###### 

Department of Computer Science &

Information Technology

**Final Project Guide Book**

**Version 1.0**

**Revision History**

This section describes the revision history of this document.

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description of Change** | **Author** |
| September 25, 2012 | 1.0 | First Draft of Final Project Deliverable Guideline | Project Coordination Office |
|  |  |  |  |
|  |  |  |  |

**Distribution List**

This section describes the distribution list for the recipients of this document.

|  |  |  |  |
| --- | --- | --- | --- |
| **Recipient** | **Role / Designation** | **Office** | **Contact Details** |
| Dr. MA Pasha | Chairman | Chairman, Department of Computer Science & Information Technology | [marpasha@yahoo.com](mailto:marpasha@yahoo.com) |
|  |  |  |  |

Table of Contents

[Chapter 1: Final Project Proposal 7](#_Toc336382722)

[1.1 Introduction 7](#_Toc336382723)

[1.2. Project Title: 8](#_Toc336382724)

[1.3. Project Overview statement: 8](#_Toc336382725)

[1.4. Project Goals & Objectives: 10](#_Toc336382726)

[1.5. High-level system components: 10](#_Toc336382727)

[1.6. List of optional functional units: 10](#_Toc336382728)

[1.7. Exclusions: 11](#_Toc336382729)

[1.8. Application Architecture: 11](#_Toc336382730)

[1.9. Gantt chart: 11](#_Toc336382731)

[1.10. Hardware and Software Specification: 12](#_Toc336382732)

[1.11. Tools and technologies used with reasoning: 12](#_Toc336382733)

[Chapter 2: First Deliverable 13](#_Toc336382734)

[2.1. Introduction 13](#_Toc336382735)

[2.2. Project/Product Feasibility Report 14](#_Toc336382736)

[2.2.1. Technical Feasibility 14](#_Toc336382737)

[2.2.2. Operational Feasibility 14](#_Toc336382738)

[2.2.3. Economic Feasibility 14](#_Toc336382739)

[2.2.4. Schedule Feasibility 15](#_Toc336382740)

[2.2.5. Specification Feasibility 15](#_Toc336382741)

[2.2.6. Information Feasibility 15](#_Toc336382742)

[2.2.7. Motivational Feasibility 15](#_Toc336382743)

[2.2.8. Legal & Ethical Feasibility 15](#_Toc336382744)

[2.3. Project/Product Scope 15](#_Toc336382745)

[2.4. Project/Product Costing 16](#_Toc336382746)

[2.4.1. Project Cost Estimation By Function Point Analysis 16](#_Toc336382747)

[2.4.2. Project Cost Estimation by using COCOMO’81 (Constructive Cost Model) 18](#_Toc336382748)

[2.4.3. Activity Based Costing 19](#_Toc336382749)

[2.5. Task Dependency Table 19](#_Toc336382750)

[2.6. CPM - Critical Path Method 20](#_Toc336382751)

[2.7. Gantt chart 23](#_Toc336382752)

[2.8. Introduction to Team member and their skill set 24](#_Toc336382753)

[2.9. Task and Member Assignment Table 24](#_Toc336382754)

[2.10. Tools and Technology with reasoning 27](#_Toc336382755)

[2.11. Vision Document 27](#_Toc336382756)

[2.12. Risk List 28](#_Toc336382757)

[2.13. Product Features/ Product Decomposition 29](#_Toc336382758)

[Chapter 3: Second Deliverable For Object Oriented Approach 30](#_Toc336382759)

[3.1 Introduction: 30](#_Toc336382760)

[3.1.1 Systems Specifications 31](#_Toc336382761)

[3.1.2. Identifying External Entities 32](#_Toc336382762)

[3.1.3. Context Level Data Flow Diagram: 32](#_Toc336382763)

[3.1.4. Capture "shall" Statements: 32](#_Toc336382764)

[3.1.5. Allocate Requirements: 32](#_Toc336382765)

[3.1.6. Prioritize Requirements: 33](#_Toc336382766)

[3.1.7. Requirements Trace-ability Matrix: 33](#_Toc336382767)

[3.2. Example: 33](#_Toc336382768)

[3.2.1. Introduction 33](#_Toc336382769)

[3.2.2. Existing System 33](#_Toc336382770)

[3.2.3. Scope of the System 34](#_Toc336382771)

[3.2.4. Summary of Requirements:(Initial Requirements) 35](#_Toc336382772)

[3.2.5. Identifying External Entities: 36](#_Toc336382773)

[3.2.6. Capture "shall" Statements: 37](#_Toc336382774)

[3.2.7. Allocate Requirements: 38](#_Toc336382775)

[3.2.8. Priorities Requirements: 39](#_Toc336382776)

[3.2.9. Requirements Traceability Matrix: 43](#_Toc336382777)

[3.2.10. High Level Usecase Diagram: 44](#_Toc336382778)

[3.2.11. Analysis Level Usecase Diagram: 47](#_Toc336382779)

[3.2.12. Usecase Description 47](#_Toc336382780)

[Chapter 4: Third Deliverable For Object Oriented Approach 49](#_Toc336382781)

[4.1. Introduction: 50](#_Toc336382782)

[4.2. Domain Model 50](#_Toc336382783)

[4.3. System Sequence Diagram 51](#_Toc336382784)

[4.4. Sequence Diagram 51](#_Toc336382785)

[4.4.1. Defining a Sequence diagram 52](#_Toc336382786)

[4.4.2. Basic Sequence Diagram Symbols and Notations 52](#_Toc336382787)

[4.4.3. Example 55](#_Toc336382788)

[4.4.4. Distributing Control Flow in Sequence Diagrams 55](#_Toc336382789)

[4.5. Collaboration Diagram 58](#_Toc336382790)

[4.5.1. Contents of Collaboration Diagrams 59](#_Toc336382791)

[4.5.2. Constructs of Collaboration Diagram: 60](#_Toc336382792)

[4.6. Operation Contracts 61](#_Toc336382793)

[4.7. Design Class Diagram 61](#_Toc336382794)

[4.7.1. Create Initial Design Classes 62](#_Toc336382795)

[4.7.2. Designing Boundary Classes 62](#_Toc336382796)

[4.7.3. Designing Entity Classes 62](#_Toc336382797)

[4.7.4. Designing Control Classes 62](#_Toc336382798)

[4.7.5. Identify Persistent Classes 63](#_Toc336382799)

[4.7.6. Define Class Visibility 64](#_Toc336382800)

[4.7.7. Design Class Relationships 68](#_Toc336382801)

[4.8. State chart diagram 72](#_Toc336382802)

[4.9. Data Model 73](#_Toc336382803)

[Chapter 5: 2nd  & 3rd Deliverable For structured Approach 78](#_Toc336382804)

[5.1. Introduction: 79](#_Toc336382805)

[5.2. Entity Relationship Diagram: 79](#_Toc336382806)

[5.3. Data flow diagram (Functional Model) 81](#_Toc336382807)

[5.4. State Transition Diagram 85](#_Toc336382808)

[5.5. Architectural design 86](#_Toc336382809)

[5.6. Component Level Design 90](#_Toc336382810)

[Chapter 6: 4th Deliverable (User Interface Design) 91](#_Toc336382811)

[6.1. Introduction 92](#_Toc336382812)

[6.2. Site Maps 92](#_Toc336382813)

[6.3. Story boards 93](#_Toc336382814)

[6.4. Navigational maps: 94](#_Toc336382815)

[S1: Microsoft Products 94](file:///C:\Users\Dr.Pasha\Desktop\UOSproject%20office\Complete%20Final%20Project%20Guidelins,%20deliverables,%20templates,%20evaluation%20forms-ver1.doc#_Toc336382816)

[6.5 Trace-ability Matrix 95](#_Toc336382817)

[Chapter 7: 5th Deliverable (Software Testing) 96](#_Toc336382818)

[7.1 Introduction: 97](#_Toc336382819)

[7.2. Test plan 97](#_Toc336382820)

[7.2.1. Purpose 97](#_Toc336382821)

[7.2.2. Outline 97](#_Toc336382822)

[7.3. Test design specification 101](#_Toc336382823)

[7.3.1. Purpose 101](#_Toc336382824)

[7.3.2. Outline 101](#_Toc336382825)

[7.4. Test Case Specification 104](#_Toc336382826)

[7.4.1. Purpose 104](#_Toc336382827)

[7.4.2. Outline 104](#_Toc336382828)

[7.5. Test procedure specification 106](#_Toc336382829)

[7.5.1. Purpose 106](#_Toc336382830)

[7.5.2 Outline 106](#_Toc336382831)

[7.6. Test item transmittal report 107](#_Toc336382832)

[7.6.1. Purpose 108](#_Toc336382833)

[7.6.2. Outline 108](#_Toc336382834)

[7.7. Test log 109](#_Toc336382835)

[7.7.1. Purpose 109](#_Toc336382836)

[7.7.2. Outline 109](#_Toc336382837)

[7.8. Test incident report 110](#_Toc336382838)

[7.8.1. Purpose 110](#_Toc336382839)

[7.8.2. Outline 110](#_Toc336382840)

[7.9. Test summary report 111](#_Toc336382841)

[7.9.1. Purpose 111](#_Toc336382842)

[7.9.2. Outline 111](#_Toc336382843)

[Appendixes: 113](#_Toc336382844)

[Appendix 1: User Interface 114](#_Toc336382845)

[1.1. Introduction 114](#_Toc336382846)

[1.2. Focus on users 114](#_Toc336382847)

[1.3. Integrated with design 114](#_Toc336382848)

[1.4. Early user testing 115](#_Toc336382849)

[1.5. Iterative design 115](#_Toc336382850)

[1.6. Guidelines 115](#_Toc336382851)

[Appendix 2: Guidelines for Research Projects 130](#_Toc336382852)

[Appendix 3: Final Documentation Format Guidelines 132](#_Toc336382853)

[Appendix 4:Final Documentation Intial Pages 136](#_Toc336382854)

[Appendix 5: Final Evaluation Matrix for Development Projects 140](#_Toc336382855)

[Appendix 6: Final Evaluation Matrix for Research Projects 142](#_Toc336382856)

[Appendix 7: Final Evaluation Matrix for Network Projects 143](#_Toc336382857)

[Appendix 8: Final Evaluation Metric 144](#_Toc336382858)

[Appendix 9: Project Registration Form 145](#_Toc336382859)

[Appendix 11: Evaluation Delay Request Form 147](#_Toc336382860)

[Appendix 12: Show Cause Form 148](#_Toc336382861)

[Appendix 13: Project Renrollemnt Form 149](#_Toc336382862)

[Appendix 14: Change Request Form 150](#_Toc336382863)

# Chapter 1: Final Project Proposal

## 1.1 Introduction

This guide will tell you how to prepare and submit the final project proposal that is the documented work for the Project. A good project proposal must define the functional and non-functional requirements in unambiguous statements, Scope of the Project, Development Schedule, Development Process, Techniques, Tools, Platform with reasoning. However, a professional and well-defined proposal should be composed under the following headings;

1. Project Title
2. Project Overview Statement
3. Project Goals
4. Project Objectives
5. High Level System Components
   1. Component No.1
   2. Component No.2
   3. Component No.3
   4. Component No.4
      1. ………..
6. List of Optional Functional Units
7. Exclusions
8. Application Architecture
9. Gantt chart

## 1.2. Project Title:

The title should be clear and unambiguous (do not make it "cute"). Think of your title as a mini-abstract. A good title should paint a quick picture for the reader of the key idea(s) of your project. The words you use in your title should clearly reflect the focus of your proposal. The most important words should come first, then the less important words. Try to remove words from your title that really are not necessary for understanding. Try and use only a single sentence for your title. If the sentence is getting too long try removing some words. When all else fails try using a two-part title with the parts separated by a colon (use only as a last resort!). Do not attempt to use the title as an abstract of your entire proposal. If your proposal is built on collaborating with other groups/organizations it is usually a good idea to include their names on the Title/Cover Page.

