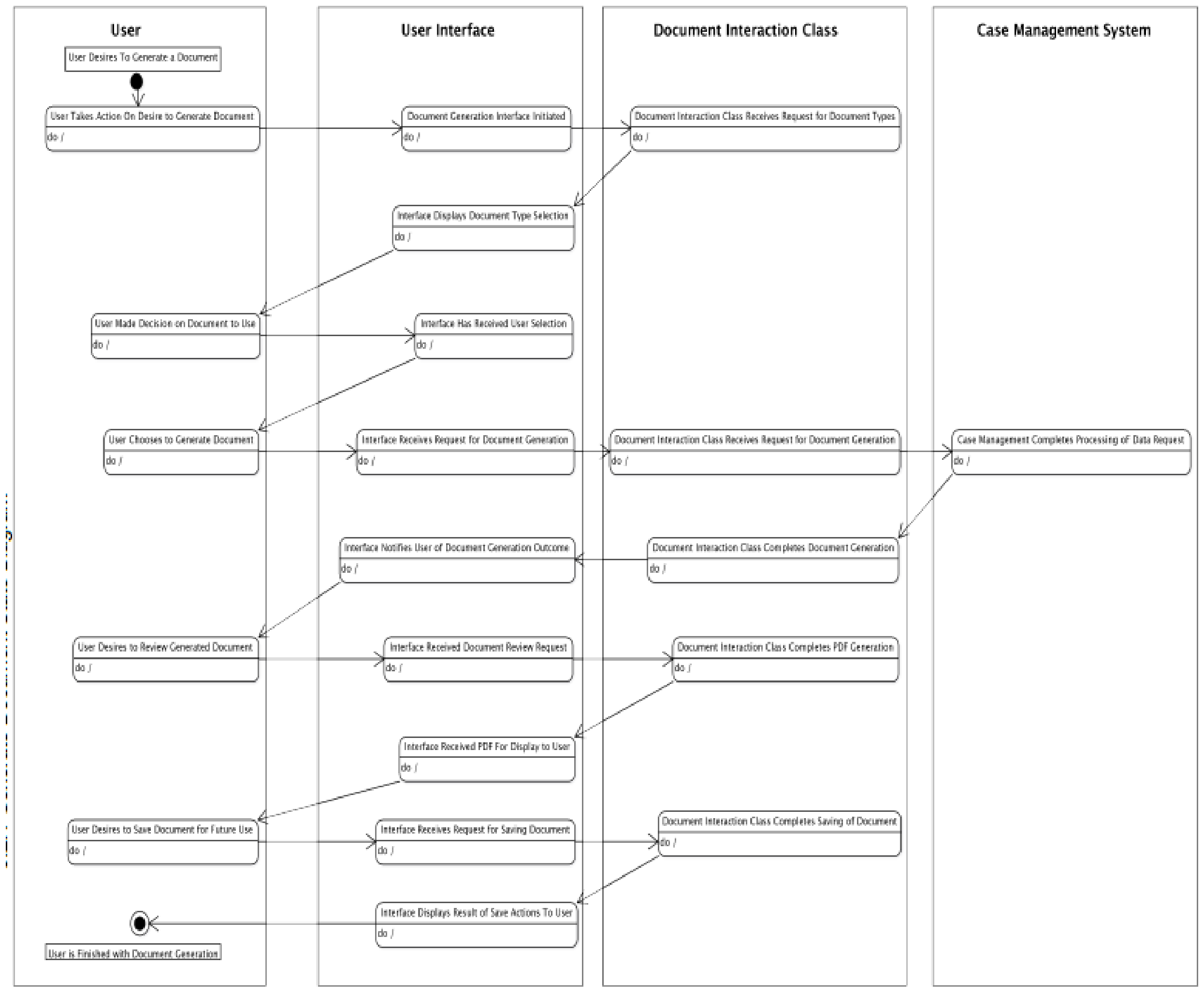
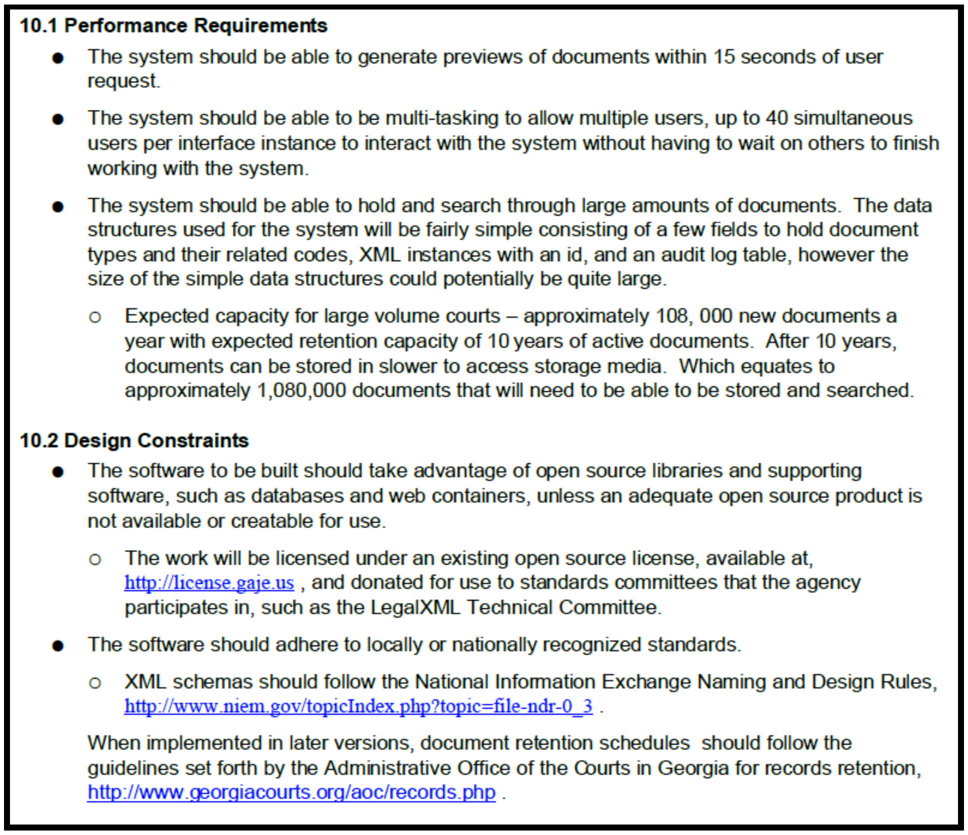

User
User Interface
Document Interaction Class
Case Management System
User Desires To Generate a Document
User Takes Action On Desire to Generate Document
do /
Document Generation Interface Initiated
do /
Document Interaction Class Receives Request for Document Types
do /
Interface Displays Document Type Selection
do /
User Made Decision on Document to Use
do /
Interface Has Received User Selection
do /
User Chooses to Generate Document
do /
Interface Receives Request for Document Generation
do /
Document Interaction Class Receives Request for Document Generation
do /
Case Management Completes Processing of Data Request
do /
Interface Notifies User of Document Generation Outcome
do /
Document Interaction Class Completes Document Generation
do /
User Desires to Review Generated Document
do /
Interface Received Document Review Request
do /
Document Interaction Class Completes PDF Generation
do /
Interface Received PDF For Display to User
do /
User Desires to Save Document For Future Use
do /
Interface Receives Request for Saving Document
do /
Document Interaction Class Completes Saving of Document
do /
Interface Displays Result of Save Actions To User
do /
User is Finished with Document Generation

FIG : Generate document state diagram

10. Non-functional Requirements

- This part lists performance requirements and design constraints, things not directly related to the design of the product i.e. it's functionality but rather system performance, things that make it a better product for use.

With respect to the product under design the constraints are shown below



10.1 Performance Requirements

- The system should be able to generate previews of documents within 15 seconds of user request.
- The system should be able to be multi-tasking to allow multiple users, up to 40 simultaneous users per interface instance to interact with the system without having to wait on others to finish working with the system.
- The system should be able to hold and search through large amounts of documents. The data structures used for the system will be fairly simple consisting of a few fields to hold document types and their related codes, XML instances with an id, and an audit log table, however the size of the simple data structures could potentially be quite large.
  - Expected capacity for large volume courts – approximately 108, 000 new documents a year with expected retention capacity of 10 years of active documents. After 10 years, documents can be stored in slower to access storage media. Which equates to approximately 1,080,000 documents that will need to be able to be stored and searched.

10.2 Design Constraints

- The software to be built should take advantage of open source libraries and supporting software, such as databases and web containers, unless an adequate open source product is not available or creatable for use.
  - The work will be licensed under an existing open source license, available at, http://license.gaje.us , and donated for use to standards committees that the agency participates in, such as the LegalXML Technical Committee.
- The software should adhere to locally or nationally recognized standards.
  - XML schemas should follow the National Information Exchange Naming and Design Rules, http://www.niem.gov/topicIndex.php?topic=file-ndr-0_3 .