## 1.3. Project Overview statement:

Think of the Project Overview as an Executive Summary (the busy executive probably only has enough time to read your Overview - not the entire proposal). Be specific and concise. Do not go into detail on aspects of your proposal that are further clarified at a later point in your proposal. The Project Overview should "paint a picture" of your proposal in the mind of the reader. It should establish the framework so that the rest of the proposal has a frame of reference. Use the Project Overview to begin to show your knowledge of the organization from which you are requesting funds. Key concerns of the funding organization can be briefly identified in relation to your proposed project. If you will be collaborating with other organizations make sure some of their interests are also highlighted in the Project Overview. This can assist in strengthening the collaboration by recognizing them at the very beginning of your proposal. The best time to prepare the Project Overview is after you have completed the entire proposal (and you understand all aspects of your proposal very well). Let the Overview be your last piece of writing and then insert it at the beginning of your proposal. Try to keep in mind that someone will be reviewing your proposal and you would like to have this person be very positive about what you have written. The Project Overview will probably form a strong impression in the mind of the reviewer. Work on your Project Overview so that you can avoid giving this person the opportunity to say things like:

1. Not an original idea
2. Rationale is weak
3. Writing is vague
4. Uncertain outcomes
5. Does not have relevant experience
6. Problem is not important
7. Proposal is unfocused
8. Project is too large.

Project Overview Statement Template

|  |
| --- |
| Project Title: |
| Project Manager: |
| Project Members:   |  |  |  |  | | --- | --- | --- | --- | | Name | Registration # | Email Address | Signature | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |
| Project Goal: |
| Objectives:   |  |  | | --- | --- | | Sr.# |  | | 1 |  | | 2 |  | | 3 |  | | 4 |  | | 5 |  | | 6 |  | |
| Project Success criteria: |
| Assumptions, Risks and Obstacles: |
| Organization Address (if any): |
| Type of project: 📺Research 📺Development |
| Target End users: |
| Development Technology: 📺Object Oriented 📺Structured |
| Platform: 📺Web based 📺Distributed  📺Desktop based 📺Setup Configurations  📺Other\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Approved By: |
| Date: |

## 1.4. Project Goals & Objectives:

Try and differentiate between your goals and your objectives - and include both. Goals are the large statements of what you hope to accomplish but usually aren't very measurable. They create the setting for what you are proposing. Objectives are operational, describe specific things you will be accomplishing in your project, and are very measurable. Your objectives will form the basis for the activities of your project and will also serve as the basis for the evaluation of your project. Try to insure that there is considerable overlap between the goals and objectives for your proposal and the goals and objectives of the funding organization. If there is not a strong overlap of goals and objectives then it might be best to identify a different funding organization. Measurable objectives for your project should be presented. If you are dealing with "things" it is easier for them to be measured than if you are dealing with abstract ideas. Your proposal is easier for a prospective funding organization to understand (and the outcomes are much more clear) if you describe your objectives in measurable ways.

## 1.5. High-level system components:

Information about the main functional units of the entire system should be present. Functional units to be included will be the inclusive components of the project developed so that the system must perform without taking any physical constraint into consideration. High-level system components are generally, a set of cooperating components assembled together to deliver a solution to a problem. They are frequently identified in terms of inputs, outputs, processes, and stored data that are needed to satisfy the system improvement objectives. If these components are missing the system fails to fulfill its primary mission.

## 1.6. List of optional functional units:

A list of functional units should be present which would include a description of other features, characteristics, and constraints that define a satisfactory system. These functional units would be developed under certain conditions (technology, expertise, or time dependent). Examples of these optional functional units would include performance (throughput and response time); ease of learning and use; budgets, costs, and cost savings; timetables and deadline; documentation and training needs; quality management; and security and internal auditing controls.

They are often requirements that specify need of compliance with any legal and regulatory requirements. They can also be design constraints due to the operating system used, the platform environment, compatibility issues, or any application standards that apply. In general, you can say that any requirement that does not allow for more than one design option should be regarded as a design constraint.

If the optional functional units are missing the system can still (for a while) fulfill its fundamental mission, but with degraded service quality.

While gathering and validating the optional functional requirements, maintain Assumptions and Issues lists.

Some activities will not give you satisfactory answers. This can be due to lack of information, or simply because you consider the answer threatens the viability of the design. Therefore, create two lists, and maintain them through the design study:

Any assumptions you make during the requirements and design process, including the rationale or thought processes behind those assumptions. Assumptions may be used to identify related subprojects or items of work, which are outside the scope of or after this project any major issues (significant concerns that could become show-stoppers).

The issues should be reviewed with the customer at the end of each phase. The assumptions need to be reviewed also, at the end of each phase, but the customer might not always be the correct person for the less important ones.

Assumptions and issues apply to all artifacts, but are particularly common for non-functional requirement.

## 1.7. Exclusions:

A list of the functional units, which will not be intended to be develop or discussed during any point in the project development, should be present. Time constraints or lack of resources for the fulfillment of the required task or any sort of other constraint preventing the completion of the functional unit could be described here.

## 1.8. Application Architecture:

Defines the overall application architecture e.g. a two-tier architecture or a three-tier architecture. It must contain a diagram depicting the system architecture properly

Architecture is the highest-level concept of a system in its environment. The architecture of a software system (at a given point in time) is its organization or structure of significant components interacting through interfaces, those components being composed of successively smaller components and interfaces.

Architecture can also be defined as the organizational structure of a system. Architecture can be recursively decomposed into parts that interact through interfaces, relationships that connect parts, and constraints for assembling parts. Parts that interact through interfaces include classes, components and subsystems.

There are a number of typical patterns of distribution in systems, depending on the functionality of the system and the type of application. In many cases, the distribution pattern is informally used to describe the 'architecture' of the system, though the full architecture encompasses this but also many more things. For example, many times a system will be described as having’ client-server architecture', although this is only the distribution aspect of the architecture.

## 1.9. Gantt chart:

The Gantt chart enumerates the activities to be performed on the vertical axis and their corresponding duration on the horizontal axis. It is possible to schedule activities by either early start or late start logic. In the early start approach; each activity is initiated as early as possible without violating the precedence relations. In the late start approach; each activity is delayed as much as possible as long as the earliest finish time of the project is not compromised.

Based on the Work Breakdown Structure (WBS), a timeline or Gantt chart showing the allocation of time to the project phases or iterations should be developed. This Gantt chart would identify major milestones with their achievement criteria. It must contain duration estimation of all the necessary activities to be carried out during the project development along with the human resources responsible for the respective tasks. Activity dependencies are also required to be mentioned in it.

Sample Gantt chart



## 1.10. Hardware and Software Specification:

Any hardware or software specifications e.g. machine type required, operating system and other utilities should be clearly specified for the system to be developed.

## 1.11. Tools and technologies used with reasoning:

The application tools, which are to be used on front and back end of the system to be developed, should be listed. The reasons for these tools should also be enlisted.

Identify what the needs for tool support are, and what the constraints are, by looking at the following:

* The development process. What tool support is required to effectively work? For example, if the organization decide to employ an iterative development process, it is necessary to automate the tests, since you will be testing several times during the project.
* Host (or development) platform(s).
* Target platform(s).
* The programming language(s) to be used.
* Existing tools. Evaluate any existing and proven tools and decide whether they can continue to be used.
* The distribution of the development organization. Is the organization physically distributed? Development tools generally support a physically distributed organization differently.
* The size of the development effort. Tools support large organizations more or less well.
* Budget and time constraints

# Chapter 2: First Deliverable

## 2.1. Introduction

First deliverable is all about planning and scheduling of project. This deliverable must contain following artifacts:

1. Project Feasibility
2. Project Scope
3. Project Costing
4. Task Dependency Table
5. Critical Path Method Analysis (CPM Analysis)
6. Gantt Chart
7. Introduction to team members
8. Tasks and member assignment table
9. Tools and Technologies
10. Vision Document
11. Risk List
12. Product Features

## 2.2. Project/Product Feasibility Report

When a project is started the first matter to establish is to assess the feasibility of a project or product. Feasibility means the extent to which appropriate data and information are readily available or can be obtained with available resources such as staff, expertise, time, and equipment. It is basically used as a measure of how practical or beneficial the development of a software system will be to you (or organization). This activity recurs throughout the life cycle.

There are many types of feasibilities:

* Technical
* Operational
* Economic
* Schedule
* Specification
* Information
* Motivational
* Legal and Ethical

### 2.2.1. Technical Feasibility

Technical Feasibility deals with asking the question as to whether the system can be developed or not. It is one of the most important questions before starting the project because it is assessing the limits of theory or technology applicable to the project. Another important query to be answered is to evaluate whether you (the project members or organization) possess the technology and technical expertise.

### 2.2.2. Operational Feasibility

Evaluation of technical ability of the staff to operate the project is the main aim of operational feasibility. In this area the question arises as to whether the problem is worth solving and if the solution provided for the problem works or not. How do end users and managers feel about the problem or solution is another query to be answered.

### 2.2.3. Economic Feasibility

Justification for the benefit/cost analysis relative to the project is to be measured in economic feasibility. Therefore, economic feasibility can be divided into two parts; cost estimates and benefit estimates. Cost estimates can further be alienated into development or acquisition costs (one time) and maintenance and operation costs (ongoing). In order to find development costs, break the project into tasks and use the lifecycle cost models. Experienced costs gained from similar projects should then be used to make estimates. The function point metric should be calculated.

Benefit estimates enclose tangible benefits and intangible benefits. Tangible benefits would include reduced costs and increased revenues. However, information quality, job satisfaction, and external standing are examples of intangible benefits.

### 2.2.4. Schedule Feasibility

Time is an important factor. The assessment and evaluation of the completion of a project with the available staff and resources within time is very essential. Meeting deadlines and milestones should always be kept in mind.

### 2.2.5. Specification Feasibility

Requirements are the features that the system must have or a constraint that must be accepted for the customer. The question arises as to whether the requirements are clear and definite. The scope boundaries must also be assessed.

### 2.2.6. Information Feasibility

The feasibility of information must be assessed regarding its completion, reliability, and meaningfulness.

### 2.2.7. Motivational Feasibility

Evaluation of the client staff regarding the motivation to perform the necessary steps correctly and promptly must occur.

### 2.2.8. Legal & Ethical Feasibility

”Do any infringements or liabilities arise from this project? “ is the main focus of this feasibility.

## 2.3. Project/Product Scope

Scope is a very dominant factor. Scope and context are both intertwined as both involve the boundaries of a system. Context would be referring to what holds outside the boundary the system. While scope would indicate whatever is inside the boundary of the system.

The scope of a project is defined by the set of requirements allocated to it. Managing project scope to fit the available resources (time, people, and money) is key to managing successful projects. Managing scope is a continuous activity that requires iterative or incremental development, which breaks project scope into smaller more manageable pieces.

Using requirement attributes, such as priority, effort, and risk, as the basis for negotiating the inclusion of a requirement is a particularly useful technique for managing scope. Focusing on the attributes rather than the requirements themselves helps desensitize negotiations that are otherwise contentious.

## 2.4. Project/Product Costing

A metric is some measurement we can make of a product or process in the overall development process. Metrics are split into two broad categories:

* Knowledge oriented metrics: these are oriented to tracking the process to evaluate, predict or monitor some part of the process.
* Achievement oriented metrics: these are often oriented to measuring some product aspect, often related to some overall measure of quality of the product.