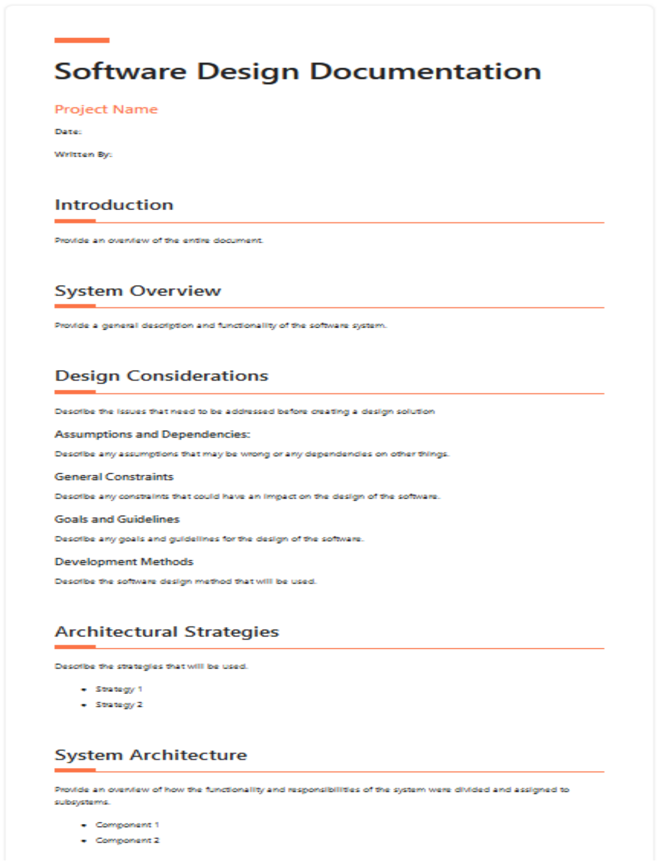
  When implemented in later versions, document retention schedules should follow the guidelines set forth by the Administrative Office of the Courts in Georgia for records retention, http://www.georgiacourts.org/aoc/records.php .

FIG : Non- functional requirements of the Document Utility [1]

11. Supplementary Documentation

This contains a reference to the tools used such as "ArgoUML" used to create the UML diagrams and "Dia" used for the Entity Relationship Diagram.

On a more general note, below are a few samples of some templates available for design documentation along with some more tools used in the design process.



Software Design Documentation

Project Name

Date:

Written By:

Introduction

Provide an overview of the entire document.

System Overview

Provide a general description and functionality of the software system.

Design Considerations

Describe the issues that need to be addressed before creating a design solution

Assumptions and Dependencies:

Describe any assumptions that may be wrong or any dependencies on other things.

General Constraints

Describe any constraints that could have an impact on the design of the software.

Goals and Guidelines

Describe any goals and guidelines for the design of the software.

Development Methods

Describe the software design method that will be used.

Architectural Strategies

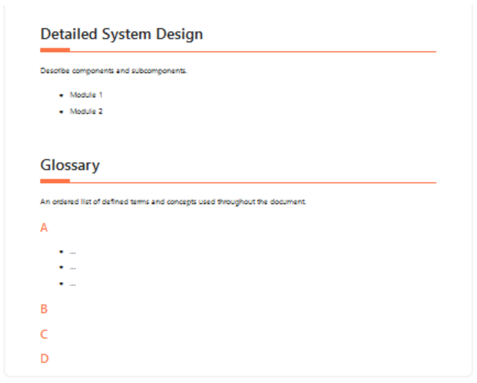
Describe the strategies that will be used.

- Strategy 1
- Strategy 2

System Architecture

Provide an overview of how the functionality and responsibilities of the system were divided and assigned to subsystems.

- Component 1
- Component 2



Detailed System Design

Describe components and subcomponents.

- Module 1
- Module 2

Glossary

An ordered list of defined terms and concepts used throughout the document.

A

- ...
- ...
- ...

B

C

D

FIG : Template for design document [6]

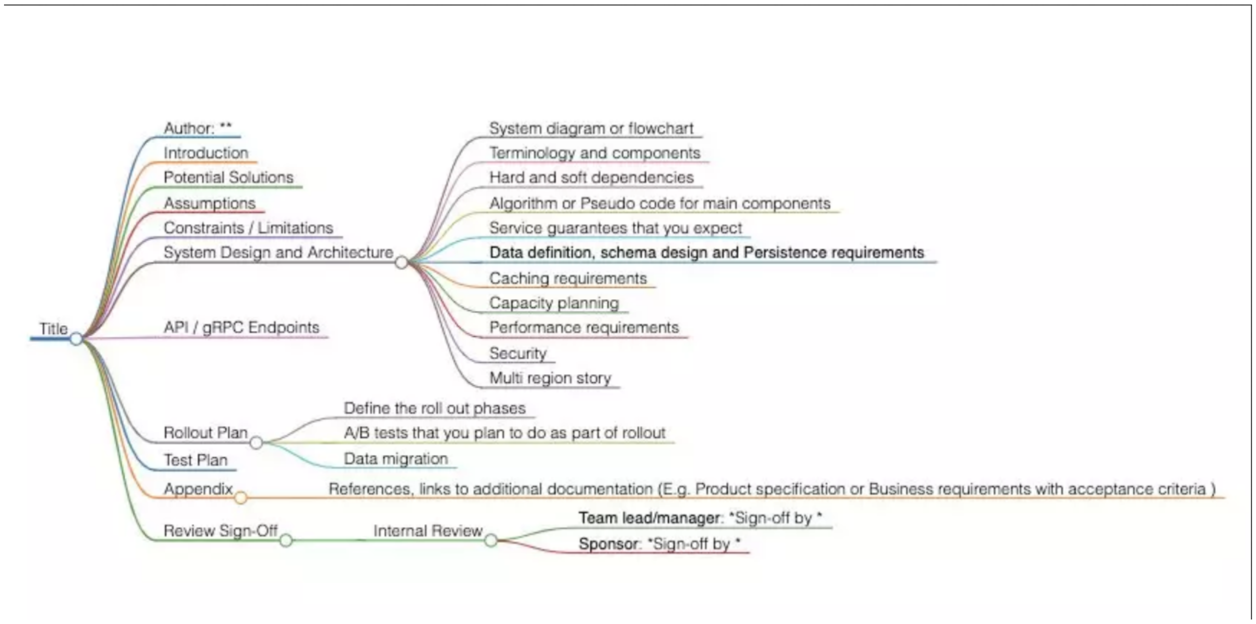

Title
Author: **
Introduction
Potential Solutions
Assumptions
Constraints / Limitations
System Design and Architecture
System diagram or flowchart
Terminology and components
Hard and soft dependencies
Algorithm or Pseudo code for main components
Service guarantees that you expect
Data definition, schema design and Persistence requirements
Caching requirements
Capacity planning
Performance requirements
Security
Multi region story
API / gRPC Endpoints
Rollout Plan
Define the roll out phases
A/B tests that you plan to do as part of rollout
Data migration
Test Plan
Appendix
References, links to additional documentation (E.g. Product specification or Business requirements with acceptance criteria)
Review Sign-Off
Internal Review
Team lead/manager: *Sign-off by *
Sponsor: *Sign-off by *

FIG : Template mentioning the sections essential in a design document ([7])