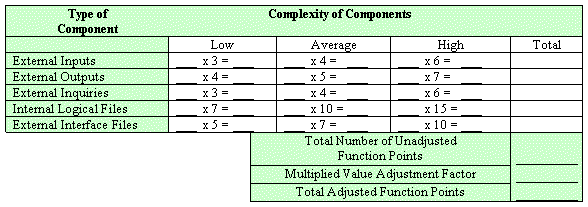
Most of the work in the cost estimation field has focused on algorithmic cost modeling. In this process costs are analyzed using mathematical formulas linking costs or inputs with metrics to produce an estimated output. The formulae used in a formal model arise from the analysis of historical data. The accuracy of the model can be improved by calibrating the model to your specific development environment, which basically involves adjusting the weightings of the metrics.

### 2.4.1. Project Cost Estimation By Function Point Analysis

Function-oriented software metrics use a measure of the functionality delivered by the application as a normalization value. Since ‘functionality’ cannot be measured directly, it must be derived indirectly using other direct measures. Function-oriented metrics were first proposed by Albrecht, who suggested a measure called the function point. Function points are derived using an empirical relationship based on countable (direct) measures of software’s information domain and assessments of software complexity.

Function Point Analysis can provide a mechanism to track and monitor scope creep. Function Point counts at the end of requirements; analysis, design, code, testing and implementation can be compared. The function point count at the end of requirements and/or designs can be compared to function points actually delivered. If the project has grown, there has been scope creep. The amount of growth is an indication of how well requirements were gathered by and/or communicated to the project team. If the amount of growth of projects declines over time it is a natural assumption that communication with the user has improved.

Function points are computed by completing the table shown in the figure below. Five information domain characteristics are determined and counts are provided in the appropriate table location.



Information domain values are defined in the following manner:

**Number of user inputs:** Each user input that provides distinct application-oriented data to the software is counted. Inputs should be distinguished from inquiries, which are counted separately.

**Number of user outputs:** Each user output that provides application-oriented information to the user is counted. In this context output refers to reports, screens, error messages, etc. Individual data items within a report are not counted separately.

**Number of user inquiries:** An inquiry is defined ass an on-line input that results in the generation of some immediate software response in the form of an on-line output. Each distinct inquiry is counted.

**Number of files:** Each logical master file (i.e. a logical grouping of data that may be one part of a large database or a separate file) is counted.

**Number of external interfaces:** All the machine-readable interfaces (e.g., data files on storage media) that are used to transmit information to another system are counted.

Once these data have been collected, a complexity value is associated with each count. Organizations that use function point methods develop criteria for determining whether a particular entry is simple, average, or complex. Nonetheless, the determination of complexity is somewhat subjective.

To compute function points (FP), the following relationship is used:

FP est. = Count Total \* [ 0.65 + 0.01 \* (Fi)]

Where count total is the sum of all FP entries obtained from above figure and (Fi) is value adjustment factor (VAF) is based on 14 general system characteristics (GSC's) that rate the general functionality of the application being counted. Each characteristic has associated descriptions that help determine the degrees of influence of the characteristics. The degrees of influence range on a scale of zero to five, from no influence to strong influence.

1. Data communications

2. Distributed data processing

3. Performance

4. Heavily used configuration

5. Transaction rate

6. On-Line data entry

7. End-user efficiency

8. On-Line update

9. Complex processing

10. Reusability

11. Installation ease

12. Operational ease

13. Multiple sites

14. Facilitate change

Finally, Total Project Cost and Total Project Effort are calculated given the average productivity parameter for the system.

The formulae are given as follows:

Cost / FP = labor rate / productivity parameter

Total Project Cost = FP est. \* (cost / FP)

Total Estimated Effort = FP est. / productivity parameter

### 2.4.2. Project Cost Estimation by using COCOMO’81 (Constructive Cost Model)

Boehm's COCOMO model is one of the mostly used models commercially. The first version of the model delivered in 1981 and COCOMO II is available now. COCOMO 81 is a model that allows one to estimate the cost, effort, and schedule when planning a new software development activity, according to software development practices that were commonly used in the 1970s through the 1980s. It exists in three forms, each one offering greater detail and accuracy the further along one is in the project planning and design process. Listed by increasing fidelity, these forms are called Basic, Intermediate, and Detailed COCOMO. However, only the Intermediate form has been implemented by USC in a calibrated software tool.

Three levels:

**Basic:** Is used mostly for rough, early estimates.

**Intermediate:** Is the most commonly used version, includes 15 different factors to account for the influence of various project attributes such as personnel capability, use of modern tools, hardware constraints, and so forth.

**Detailed:** Accounts for the influence of the different factors on individual project phases: design, coding/testing, and integration/testing. Detailed COCOMO is not used very often.

Each level includes three software development types:

1. **Organic:** Relatively small software teams develop familiar types of software in an in-house environment. Most of the personnel have experience working with related systems.
2. **Embedded:** The project may require new technology, unfamiliar algorithms, or an innovative new method
3. **Semi-detached:** Is an intermediate stage between organic and embedded types.

Basic COCOMO

Type Effort Schedule

Organic PM= 2.4 (KLOC)1.05 TD= 2.5(PM)0.38

Semi-Detached PM= 3.0 (KLOC)1.12 TD= 2.5(PM)0.35

Embedded PM= 2.4 (KLOC)1.20 TD= 2.5(PM)0.32

PM= person-month (effort)

KLOC= lines of code, in thousands

TD= number of months estimated for software development (duration)

Intermediate COCOMO

Type Effort

Organic PM= 2.4 (KLOC)1.05 x M

Semi-Detached PM= 3.0 (KLOC)1.12 x M

Embedded PM= 2.4 (KLOC)1.20 x M

PM= person-month

KLOC= lines of code, in thousands

M.- reflects 15 predictor variables, called cost drivers

The schedule is determined using the Basic COCOMO schedule equations.

People Required = Effort / Duration

### 2.4.3. Activity Based Costing

Activity-based costing (ABC) is a methodology that measures the cost and performance of activities, resources, and cost objects. Resources are assigned to activities, then activities are assigned to cost objects based on their use. Activity-based costing recognizes the causal relationships of cost drivers to activities.

Activity-based costing is about:

* Measuring business process performance, activity by activity.
* Estimating the cost of business process outputs based on the cost of the resources used in producing the product.
* Identifying opportunities to improve process efficiency and effectiveness.

Activity costs are used as the quantitative measurement. If activities have unusually high costs or vice versa, they become targets for re-engineering.

Activity-based management (ABM) is a broad discipline

Basic Cost Drivers:

For each activity state in an activity diagram, the basic cost drivers are:

* Resources: determine what business workers and business entities are participating, and how many instances of each. The allocation of a resource to a workflow implies a certain cost.
* Cost rate: each business worker or business entity instance may have a cost per time in use.
* Duration: an activity occurs for a certain time, therefore a resource can either be allocated for the duration of the activity, or for a fixed amount of time.
* Overhead: any fixed costs that the invocation of a workflow or an activity would incur.

## 2.5. Task Dependency Table

The following are the steps to develop a task dependency table:

1. Brainstorm for all of the tasks necessary to complete your project, or take the output from a Tree Diagram that you have already produced.

2. Write each task on a Post-It Note or job card.

3. Establish which task is the first one that must be carried out and place its card on the left hand side of a large work surface.

4. Determine whether there are any tasks that can be done at the same time, i.e. tasks that are not dependent on this first task finishing before they can be carried out. If there are, place their job cards vertically above or below the first job card.

5. Then decide which is the next task that must be carried out and place its card on the right of the first card. Determine whether there are any tasks that can be done at the same time as this task. If there are, place their job cards vertically above or below its job card.

6. Repeat the process until all of the job cards have been arranged in sequence and in parallel.

## 2.6. CPM - Critical Path Method

In 1957, DuPont developed a project management method designed to address the challenge of shutting down chemical plants for maintenance and then restarting the plants once the maintenance had been completed. Given the complexity of the process, they developed the Critical Path Method (CPM) for managing such projects.

CPM provides the following benefits:

* Provides a graphical view of the project.
* Predicts the time required to complete the project.
* Shows which activities are critical to maintaining the schedule and which are not.

CPM models the activities and events of a project as a network. Activities are depicted as nodes on the network and events that signify the beginning or ending of activities are depicted as arcs or lines between the nodes. The following is an example of a CPM network diagram:

Steps in CPM Project Planning

1. Specify the individual activities.

2. Determine the sequence of those activities.

3. Draw a network diagram.

4. Estimate the completion time for each activity.

5. Identify the critical path (longest path through the network)

6. Update the CPM diagram as the project progresses.

**1. Specify the Individual Activities**

From the work breakdown structure, a listing can be made of all the activities in the project. This listing can be used as the basis for adding sequence and duration information in later steps.

**2. Determine the Sequence of the Activities**

Some activities are dependent on the completion of others. A listing of the immediate predecessors of each activity is useful for constructing the CPM network diagram.

**3. Draw the Network Diagram**

Once the activities and their sequencing have been defined, the CPM diagram can be drawn. CPM originally was developed as an activity on node (AON) network, but some project planners prefer to specify the activities on the arcs.

**4. Estimate Activity Completion Time**

The time required to complete each activity can be estimated using past experience or the estimates of knowledgeable persons. CPM is a deterministic model that does not take into account variation in the completion time, so only one number is used for an activity's time estimate.

**5. Identify the Critical Path**

The critical path is the longest-duration path through the network. The significance of the critical path is that the activities that lie on it cannot be delayed without delaying the project. Because of its impact on the entire project, critical path analysis is an important aspect of project planning.

Determining the following six parameters for each activity which can identify the critical path:

**ES:** earliest start time: the earliest time at which the activity can start given that its precedent activities must be completed first.

ES (K)= max [EF(J) : J is an immediate predecessor of K]

**EF:** earliest finish time: equal to the earliest start time for the activity plus the time required to complete the activity.

EF (K)= ES (K) + Dur (K)

**LF:** latest finish time: the latest time at which the activity can be completed without delaying the project.

LF (K)= min [LS(J) : J is a successor of K]

**LS:** latest start time: equal to the latest finish time minus the time required to complete the activity.

LS (K)= LF(K) – Dur (K)

**TS:** Total Slack: the time that the completion of an activity can be delayed without delaying the end of the project

TS (K)= LS(K) – ES(K)

**FS:** Free Slack: the time that an activity can be delayed without delaying both the start of any succeeding activity and the end of the project.

FS (K)= min [ES(J) : J is successor of K] – EF(K)

The slack time for an activity is the time between its earliest and latest start time, or between its earliest and latest finish time. Slack is the amount of time that an activity can be delayed past its earliest start or earliest finish without delaying the project.

The critical path is the path through the project network in which none of the activities have slack, that is, the path for which ES=LS and EF=LF for all activities in the path. A delay in the critical path delays the project. Similarly, to accelerate the project it is necessary to reduce the total time required for the activities in the critical path.

**6. Update CPM Diagram**

As the project progresses, the actual task completion times will be known and the network diagram can be updated to include this information. A new critical path may emerge, and structural changes may be made in the network if project requirements change.