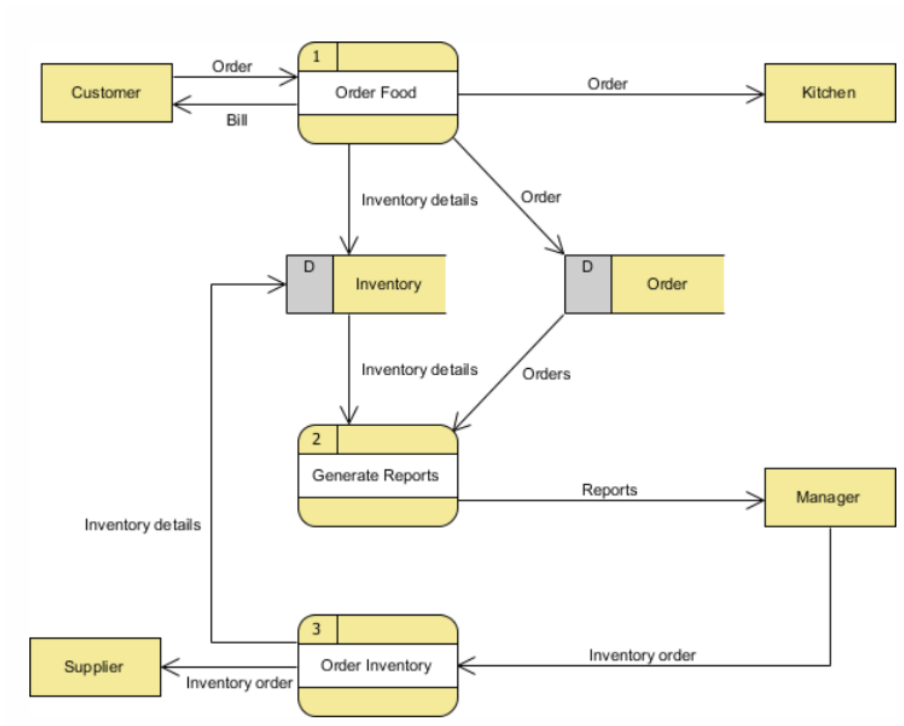


FIG : Data Flow Diagram ([8]) : Data Flow Diagram (DFD) provides a visual representation of the flow of information (i.e. data) within a system

Coding

The coding is the process of transforming the design of a system into a computer language format. This coding phase of software development is concerned with software translating design specification into the source code. It is necessary to write source code & internal documentation so that conformance of the code to its specification can be easily verified.

Coding is done by the coder or programmers who are independent people rather than the designer. The goal is not to reduce the effort and cost of the coding phase, but to cut to the cost of a later stage. The cost of testing and maintenance can be significantly reduced with efficient coding.

Goals of Coding

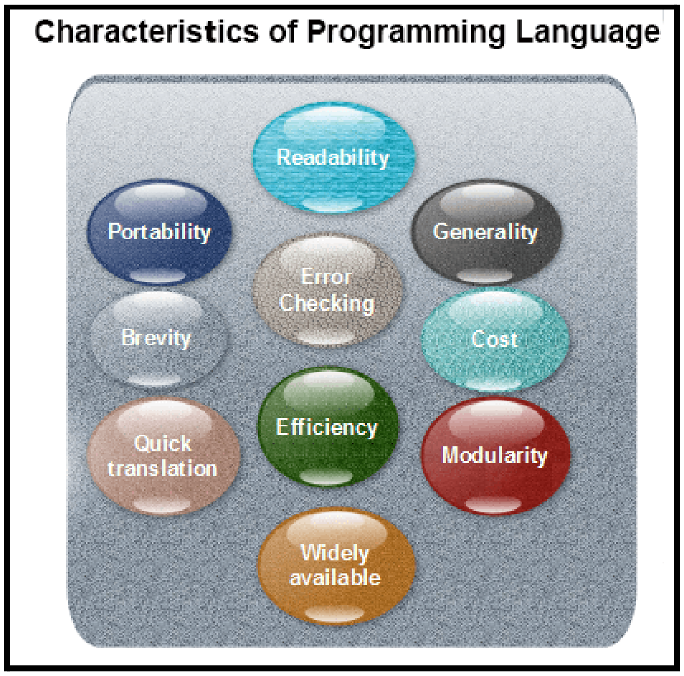
➢ To translate the design of system into a computer language format

➢ To reduce the cost of later phases

➢ Making the program more readable

For implementing our design into code, we require a high-level functional language.

❖ Characteristics of Programming Language



Readability: A good high-level language will allow programs to be written in some methods that resemble a quite-English description of the underlying functions. The coding may be done in an essentially self-documenting way.

Portability: High-level languages, being virtually machine-independent, should be easy to develop portable software.

Generality: Most high-level languages allow the writing of a vast collection of programs, thus relieving the programmer of the need to develop into an expert in many diverse languages.

Brevity: Language should have the ability to implement the algorithm with less amount of code. Programs in high-level languages are often significantly shorter than their low-level equivalents.

Error checking: A programmer is likely to make many errors in the development of a computer program. Many high-level languages invoke a lot of bugs checking both at compile-time and run-time.

Cost: The ultimate cost of a programming language is a task of many of its characteristics.

Quick translation: It should permit quick translation.

Efficiency: It should authorize the creation of an efficient object code.

Modularity: It is desirable that programs can be developed in the language as several separately compiled modules, with the appropriate structure for ensuring self-consistency among these modules.

Widely available: Language should be widely available, and it should be feasible to provide translators for all the major machines and all the primary operating systems.

❖ Coding Standards

A coding standard lists several rules to be followed during coding, such as the way variables are to be named, the way the code is to be laid out, error return conventions, etc.

General coding standards refers to how the developer writes code, so here we will discuss some essential standards regardless of the programming language being used.

1. **Indentation:** Proper and consistent indentation is essential in producing easy to read and maintainable programs.
   Indentation should be used to:
   - Emphasize the body of a control structure such as a loop or a select statement.
   - Emphasize the body of a conditional statement
   - Emphasize a new scope block
2. **Inline comments:** Inline comments analyze the functioning of the subroutine, or key aspects of the algorithm shall be frequently used.
3. **Rules for limiting the use of global:** These rules file what types of data can be declared global and what cannot.
4. **Structured Programming:** Structured (or Modular) Programming methods shall be used. "GOTO" statements shall not be used as they lead to "spaghetti" code, which is

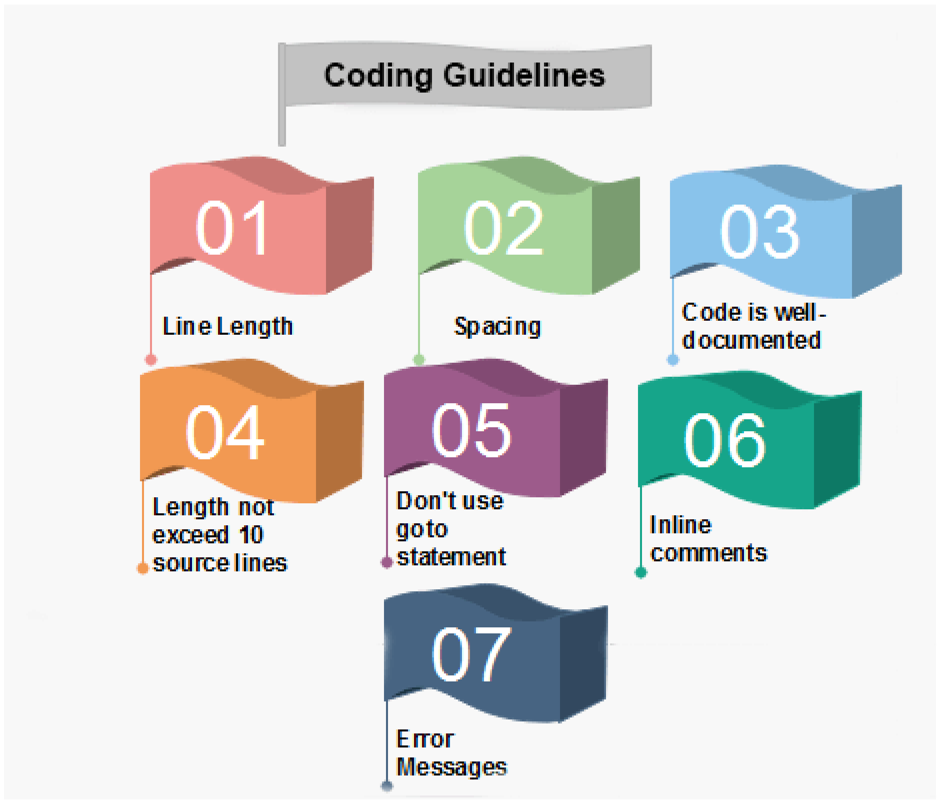
hard to read and maintain, except as outlined in the FORTRAN Standards and Guidelines.

5. **Naming conventions for global variables, local variables, and constant identifiers:** A possible naming convention can be that global variable names always begin with a capital letter, local variable names are made of small letters, and constant names are always capital letters.
6. **Error return conventions and exception handling system:** Different functions in a program report the way error conditions are handled should be standard within an organization. For example, different tasks while encountering an error condition should either return a 0 or 1 consistently.

❖ Coding Guidelines

General coding guidelines provide the programmer with a set of the best methods which can be used to make programs more comfortable to read and maintain. Most of the examples use the C language syntax, but the guidelines can be tested to all languages.

The following are some representative coding guidelines recommended by many software development organizations.



1. Line Length: It is considered a good practice to keep the length of source code lines at or below 80 characters. Lines longer than this may not be visible properly on some terminals and tools. Some printers will truncate lines longer than 80 columns.

2. Spacing: The appropriate use of spaces within a line of code can improve readability.

Example:

Bad: cost=price+(price*sales_tax)
fprintf(stdout ,"The total cost is %5.2f\n",cost);

Better: cost = price + (price * sales_tax)
fprintf (stdout,"The total cost is %5.2f\n",cost);

3. The code should be well-documented: As a rule of thumb, there must be at least one comment line on the average for every three-source line.