**Example:**

|  |  |  |
| --- | --- | --- |
| **Activity** | **Immediate Predecessor** | **Duration (Weeks)** |
| A | None | 5 |
| B | None | 3 |
| C | A | 8 |
| D | A, B | 7 |
| E | None | 7 |
| F | C, D, E | 4 |
| G | F | 5 |

Network Diagram for the above-mentioned activities

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Activity** | **Duration** | **ES** | **EF** | **LS** | **LF** | **TS** | **FS** |
| A | 5 | 0 | 5 | 0 | 5 | 0 | 0 |
| B | 3 | 0 | 3 | 3 | 6 | 3 | 2 |
| C | 8 | 5 | 13 | 5 | 13 | 0 | 0 |
| D | 7 | 5 | 12 | 6 | 13 | 1 | 1 |
| E | 7 | 0 | 7 | 6 | 13 | 6 | 6 |
| F | 4 | 13 | 17 | 13 | 17 | 0 | 0 |
| G | 5 | 17 | 22 | 17 | 13 | 0 | 0 |

The parameters and slacks are calculated as follows:

**The critical path is:**

A, C, F, G

## 2.7. Gantt chart

The Gantt chart enumerates the activities to be performed on the vertical axis and their corresponding duration on the horizontal axis. It is possible to schedule activities by either early start or late start logic. In the early start approach, each activity is initiated as early as possible without violating the precedence relations. In the late start approach, each activity is delayed as much as possible as long as the earliest finish time of the project is not compromised.

Based on the Work Breakdown Structure (WBS), a timeline or Gantt chart showing the allocation of time to the project phases or iterations should be developed. This Gantt chart would identify major milestones with their achievement criteria. It must contain duration estimation of all the necessary activities to be carried out during the project development along with the human resources responsible for the respective tasks. Activity dependencies are also required to be mentioned in it.

## 2.8. Introduction to Team member and their skill set

A brief but a concise introduction of the team members should be provided signifying their skill set. This skill set would especially be representative of the tasks and activities assigned to him.

## 2.9. Task and Member Assignment Table

A table should be formed which consists of a list of tasks and correspondingly allocation of members to that task. The basic aim of this table would be an indication of the amount of work the members would be performing.

Example for Task Durations and Dependencies, Activity Network Diagram, Gantt chart, and Allocation of People to Activities

|  |  |  |
| --- | --- | --- |
| Task | Duration (days) | Dependencies |
| T1 | 8 |  |
| T2 | 15 |  |
| T3 | 15 | T1(M1) |
| T4 | 10 |  |
| T5 | 10 | T2, T4(M2) |
| T6 | 5 | T1, T2 (M3) |
| T7 | 20 | T1 (M1) |
| T8 | 25 | T4 (M5) |
| T9 | 15 | T3, T6 (M4) |
| T10 | 15 | T5, T7 (M7) |
| T11 | 7 | T9 (M6) |
| T12 | 10 | T11 (M8) |

Consider the set of activities shown in figure. This table shows activities, their duration, and activity interdependencies. From figure, you can see that Task T3 is dependent on Task T1. This means that T1 must be completed before T3 starts. For example, T1 might be the preparation of a component design and T3, the implementation of that design. Before implementation starts, the design should be complete.



Task durations and dependencies

Given dependency and estimated duration of activities, and activity network that shows activity sequences may be generated. It shows which activities can be carried out in parallel and which must be executed in sequence because of a dependency on an earlier activity. Activities are represented as rectangles. Milestones and project deliverables are shown with rounded corners. Dates in this diagram show the start date of the activity and are written in British style where the day precedes the month. You should read the network from left to right and from top to bottom.

In the project management tool used to produce this chart, all activities must end in milestones. An activity may start when its preceding milestone (which may depend on several activities) has been reached. Therefore, in the third column in figure the corresponding milestone (e.g. M5) has been shown which is reached when the tasks in that column finish.

Before progress can be made form one milestone to another, all paths leading to it must be complete. For example, task T9, shown in the activity network below cannot be started until tasks T3 and T are finished. The arrival at milestone M4 shows that these tasks have been completed.

The minimum required to finish the project can be estimated be considering the longest part in the activity graph (the critical path). In this case, it is 11 weeks of elapsed time or 55 working days. In the activity network diagram the critical path is shown as a sequence of emboldened boxes. The overall schedule of the project depends on the critical path. Any slippage in the completion of any critical activity causes project delays.

Delays in activities, which do not lie on the critical path, however, need not cause an overall schedule slippage. So long as the delays do not extend these activities so much that the total time exceeds the critical path the project schedule will not be affected. For example, if T8 is delayed, it may not affect the final completion date of the project, as it does not lie on the critical path.

Managers also use activity networks when allocating project work. They can provide insights into activity dependencies, which are not intuitively obvious. It may be possible to modify the system design so that the critical path is shortened. The project schedule may be shortened because of the reduced amount of time spent waiting for activities to finish.



**Activity Bar Chart**

Figure with the gantt cart is an alternative way of representing project schedule information. It is a bar chart (sometimes called a Gantt chart, after its inventor) showing a project calendar and the start and finish dates of activities.

Some of the activities in the Gantt chart are followed by a shaded bar whose length is computed by the scheduling tool. This shows that there is some flexibility in the completion date of these activities. If an activity does not complete on time, the critical path will not be affected until the end of the period marked by the shaded bar. Activities, which lie on the critical path, have no margin of error and they can be identified because they have no associated shaded bar.

As well as considering schedules, project managers must also consider resource allocation and, in particular, the allocation of staff to project activities. Below is a figure showing the allocation of people to activities.

Project management support tools can also process the figure and a bar chart generated which shows the time periods where staff is employed on the project. Staff doesn’t have to be assigned to a project at all time. During intervening periods they may be on holiday, working on other projects, attending training courses or some other activity.

Large organizations usually employ a number of specialists who work on a project as required. This can cause scheduling problems. If one project is delayed while a specialist is working on it, this may have a knock-on effect on other projects. They may also be delayed because the specialist is not available.

**Allocation of People to Activities:**

|  |  |
| --- | --- |
| Task | Engineer |
| T1 | Jane |
| T2 | Anne |
| T3 | Jane |
| T4 | Fred |
| T5 | Mary |
| T6 | Anne |
| T7 | Jim |
| T8 | Fred |
| T9 | Jane |
| T10 | Anne |
| T11 | Fred |
| T12 | Fred |

**Staff Allocation:**

****

## 2.10. Tools and Technology with reasoning

The application tools, which are to be used on front and back end of the system to be developed, should be listed. The reasons for these tools should also be described.

Identify what the needs for tool support are, and what the constraints are, by looking at the following:

* The development process. What tool support is required to effectively work? For example, if the organization decide to employ an iterative development process, it is necessary to automate the tests, since you will be testing several times during the project.
* Host (or development) platform(s).
* Target platform(s).
* The programming language(s) to be used.
* Existing tools. Evaluate any existing and proven tools and decide whether they can continue to be used.
* The distribution of the development organization. Is the organization physically distributed? Development tools generally support a physically distributed organization differently.
* The size of the development effort. Tools support large organizations more or less well.
* Budget and time constraints

## 2.11. Vision Document

The Vision defines the stockholder’s view of the product to be developed, specified in terms of the stockholder’s key needs and features. Containing an outline of the envisioned core requirements, it provides the contractual basis for the more detailed technical requirements.

A Vision Document is the starting point for most software projects. It is the primary deliverable and is therefore the first document produced in the planning process. The main purpose of this document is to move the project forward into detailed project planning and ultimately into development.

The Vision Document is designed to make sure that key decision makers on both sides have a clear, shared vision of the objectives and scope of the project. It identifies alternatives and risks associated with the project. Finally, it presents a budget for the detailed planning phase for the stakeholders to approve.

The Vision document provides a high-level for the more detailed technical requirements. There can also be a formal requirements specification. The Vision captures very high-level requirements and design constraints to give the reader an understanding of the system to be developed. It provides input to the project-approval process and is, therefore, intimately related to the Business Case. It communicates the fundamental "whys and what's" related to the project and is a gauge against which all future decisions should be validated.

A project vision is meant to be changeable as the understanding of requirements, architecture, plans, and technology evolves. However, it should be changing slowly and normally throughout the earlier portion of the lifecycle.

It is important to express the vision in terms of its use cases and primary scenarios as these are developed, so that you can see how the vision is realized by the use cases. The use cases also provide an effective basis for evolving a test case suite.

Another name used for this document is the Product Requirement Document. There are certain checkpoints that help to verify that the vision document is fulfilled.

Checkpoints:

* Have you fully explored what the "problem behind the problem" is?
* Is the problem statement correctly formulated?
* Is the list of stakeholders complete and correct?
* Does everyone agree on the definition of the system boundaries?
* If system boundaries have been expressed using actors, have all actors been defined and correctly described?
* Have you sufficiently explored constraints to be put on the system?
* Have you covered all kinds of constraints - for example political, economic, and environmental?
* Have all key features of the system been identified and defined?
* Will the features solve the problems that are identified?
* Are the features consistent with constraints that are identified?

## 2.12. Risk List

The possibility of suffering harm or loss in terms of danger is called risk. Regarding the importance of risks a list is to be maintained. Risk list is a sorted list of known, open risks to the project, sorted in decreasing order of importance, associated with specific mitigation or contingency actions.

Purpose

The Risk List is designed to capture the perceived risks to the success of the project. It identifies, in decreasing order of priority, the events that could lead to a significant negative outcome. It serves as a focal point for project activities and is the basis around which iterations are organized

The Risk List is maintained throughout the project. It is created early in the Inception phase, and is continually updated as new risks are uncovered and existing risks are mitigated or retired. At a minimum, it is revisited at the end of each iteration, as the iteration is assessed.

## 2.13. Product Features/ Product Decomposition

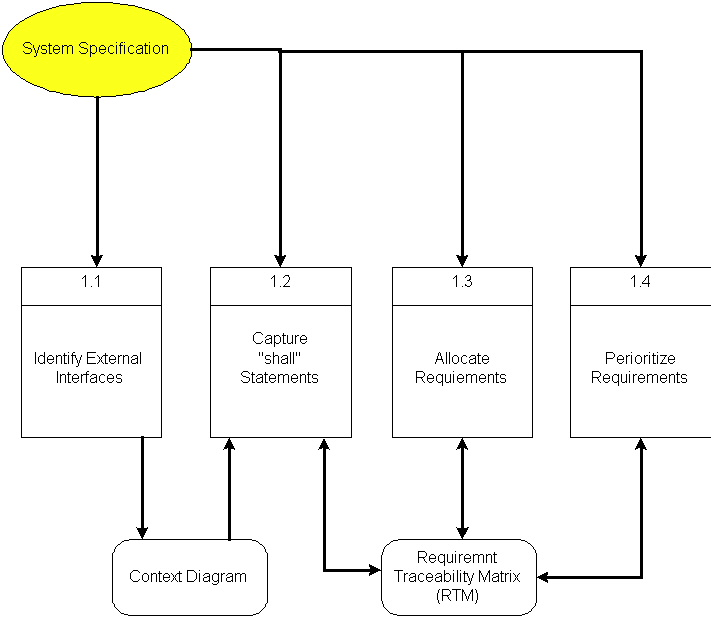
Functional requirements capture the intended behavior of the system. This behavior may be expressed as services, tasks or functions the system is required to perform.

# Chapter 3: Second Deliverable For Object Oriented Approach

## 3.1 Introduction:

Requirements engineering process provides the appropriate mechanism for understanding what the customer wants, analyzing need, assessing feasibility, negotiating a reasonable solution, specifying the solution unambiguously, validating the specification and managing the requirements as they are transformed into an operational system. The task of capturing, structuring, and accurately representing the user's requirements so that they can be correctly embodied in systems which meet those requirements (i.e. are of good quality).

* Requirements elicitation
* Requirements analysis and negotiation
* Requirements specification
* System modeling
* Requirements validation
* Requirements management



Here, requirements specification is to be discussed. Requirements specification would lead to the following four steps:

* Identify external interfaces
* Development of context diagram
* Capture “shall statements
* Allocate requirements
* Prioritize requirements
* Development of requirements traceability matrix

### 3.1.1 Systems Specifications

The following are the clauses that must be included while describing the system specifications.