4. The length of any function should not exceed 10 source lines: A very lengthy function is generally very difficult to understand as it possibly carries out many various functions. For the same reason, lengthy functions are possible to have a disproportionately larger number of bugs.

5. Do not use goto statements: Use of goto statements makes a program unstructured and very tough to understand.

6. Inline Comments: Inline comments promote readability.

7. Error Messages: Error handling is an essential aspect of computer programming. This does not only include adding the necessary logic to test for and handle errors but also involves making error messages meaningful.

EXAMPLE:

Sample code for designing a login page:-

```
import java.util.Scanner;
public class Login_page {
    public static void main(String[] args)
{
        Scanner sc = new Scanner(System.in);
        String sp=" ";
        System.out.println("Enter the Username");
        String uname = sc.nextLine();
        if((uname.contains(sp)) || uname.length()<4)
{
            System.out.println("Invalid Username");
            return;
        }

        System.out.println("Enter the Password");
        String upass = sc.nextLine();
        if((upass.contains(sp)) || upass.length()<8)
{
            System.out.println("Invalid Password");
            return;
        }

        System.out.println(uname+" you are Registered Successfully");

        System.out.println("Enter the Username");
        String lname = sc.nextLine();
        System.out.println("Enter the Password");
        String lpass = sc.nextLine();

        if(uname.equals(lname) && upass.equals(lpass)){
            System.out.println("Welcome "+lname+" you have Logged-in Successfully");
        }
        else{
            System.out.println("Username or password Mismatch");
        }
    }
}
```

Testing

Two main forms of testing whitebox and blackbox are described below

❖ BLACKBOX TESTING / Behavioral testing

Black Box Testing is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. It mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications.

STEPS TO CARRY OUT BLACKBOX TESTING :

1. Initially, the requirements and specifications of the system are examined.
2. Tester chooses valid inputs (positive test scenario) to check whether SUT processes them correctly. Also, some invalid inputs (negative test scenario) are chosen to verify that the SUT is able to detect them.
3. Tester determines expected outputs for all those inputs.
4. Software tester constructs test cases with the selected inputs.
5. The test cases are executed.
6. Software tester compares the actual outputs with the expected outputs.
7. Defects if any are fixed and re-tested.

TYPES OF BLACKBOX TESTING

1. **Functional testing** – This black box testing type is related to the functional requirements of a system; it is done by software testers. Some tools used are QTP, Selenium.
2. **Non-functional testing** – This type of black box testing is not related to testing of specific functionality, but non-functional requirements such as performance, scalability, usability. Some tools used are LoadRunner
3. **Regression testing** – Regression Testing is done after code fixes, upgrades or any other system maintenance to check the new code has not affected the existing code.

WIDELY USED BLACKBOX TESTING TECHNIQUES

1. **Equivalence Class Testing:** It is used to minimize the number of possible test cases to an optimum level while maintaining reasonable test coverage.
2. **Boundary Value Testing:** Boundary value testing is focused on the values at boundaries. This technique determines whether a certain range of values are acceptable by the system or not. It is very useful in reducing the number of test cases. It is most suitable for the systems where an input is within certain ranges.
3. **Decision Table Testing**: A decision table puts causes and their effects in a matrix. There is a unique combination in each column.

❖ WHITEBOX TESTING / Clearbox testing

White Box Testing is software testing technique in which internal structure, design and coding of software are tested to verify flow of input-output and to improve design, usability and security. Since the code is visible to testers, it is also known as : Clearbox testing, Openbox testing, Transparentbox testing, Codebased testing and Glassbox testing. One of the basic goals of whitebox testing is to verify a working flow for an application. It involves testing a series of predefined inputs against expected or desired outputs so that when a specific input does not result in the expected output, you have encountered a bug.

THE FOLLOWING ASPECTS ARE TESTED IN WHITEBOX TESTING

1. Internal security holes
2. Broken or poorly structured paths in the coding processes
3. The flow of specific inputs through the code
4. Expected output
5. The functionality of conditional loops
6. Testing of each statement, object, and function on an individual basis

STEPS TO CARRY OUT WHITEBOX TESTING

1. UNDERSTAND THE SOURCE CODE

   The first thing a tester will often do is learn and understand the source code of the application. Since white box testing involves the testing of the inner workings of an application, the tester must be very knowledgeable in the programming languages used in the applications they are testing. Also, the testing person must be highly aware of secure coding practices. Security is often one of the primary objectives of testing software. The tester should be able to find security issues and prevent attacks from hackers and naive users who might inject malicious code into the application either knowingly or unknowingly.

2. CREATE TEST CASES AND EXECUTE

   The second basic step to white box testing involves testing the application's source code for proper flow and structure. One way is by writing more code to test the application's source code. The tester will develop little tests for each process or series of processes in the application. This method requires that the tester must have intimate knowledge of the code and is often done by the developer. Other methods include Manual Testing, trial, and error testing and the use of testing tools as we will explain further on in this article.

TYPES OF WHITEBOX TESTING

1. **Unit Testing:** It is often the first type of testing done on an application. Unit Testing is performed on each unit or block of code as it is developed. Unit Testing is essentially done by the programmer. As a software developer, you develop a few lines of code, a

single function or an object and test it to make sure it works before continuing Unit Testing helps identify a majority of bugs, early in the software development lifecycle. Bugs identified in this stage are cheaper and easy to fix.

2. **Testing for Memory Leaks**: Memory leaks are leading causes of slower running applications. A QA specialist who is experienced at detecting memory leaks is essential in cases where you have a slow running software application.

ADVANTAGES OF WHITEBOX TESTING

1. Code optimization by finding hidden errors.
2. White box test cases can be easily automated.
3. Testing is more thorough as all code paths are usually covered.
4. Testing can start early in SDLC even if GUI is not available.

Maintenance

After all the above processes of Software Development, the Product is in public, and people are using it. There will be situations where users might face problems in terms of bad User experience or some Program Bugs. Fixing those issues are necessary to satisfy user expectations and experience. This is achieved by performing maintenance. This is the phase for all the future revisions of the product to fix bugs, improve software design, enhance the product, accommodate new features and hardware, etc.

❖ Categories of Software Maintenance

1. **Corrective Maintenance**
    a. Rectify/Correct bugs in the product found by users
    b. Enhance the performance of the product
2. **Adaptive Maintenance**
    a. Updates to the software to allow products to run in new environments.
3. **Perfective Maintenance**
    a. New Features for the product for the users
4. **Preventive Maintenance**
    a. Fixes to prevent problems in the product in future.

❖ Case Study

Windows Insider

a. What this is:
    i. A Beta testing program on which Microsoft, as a company, can evaluate their future versions of the Windows Operating System.
b. How it works
    i. Users who want to try beta testing in windows can sign up to this program to get unstable versions of the Operating System and can take part in the testing and maintenance of the Windows.
    ii. If any bugs and inconsistencies are found in the operating system, the users can send their reports in an app name “Feedback Hub” which send the reports directly to the Microsoft Windows Development Division
    iii. From there, if the reports are valid, engineers at Microsoft fix those issues
    iv. They also provide new releases for users to evaluate and generate any error report.
    v. This entire process of bug reports and fixing happens weekly and in the span of months, provides a legitimate and stable version of windows that can be pushed as an update to the public.

REFERENCES FOR FEASIBILITY STUDY

1. Feasibility Study in SDLC - FreeEducator.com
2. Feasibility study
3. Sample Software Engineering Feasibility Study Report

REFERENCES FOR REQUIREMENT ANALYSIS

1. SRSExample-webapp.doc (live.com)
2. SRS4.0.doc (live.com)
3. Requirements Document Example
4. BalticLSC Software User Requirements Specification
5. (TECHNICAL SPECIFICATION) DEVELOPMENT OF WEB APPLICATION FOR AMIS/ZPIS
6. https://jelvix.com/blog/software-requ...
7. https://jelvix.com/blog/functional-vs...

REFERENCES FOR DESIGN

1. Example XML Legal Document Utility Software Design Document
2. UML Use Case Diagram - Javatpoint
3. UML Collaboration Diagram - Javatpoint
4. UML Sequence Diagram - Javatpoint
5. https://www.javatpoint.com/uml-state-machine-diagram
6. Software Design Document: What is it & How to Create it! (Template Included)
7. Best Software Design Document Templates - DevTeam.Space
8. Data Flow Diagram: Examples - Food Ordering System

REFERENCES FOR CODING

1. Java Program to Create Login Page - Quiz for Exam
2. Software Engineering | Coding - javatpoint

REFERENCES FOR TESTING

1. Manual Testing Tutorial: What is, Concepts, Types & Tool
2. Unit Testing Tutorial: What is, Types, Tools & Test EXAMPLE

REFERENCES FOR MAINTENANCE

1. Windows Insider