**Introduction**

This clause should contain brief “Introduction” of the system under discussion domain knowledge. It can also contain company, its location, its historical background and its current status in the market. The most important part of this clause is to give an overview of the major business areas of the company. This overview must be very brief so that one can get a bird’s eye view of the organization under study.

**Existing System**

This clause must be focusing on providing a comprehensive detail of main business areas of the organizations that we have just mentioned in the previous clause. But here the discussion should be more elaborative.

**Organizational Chart**

Organizational chart will be very much supportive to get a better overview of the organization’s business areas and their decomposition into different departments.

**Scope of the System**

The Scope may include the boundaries of the system under study. To what domain you want to restrict your project must be clearly mentioned in this clause.

**Summary of Requirements: (Initial Requirements)**

An abstract is necessary at this stage to give an understanding of the initial requirements of the system. This will show what high level requirements the proposed system must address. This abstract will act as a foundation for the future analysis of the system.

### 3.1.2. Identifying External Entities

The identification of the external entities will be based on the information contained in your Abstract. This identification is done after two phases. We will map the “Green wood” case study to make things more comprehensible.

The Identification of External Entities is done in two phases.

**a. Over Specify Entities from Abstract:**

On the basis of the Abstract, one might identify the entities from the problem.

**b. Perform Refinement:**

After over specifying the entities, you have to refine them on the basis of your business logic. For example, in this example we found the following entities more related to our business logic;

### 3.1.3. Context Level Data Flow Diagram:

Context level data flow diagram contains only one process, representing the entire system. The process is given the number zero and all external entities are shown on the context diagram as well as major data flow to and from them. The diagram does not contain any data stores.

### 3.1.4. Capture "shall" Statements:

Identify “shall” statements, as they would be all functional requirements.

### 3.1.5. Allocate Requirements:

Allocate the requirements in the use cases.

### 3.1.6. Prioritize Requirements:

Requirements must be prioritized as this will help achieve tasks easily. Rank them as “highest, medium, and lowest”.

### 3.1.7. Requirements Trace-ability Matrix:

The requirements trace-ability matrix is a table used to trace project life cycle activities and work products to the project requirements. The matrix establishes a thread that traces requirements from identification through implementation.

## 3.2. Example:

Here is an example to explain all the above. We are taking the system of Green Wood Company.

### 3.2.1. Introduction

Green Wood (GW) is a multinational company, which deals in manufacturing, delivery and selling of sports goods and sports ware throughput the world. GW deals in almost all types of support goods and has its manufacturing set-up in Sialkot, Pakistan. They have their own products selling outlets and showrooms throughout the world. They also supply their goods to other dealers on wholesale ordering basis. Currently GW is managing their operations manually. GW management has decided to completely automate the whole business processes of the company. Also in order to increase their sales, GW wants to have fully automated system, which can support online 24x7 electronic buying and selling.

### 3.2.2. Existing System

**Business Organization**

GW deals in following three main business areas:

* Sport goods manufacturing
* Sport goods ordering and supply
* Consumer Outlets & Showrooms

Following departments/offices facilitates above mentioned business services:

Sport Goods Manufacturing Department

Deals in manufacturing of sport goods.

GW Supplier Office

It deals in supply of sport goods to their own selling outlets or to other dealers. It also processes orders from the dealers. Following are some business processes, which are handled in this department.

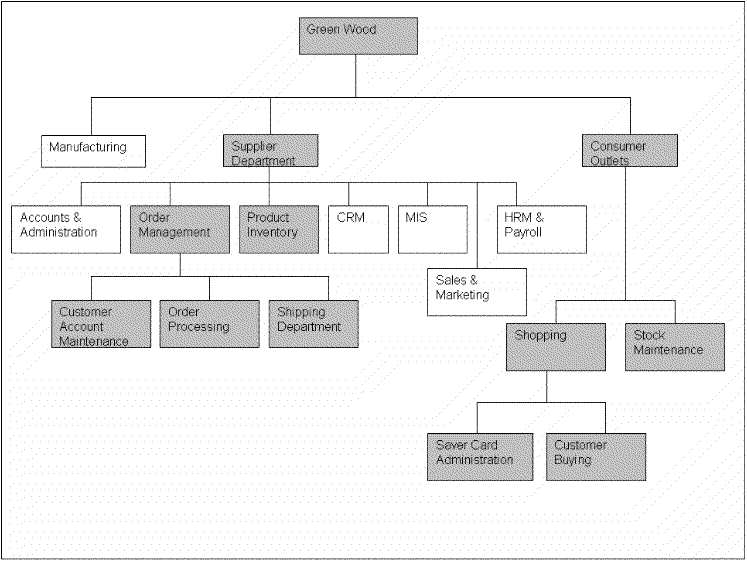
* Order Management
* Customer Account Maintenance
* Order Processing
* Shipping Department
* Product Inventory
* Accounts & Administration
* CRM
* MIS
* HRM & Pay Roll
* Sales & Marketing

GW Consumer Outlets & Showrooms

They directly deals with buying and selling of goods to customers

* Shopping Centre
* Stock Maintenance

**Business Organization Chart**



### 3.2.3. Scope of the System

The GW System is divided in to three phases.

#### Phase I

Phase I includes following business areas:

* Customer Account Maintenance
* Order Processing
* Product Inventory

#### Phase II

Phase II involves complete automation of the Supplier Department. Phase II includes following business areas:

* Accounts and Administration
* CRM
* MIS
* HRM and Payroll
* Sales and Marketing

#### Phase III

Phase III covers a complete solution for Green Wood. Phase III includes remaining business areas which are not developed in previous phases.

 This document scope is limited to Phase I only.

### 3.2.4. Summary of Requirements:(Initial Requirements)

The purposed system must fulfill following requirements as follow:

#### 3.2.4.1. Supplier Department Requirements

**Order Management**

1. Only registered customer could place order for goods. So a customer must be able to register himself to the system by requesting for registration. There should have to be two types of registration process, normal and privileged. Customer should provide his personal, organizational, authorizer and payment details in the registration request process. All the requests are to be viewed by the customer account administrator (CA). CA could accept, reject and temporarily waive the requests on the basis of credentials provided. If admin accept the registration request, a login information (Password, Id & role) should be assigned and mailed to the corresponding customer. Similarly customer could also request for the updating of his record. He could request for different types of updating e.g. updating of his personal/shipping details, or upgrading of his status from registered to privileged customer, or updating of his payment methodology. Customer could also view his details for verification purposes and similarly CA could search any customer detail and could also view the whole list of currently registered customers.

2. Both registered and privileged customers could order for goods. Customer places an order by providing his ID and other order related details A complete order must contain personal details of the customer, shipping information, product list along with product quantity and payment details. Customer could make payment either through cash or through a credit card. Accordingly invoice should be generated, and user should be given the option to finally place the order and in the end confirmation receipt must be given to the customer. Invoice contains the list of complete product along with their pricing details. It also contains discounts, sales tax and total pricing details. User could also view the status of their orders by providing the Order Number. Privileged customers could also place the request for the updating of their orders if the orders are not shipped. They could place request for the updating of shipping address and product quantity only. Similarly the privileged customer could also place the request for the cancellation of the order. But all these updating and cancellation requests are to be viewed by the Order Administrator in order to accept, reject, or waive them.

3.Action List mechanism should be adopted for better notification/messaging services, business interaction and control. An action event should be generated for a corresponding administrator when a request is placed for updating of orders or customer details etc. These actions could be generated by the Order Operator or through the updating process. Similarly on the other hand corresponding administrator could view his Action List containing different actions, and correspondingly process these pending actions. Similarly when the action processing is completed or if the action is just a notification message then administrator could delete these actions from the action list. Actions List configuration should be done by System Admin, who could add new action events and delete any current event from the system.

4. Shipping Department ships the corresponding orders.

#### 3.4.2.2. Product Inventory

Deals with addition, searching, updating of products and their stocks. Whenever a product stock arrives, the Inventory Administrator updates the stocks of the products. He could add new product in the inventory. He could also view, search and modify the product details. The Admin could view the whole product list and their product summaries.

#### 3.4.2.3. Consumer Dealing Department Requirements

Deals with front office customer dealing related to goods sales and marketing.

Shopping Centre

* Deals with customer registration and saver card administration
* Also deals with customer buying and returning of goods

#### 3.4.2.3. Product Stock Maintenance

Deals with addition, searching, updating of products and their stocks.

### 3.2.5. Identifying External Entities:

The identification of the external entities will be based on the information contained in your Abstract. This identification is done after two phases. We will map the “Green wood” case study to make things more comprehensible.

The Identification of External Interfaces is done in two phases.

**Over Specify Entities from Abstract:**

On the basis of the Abstract, one might identify the following entities from the Green Wood case study.

* Customer
* Order
* Order Product
* Shipment
* Invoice
* Product
* Payment
* Account
* Credit Card
* Cheque
* Request

**Perform Refinement:**

After over specifying the entities, you have to refine them on the basis of your Business Logic. For example, in this example we found the following entities more related to our Business Logic;

* Customer
* Inventory
* Shipment
* Account

### 3.2.6. Capture "shall" Statements:

|  |  |
| --- | --- |
| **Para #** | **Initial Requirements** |
| 1.0 | A customer “shall” place order for goods |
| 1.0 | A customer “shall” register himself to the system |
| 1.0 | The system “shall” provide two types of registration process, normal and privileged |
| 1.0 | CA “shall” accept, reject and temporarily waive the requests on the basis of credentials provided. |
| 1.0 | A customer “shall” login to the system and can change his password |
| 1.0 | System “shall” update the customers Request |
| 1.0 | System “shall” process different types of updating e.g. updating of his personal/shipping details, or upgrading of his status from registered to privileged customer, or updating of his payment methodology |
| 1.0 | A customer “shall” view his details for verification purposes |
| 1.0 | CA “shall”accept, reject and temporarily waive the requests on the basis of credentials provided. |
| 1.0 | System “shall” search any customer details |
| 2.0 | Both registered and privileged customers “will”order for goods. |
| 2.0 | Customer “shall” make payment; either through cash or through a credit card |
| 2.0 | System “shall” generate invoice, confirmation receipt and finally will place order |
| 2.0 | User “shall” view the status of their orders by providing the Order Number |
| 2.0 | Privileged customers “shall”place the request for the updating of their orders if the orders are not shipped. |
| 2.0 | Privileged customer “shall” place the request for the cancellation of the order. But all these updating and cancellation requests are to be viewed by the Order Administrator in order to accept, reject, or waive them. |
| 3.0 | An action event "shall" be generated for a corresponding administrator when a request is placed for updating of orders or customer details etc |
| 3.0 | Corresponding administrator "shall" view his Action List containing different actions, and correspondingly process these pending actions |
| 3.0 | When the action processing is completed or if the action is just a notification message then administrator "shall" delete these actions from the action list |

### 3.2.7. Allocate Requirements:

|  |  |  |
| --- | --- | --- |
| **Para #** | **Initial Requirements** | **Use Case Name** |
| 1.0 | A customer “will” place order for goods | UC\_Place\_Order |
| 1.0 | A customer “shall” register himself to the system | UC\_Registration\_Request |
| 1.0 | The system “shall” provide two types of registration process, normal and privileged | UC\_Place\_Order\_Request |
| 1.0 | CA “shall”accept, reject and temporarily waive the requests on the basis of credentials provided. | UC\_Process\_Customer\_Request |
| 1.0 | A customer “shall” login to the system and can change his password | UC\_Login |
| 1.0 | System “shall” update the customers Request | UC\_Update\_Request |
| 1.0 | System “shall” process different types of updating e.g. updating of his personal/shipping details, or upgrading of his status from registered to privileged customer, or updating of his payment methodology | UC\_Change\_Status |
| 1.0 | A customer “shall” view his details for verification purposes | UC\_View\_Customer\_Details |
| 1.0 | System “shall” search any customer details | UC\_Search\_Customer |
| 1.0 | CA “shall”accept, reject and temporarily waive the requests on the basis of credentials provided. | UC\_Accept\_Customer\_Request  UC\_Reject\_Customer\_Request  UC\_View\_Customer\_Request |
|  |  |  |
| 2.0 | Both registered and privileged customers “will”order for goods. | UC\_Place\_Order\_Privleged |
| 2.0 | Customer “will” make payment; either through cash or through a credit card | UC\_Pay\_For\_Order |
| 2.0 | System “will” generate invoice, confirmation receipt and finally will place order | UC\_Invoice\_Generation, |
| 2.0 | User “shall” view the status of their orders by providing the Order Number | UC\_Serach\_Orders |
| 2.0 | Privileged customers “shall”place the request for the updating of their orders if the orders are not shipped. | UC\_Update\_Request |
| 2.0 | Privileged customer “shall” place the request for the cancellation of the order. But all these updating and cancellation requests are to be viewed by the Order Administrator in order to accept, reject, or waive them. | UC\_Change\_Payment\_Details,   UC\_Change\_Status,  UC\_Change\_Personal\_Details |
| 3.0 | The System “shall” generate an action event for a corresponding administrator when a request is placed for updating of orders or customer details etc | UC\_Create\_Action, |
| 3.0 | Corresponding administrator “shall ” view his Action List containing different actions, and correspondingly process these pending actions | UC\_View\_Action, |

### 

### 3.2.8. Priorities Requirements:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Para # | **Rank** | **Initial Requirements** | **Use Case ID** | **Use Case Name** |
| 1.0 | Highest | A customer “will” place order for goods | UC\_1 | UC\_PlaceOrder |
| 1.0 | Highest | A customer “shall” register himself to the system | UC\_2 | UC\_Registration\_Request |
| 2.0 | Highest | Customer “will” make payment either through cash or through a credit card | UC\_3 | UC\_Pay\_For\_Order |
| 2.0 | Highest | System “will” generate invoice, confirmation receipt and finally will place order | UC\_4 | UC\_Invoice\_Generation, |
| 2.0 | Medium | Both registered and privileged customers “will”order for goods. | UC\_5 | UC\_Place\_Order\_Privleged |
| 1.0 | Medium | The system “shall” provide two types of registration process, normal and privileged | UC\_6 | UC\_Place\_Order\_Request |
| 3.0 | Medium | The System “shall” generate an action event for a corresponding administrator when a request is placed for updating of orders or customer details etc | UC\_7 | UC\_Create\_Action |
| 1.0 | Medium | CA “shall”accept, reject and temporarily waive the requests on the basis of credentials provided. | UC\_8  UC\_9  UC\_10 | UC\_Accept\_Customer\_Request  UC\_Reject\_Customer\_Request  UC\_View\_Customer\_Request |
| 1.0 | Medium | System “shall” update the customers Request | UC\_11 | UC\_Update\_Request |
| 1.0 | Medium | System “shall” process different types of updating e.g. updating of his personal/shipping details, or upgrading of his status from registered to privileged customer, or updating of his payment methodology | UC\_12  UC\_13  UC\_14 | UC\_Change\_Payment\_Details,   UC\_Change\_Status,  UC\_Change\_Personal\_Details |
| 1.0 | Medium | A customer “shall” view his details for verification purposes | UC\_15 | UC\_View\_Customer\_Details |
| 1.0 | Medium | System “shall” search any customer details | UC\_16 | UC\_Search\_Customer |
| 2.0 | Medium | User “shall” view the status of their orders by providing the Order Number | UC\_17 | UC\_Serach\_Orders |
| 2.0 | Medium | Privileged customers “shall”place the request for the updating of their orders if the orders are not shipped. | UC\_18 | UC\_Update\_Request |
| 2.0 | Medium | Privileged customer “shall” place the request for the cancellation of the order. But all these updating and cancellation requests are to be viewed by the Order Administrator in order to accept, reject, or waive them. | UC\_19  UC\_20  UC\_21 | UC\_View\_All\_Orders  UC\_Manage\_Order |
| 1.0 | Lowest | A customer “shall” login to the system and can change his password | UC\_22  UC\_23 | UC\_Login, |
| 3.0 | Lowest | Corresponding administrator “shall ” view his Action List containing different actions, and correspondingly process these pending actions | UC\_24 | UC\_View\_Action, |
| 3.0 | Lowest | When the action processing is completed or if the action is just a notification message then administrator “shall” delete these actions from the action list | UC\_25 | UC\_Delete\_Action |

## 

### 3.2.9. Requirements Traceability Matrix:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr#** | **Para #** | **System Specification Text** | **Build** | **Use Case Name** | **Category** |
| 1 | 1.0 | A customer “will” place order for goods | B1 | UC\_Place\_Order | Business |
| 2 | 1.0 | A customer “shall” register himself to the system | B1 | UC\_Registration\_Request | Business |
| 3 | 1.0 | The system “shall” provide two types of registration process, normal and privileged | B1 | UC\_PlaceOrderRequest,  UC\_PlaceCustomerRequest | Business |
| 4 | 1.0 | CA “shall”accept, reject and temporarily waive the requests on the basis of credentials provided. | B1 | UC\_Accept\_Customer\_Request  UC\_Reject\_Customer\_Request  UC\_View\_Customer\_Request | Business |
| 5 | 1.0 | A customer “shall” login to the system and can change his password | B1 | UC\_Login, | Business |
| 6 | 1.0 | System “shall” update the customers Request | B1 | UC\_Update\_Request | Business |
| 7 | 1.0 | System “shall” process different types of updating e.g. updating of his personal/shipping details, or upgrading of his status from registered to privileged customer, or updating of his payment methodology | B1 | UC\_Change\_Payment\_Details,   UC\_Change\_Status,  UC\_Change\_Personal\_Details | Business |
| 8 | 1.0 | A customer “shall” view his details for verification purposes | B1 | UC\_View\_Customer\_Details | Business |
| 9 | 1.0 | System “shall” search any customer details | B1 | UC\_SearchCustomer | Business |
| 10 | 2.0 | Both registered and privileged customers “will”order for goods. | B1 | UC\_Place\_Order\_Privellged | Business |
| 11 | 2.0 | Customer “will” make payment; either through cash or through a credit card | B1 | UC\_Pay\_For\_Order | Business |
| 12 | 2.0 | System “will” generate invoice, confirmation receipt and finally will place order | B1 | UC\_Invoice\_Generation | Business |

### 3.2.10. High Level Usecase Diagram:

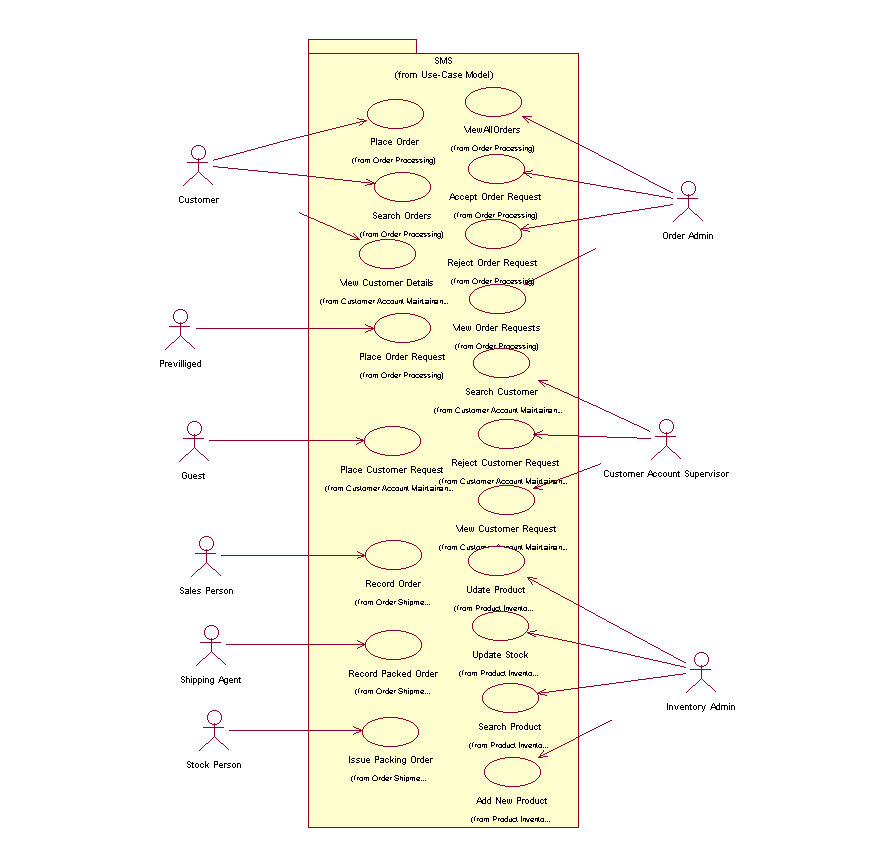
A use case scenario is a visual description, typically written in structured English or point form, of a potential business situation that a system may or may not be able to handle.

A use case defines a goal-oriented set of interactions between external actors and the system under consideration.

A use case is initiated by a user with a particular goal in mind, and completes successfully when that goal is satisfied. It describes the sequence of interactions between actors and the system necessary to deliver the service that satisfies the goal. It also includes possible variants of this sequence, e.g., alternative sequences that may also satisfy the goal, as well as sequences that may lead to failure to complete the service because of exceptional behavior, error handling, etc. The system is treated as a “black box”, and the interactions with system, including system responses, are as perceived from outside the system.

Thus, use cases capture who (actor) does what (interaction) with the system, for what purpose (goal), without dealing with system internals. A complete set of use cases specifies all the different ways to use the system, and therefore defines all behavior required of the system, bounding the scope of the system.

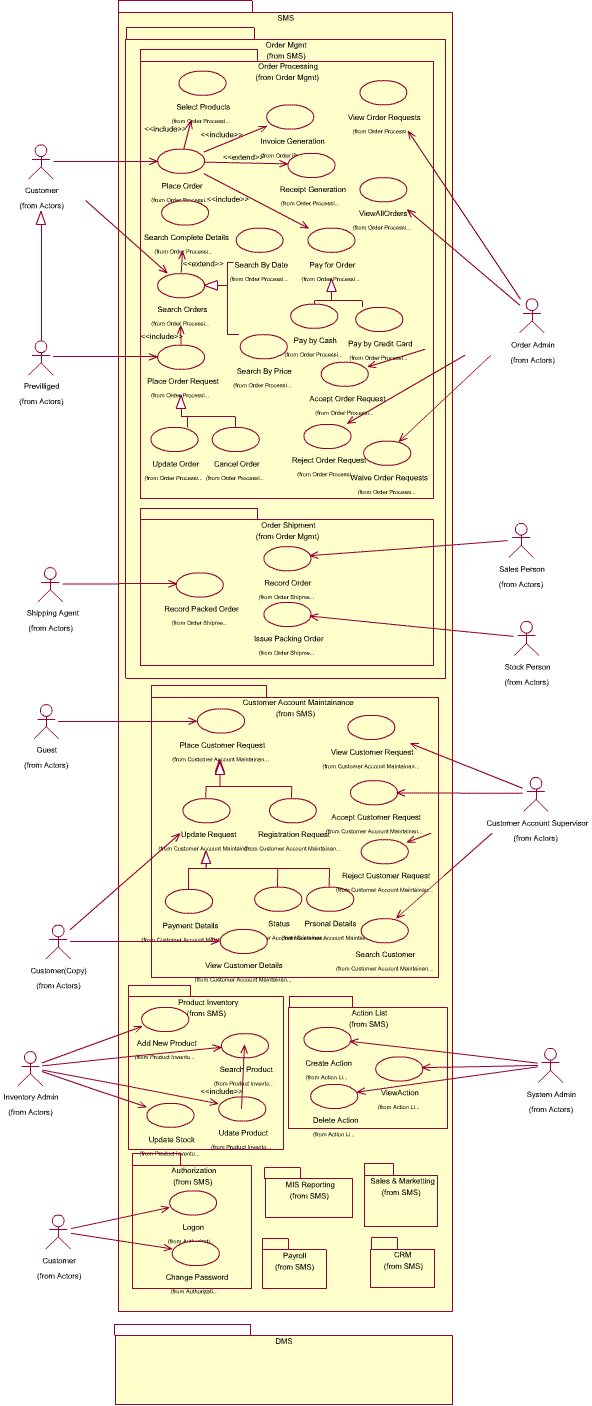
Generally, use case steps are written in an easy-to-understand structured narrative using the vocabulary of the domain. This is engaging for users who can easily follow and validate the use cases, and the accessibility encourages users to be actively involved in defining the requirements.

**Example:**

### 3.2.11. Analysis Level Usecase Diagram:

Analysis level usecase diagram is actually the explanation of high level usecas diagram. In this diagram high level usecases are expanded in a way that exhibit how high level usecases will reach to their functionality. Two types of relationships are used in this diagram. Which are:

* Extend
* Include

****

### 3.2.12. Usecase Description

While technically not part of UML, use case documents are closely related to UML use cases. A use case document is text that captures the detailed functionality of a use case. Such documents typically contain the following parts:

#### Brief description

Used to describe the overall intent of the use case. Typically, the brief description is only a few paragraphs, but it can be longer or shorter as needed. It describes what is considered the happy path—the functionality that occurs when the use case executes without errors. It can include critical variations on the happy path, if needed.

#### Preconditions

Conditionals that must be true before the use case can begin to execute. Note that this means the author of the use case document does not need to check these conditions during the basic flow, as they must be true for the basic flow to begin.

#### Basic flow

Used to capture the normal flow of execution through the use case. The basic flow is often represented as a numbered list that describes the interaction between an actor and the system. Decision points in the basic flow branch off to alternate flows. Use case extension points and inclusions are typically documented in the basic flow.

#### Alternate flows

Used to capture variations to the basic flows, such as user decisions or error conditions. There are typically multiple alternate flows in a single use case. Some alternate flows rejoin the basic flow at a specified point, while others terminate the use case.

#### Post conditions

Conditions that must be true for the use case to completed. Post conditions are typically used by the testers to verify that the realization of the use case is implemented correctly.

# Chapter 4: Third Deliverable For Object Oriented Approach

## 4.1. Introduction:

Third deliverable is all about the software design. In the previous deliverable, analysis of the system is completed. So we understand the current situation of the problem domain. Now we are ready to strive for a solution for the problem domain by using object-oriented approach. Following artifacts must be included in the 3rd deliverable.

1. Domain Model
2. System Sequence Diagram
3. Sequence Diagram
4. Collaboration Diagram
5. Operation Contracts
6. Design Class Diagram
7. State Transition Diagram
8. Data Model

Now we discuss these artifacts one by one as follows:

## 4.2. Domain Model

Domain models represent the set of requirements that are common to systems within a product line. There may be many domains, or areas of expertise, represented in a single product line and a single domain may span multiple product lines. The requirements represented in a domain model include:

* Definition of scope for the domain
* Information or objects
* Features or use cases, including factors that lead to variation
* Operational/behavioral characteristics

A product line definition will describe the domains necessary to build systems in the product line.

What is domain modeling?

According to Rational Unified Process,® or RUP,® a domain model is a business object model that focuses on "product, deliverables, or events that are important to the business domain." A domain model is an "incomplete" business model, in that it omits individual worker responsibilities. The point of domain modeling is to provide "the big picture" of the interrelationships among business entities in a complex organization. The domain model typically shows the major business entities, and the relationships among the entities. A model that typically does not include the responsibilities people carry is often referred to as a domain model.

It also provides a high-level description of the data that each entity provides. Domain modeling plays a central role in understanding the current environment and planning for the future.

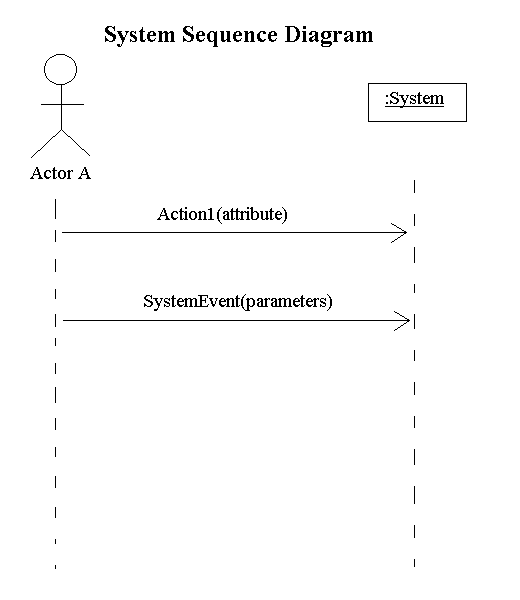
* The typical steps involved in domain modeling are:
* Illustrate meaningful conceptual classes in a real-world problem domain
* Identify conceptual classes or domain objects
* Show associations between them
* Indicate their attributes when appropriate
* Purposely incomplete

## 4.3. System Sequence Diagram

The UML system sequence diagram (SSD) illustrates events sequentially input from an external source to the system. The SSD will define the system events and operations. System sequence diagrams are a timeline drawing of an expanded use case. Events are related by time with the top events occurring first. System events are the important items. These are events that cause a system response.

Use case text may be placed on the left side of the system sequence diagram if desired. If this is done it is best if the use case information lines up with the events in the system sequence diagram.

There may be more than one actor to the system. An actor may be an external automated system that the system may communicate with. Automated actors or robots are shown as actors with a line horizontally through the head.



## 4.4. Sequence Diagram

A Sequence diagram depicts the sequence of actions that occur in a system. The invocation of methods in each object, and the order in which the invocation occurs is captured in a Sequence diagram. This makes the Sequence diagram a very useful tool to easily represent the dynamic behavior of a system.

A Sequence diagram is two-dimensional in nature. On the horizontal axis, it shows the life of the object that it represents, while on the vertical axis, it shows the sequence of the creation or invocation of these objects.

Because it uses class name and object name references, the Sequence diagram is very useful in elaborating and detailing the dynamic design and the sequence and origin of invocation of objects. Hence, the Sequence diagram is one of the most widely used dynamic diagrams in UML.

### 4.4.1. Defining a Sequence diagram

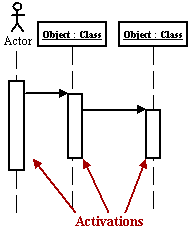
A sequence diagram is made up of objects and messages. Objects are represented exactly how they have been represented in all UML diagrams—as rectangles with the underlined class name within the rectangle.

Sequence diagrams describe interactions among classes in terms of an exchange of messages over time.

### 4.4.2. Basic Sequence Diagram Symbols and Notations

**Class roles**

Class roles describe the way an object will behave in context. Use the UML object symbol to illustrate class roles, but don't list object attributes.

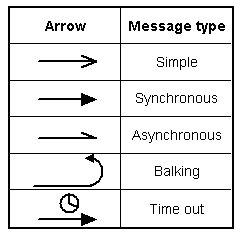


Activation

Activation boxes represent the time an object needs to complete a task.

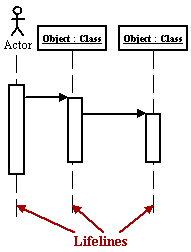
Messages

Messages are arrows that represent communication between objects. Use half-arrowed lines to represent asynchronous messages. Asynchronous messages are sent from an object that will not wait for a response from the receiver before continuing its tasks.



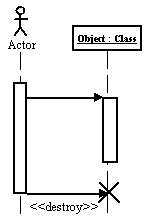
Lifelines

Lifelines are vertical dashed lines that indicate the object's presence over time.



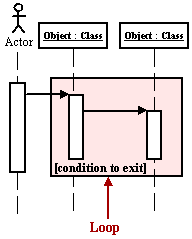
Destroying Objects

Objects can be terminated early using an arrow labeled "< < destroy > >" .



Loops

A repetition or loop within a sequence diagram is depicted as a rectangle. Place the condition for exiting the loop at the bottom left corner in square brackets [ ].



Objects

An object is shown as a vertical dashed line called the "lifeline". The lifeline represents the existence of the object at a particular time. An object symbol is drawn at the head of the lifeline, and shows the name of the object and its class underlined, and separated by a colon:

objectname : classname

You can use objects in sequence diagrams in the following ways:

* A lifeline can represent an object or its class. Thus, you can use a lifeline to model both class and object behavior. Usually, however, a lifeline represents all the objects of a certain class.
* An object's class can be unspecified. Normally you create a sequence diagram with objects first, and specify their classes later.
* The objects can be unnamed, but you should name them if you want to discriminate different objects of the same class.
* Several lifelines in the same diagram can represent different objects of the same class; but, as stated previously, the objects should be named that so you can discriminate between the two objects.
* A lifeline that represents a class can exist in parallel with lifelines that represent objects of that class. The object name of the lifeline that represents the class can be set to the name of the class.

Actors

Normally an actor instance is represented by the first (left-most) lifeline in the sequence diagram, as the invoker of the interaction. If you have several actor instances in the same diagram, try keeping them either at the left-most, or the right-most lifelines.

Messages

A message is a communication between objects that conveys information with the expectation that activity will ensue; in sequence diagrams, a message is shown as a horizontal solid arrow from the lifeline of one object to the lifeline of another object. In the case of a message from an object to itself, the arrow may start and finish on the same lifeline. The arrow is labeled with the name of the message, and its parameters. The arrow may also be labeled with a sequence number to show the sequence of the message in the overall interaction. Sequence numbers are often omitted in sequence diagrams, in which the physical location of the arrow shows the relative sequence.

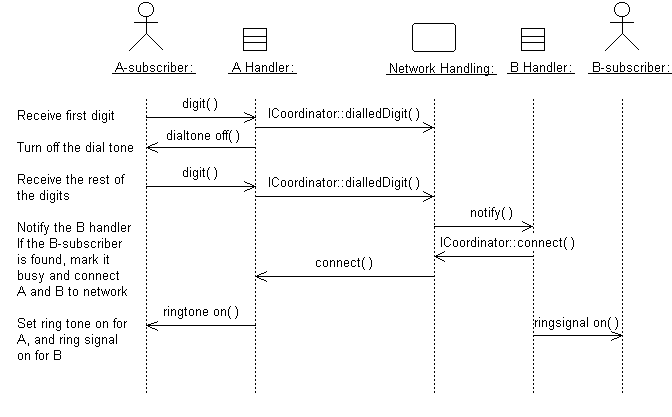
A message can be unassigned, meaning that its name is a temporary string that describes the overall meaning of the message and is not the name of an operation of the receiving object. You can later assign the message by specifying the operation of the message's destination object. The specified operation will then replace the name of the message.

Scripts

Scripts describe the flow of events textually in a sequence diagram.

You should position the scripts to the left of the lifelines so that you can read the complete flow from top to bottom (see figure above). You can attach scripts to a certain message, thus ensuring that the script moves with the message.

### 4.4.3. Example

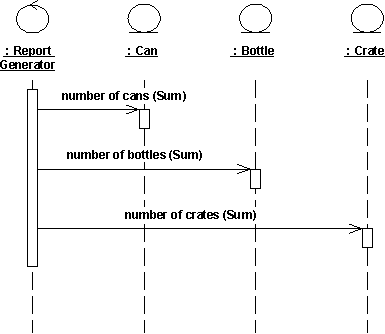
A sequence diagram that describes part of the flow of events of the use case Place Local Call in a simple Phone Switch.

### 4.4.4. Distributing Control Flow in Sequence Diagrams

**Centralized control** of a flow of events or part of the flow of events means that a few objects steer the flow by sending messages to, and receiving messages from other objects. These controlling objects decide the order in which other objects will be activated in the use case. Interaction among the rest of the objects is very minor or does not exist.

Example

In the Recycling-Machine System, the use case Print Daily Report keeps track of - among other things - the number and type of returned objects, and writes the tally on a receipt. The Report Generator control object decides the order in which the sums will be extracted and written.



The behavior structure of the use case Print Daily Report is centralized in the Report Generator control object.

This is an example of centralized behavior. The control structure is centralized primarily because the different sub-event phases of the flow of events are not dependent on each other. The main advantage of this approach is that each object does not have to keep track of the next object's tally. To change the order of the sub-event phases, you merely make the change in the control object. You can also easily add still another sub-event phase if, for example, a new type of return item is included. Another advantage to this structure is that you can easily reuse the various sub-event phases in other use cases because the order of behavior is not built into the objects.

Decentralized control arises when the participating objects communicate directly with one another, not through one or more controlling objects.

Example

In the use case Send Letter someone mails a letter to another country through a post office. The letter is first sent to the country of the addressee. In the country, the letter is sent to a specific city. The city, in turn, sends the letter to the home of the addressee.

The behavior structure of the use case **Send Letter** is decentralized.

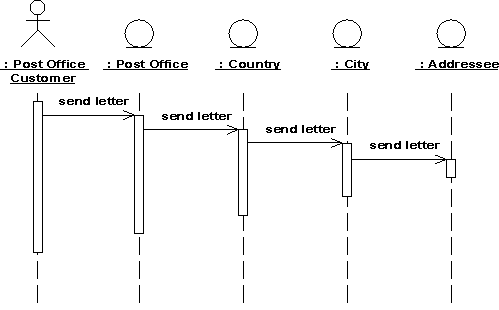
The use case behavior is a decentralized flow of events. The sub-event phases belong together. The sender of the letter speaks of "sending a letter to someone." He neither needs nor wants to know the details of how letters are forwarded in countries or cities. (Probably, if someone were mailing a letter within the same country, not all these actions would occur.)

The type of control used depends on the application. In general, you should try to achieve independent objects, that is, to delegate various tasks to the objects most naturally suited to perform them.

A flow of events with centralized control will have a "fork-shaped" sequence diagram. On the other hand, a "stairway-shaped" sequence diagram illustrates that the control-structure is decentralized for the participating objects.

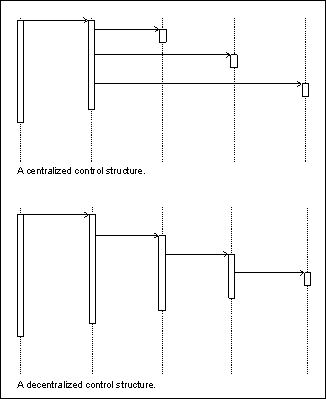
A centralized control structure in a flow of events produces a "fork-shaped" sequence diagram. A decentralized control structure produces a "stairway-shaped" sequence diagram.

The behavior structure of a use-case realization most often consists of a mix of centralized and decentralized behavior.



A decentralized structure is appropriate:

* If the sub-event phases are tightly coupled. This will be the case if the participating objects:
* Form a part-of or consists-of hierarchy, such as Country - State - City;
* Form an information hierarchy, such as CEO - Division Manager - Section Manager;
* Represent a fixed chronological progression (the sequence of sub-event phases will always be performed in the same order), such as Advertisement - Order - Invoice -Delivery - Payment; or
* Form a conceptual inheritance hierarchy, such as Animal - Mammal - Cat.
* If you want to encapsulate, and thereby make abstractions of, functionality. This is good for someone who always wants to use the whole functionality, because the functionality can become unnecessarily hard to grasp if the behavior structure is centralized.
* A centralized structure is appropriate:
* If the order in which the sub-event phases will be performed is likely to change.
* If you expect to insert new sub-event phases.
* If you want to keep parts of the functionality reusable as separate pieces.



## 4.5. Collaboration Diagram

A collaboration diagram describes a pattern of interaction among objects; it shows the objects participating in the interaction by their links to each other and the messages that they send to each other.

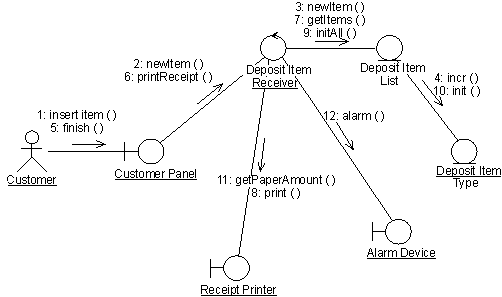
Collaboration diagrams are used to show how objects interact to perform the behavior of a particular use case, or a part of a use case. Along with sequence diagrams, collaborations are used by designers to define and clarify the roles of the objects that perform a particular flow of events of a use case. They are the primary source of information used to determining class responsibilities and interfaces.

Unlike a sequence diagram, a collaboration diagram shows the relationships among the objects. Sequence diagrams and collaboration diagrams express similar information, but show it in different ways. Collaboration diagrams show the relationships among objects and are better for understanding all the effects on a given object and for procedural design.

Because of the format of the collaboration diagram, they tend to better suited for analysis activities. Specifically, they tend to be better suited to depicting simpler interactions of smaller numbers of objects. As the number of objects and messages grows, the diagram becomes increasingly hard to read. In addition, it is difficult to show additional descriptive information such as timing, decision points, or other unstructured information that can be easily added to the notes in a sequence diagram.

### 4.5.1. Contents of Collaboration Diagrams

You can have objects and actor instances in collaboration diagrams, together with links and messages describing how they are related and how they interact. The diagram describes what takes place in the participating objects, in terms of how the objects communicate by sending messages to one another. You can make a collaboration diagram for each variant of a use case's flow of events.



A collaboration diagram that describes part of the flow of events of the use case Receive Deposit Item in the Recycling-Machine System.

### 4.5.2. Constructs of Collaboration Diagram:

Objects

An object is represented by an object symbol showing the name of the object and its class underlined, separated by a colon:

objectname : classname

You can use objects in collaboration diagrams in the following ways:

An object's class can be unspecified. Normally you create a collaboration diagram with objects first and specify their classes later.

The objects can be unnamed, but you should name them if you want to discriminate different objects of the same class.

An object's class can itself be represented in a collaboration diagram, if it actively participates in the collaboration.

Actors

Normally an actor instance occurs in the collaboration diagram, as the invoker of the interaction. If you have several actor instances in the same diagram, try keeping them in the periphery of the diagram.

Links

Links are defined as follows:

A link is a relationship among objects across which messages can be sent. In collaboration diagrams, a link is shown as a solid line between two objects.

An object interacts with, or navigates to, other objects through its links to these objects.

A link can be an instance of an association, or it can be anonymous, meaning that its association is unspecified.

Message flows are attached to links.

Messages

A message is a communication between objects that conveys information with the expectation that activity will ensue. In collaboration diagrams, a message is shown as a labeled arrow placed near a link. This means that the link is used to transport, or otherwise implement the delivery of the message to the target object. The arrow points along the link in the direction of the target object (the one that receives the message). The arrow is labeled with the name of the message, and its parameters. The arrow may also be labeled with a sequence number to show the sequence of the message in the overall interaction. Sequence numbers are often used in collaboration diagrams, because they are the only way of describing the relative sequencing of messages.

A message can be unassigned, meaning that its name is a temporary string that describes the overall meaning of the message. You can later assign the message by specifying the operation of the message's destination object. The specified operation will then replace the name of the message.

## 4.6. Operation Contracts

A UML Operation contract identifies system state changes when an operation happens. Effectively, it will define what each system operation does. An operation is taken from a system sequence diagram. It is a single event from that diagram. A domain model can be used to help generate an operation contract.

Operation Contract Syntax

Name: appropriateName

Responsibilities: Perform a function

Cross References: System functions and Use Cases

Exceptions: none

Preconditions: Something or some relationship exists

Postconditions: An association was formed

When making an operation contract, think of the state of the system before the action (snapshot) and the state of the system after the action (a second snapshot). The conditions both before and after the action should be described in the operation contract. Do not describe how the action or state changes were done. The pre and post conditions describe state, not actions.

Typical postcondion changes:

* Object attributes were changed.
* An instance of an object was created.
* An association was formed or broken.
* Postconditions are described in the past tense. They declare state changes to the system. Fill in the name, then responsibilities, then postconditions.

## 4.7. Design Class Diagram

Classes are the work-horses of the design effort—they actually perform the real work of the system. The other design elements—subsystems, packages and collaborations simply describe how classes are grouped or how they interoperate.

Capsules are also stereotyped classes, used to represent concurrent threads of execution in real-time systems. In such cases, other design classes are 'passive' classes, used within the execution context provided by the 'active' capsules. When the software architect and designer choose not to use a design approach based on capsules, it is still possible to model concurrent behavior using 'active' classes.

Active classes are design classes, which coordinate and drive the behavior of the passive classes - an active class is a class whose instances are active objects, owning their own thread of control.

### 4.7.1. Create Initial Design Classes

Start by identifying one or several (initial) design classes from the domain model, and assign trace dependencies. The design classes created in this step will be refined, adjusted, split and/or merged in the subsequent steps when assigned various "design" properties, such as operations, methods, and a state machine, describing how the analysis class is designed.

Depending on the type of the analysis class (boundary, entity, or control) that is to be designed, there are specific strategies that can be used to create initial design classes.

### 4.7.2. Designing Boundary Classes

The general rule in analysis is that there will be one boundary class for each window, or one for each form, in the user interface. The consequence of this is that the responsibilities of the boundary classes can be on a fairly high level, and need then be refined and detailed in this step.

The design of boundary classes depends on the user interface (or GUI) development tools available to the project. Using current technology, it is quite common that the user interface is visually constructed directly in the development tool, thereby automatically creating user interface classes that need to be related to the design of control and/or entity classes. If the GUI development environment automatically creates the supporting classes it needs to implement the user interface, there is no need to consider them in design - only design what the development environment does not create for you.

Additional input to this work are sketches, or screen dumps from an executable user-interface prototype, that may have been created to further specify the requirements made on the boundary classes.

Boundary classes which represent the interfaces to existing systems are typically modeled as subsystems, since they often have complex internal behavior. If the interface behavior is simple (perhaps acting as only a pass-through to an existing API to the external system) one may choose to represent the interface with one or more design classes. If this route is chosen, use a single design class per protocol, interface, or API, and note special requirements about used standards and so on in the special requirements of the class.

### 4.7.3. Designing Entity Classes

During analysis, entity classes represent manipulated units of information; entity objects are often passive and persistent. In analysis, these entity classes may have been identified and associated with the analysis mechanism for persistence. Performance considerations may force some re-factoring of persistent classes, causing changes to the Design Model, which are discussed jointly between the Database Designer and the Designer.

### 4.7.4. Designing Control Classes

A control object is responsible for managing the flow of a use case and thus coordinates most of its actions; control objects encapsulate logic that is not particularly related to user interface issues (boundary objects), or to data engineering issues (entity objects). This logic is sometimes called application logic, or business logic.

Given this, at least the following issues need to be taken into consideration when control classes are designed:

Complexity:

Simple controlling or coordinating behavior can be handled by boundary and/or entity classes. As the complexity of the application grows, however, significant drawbacks to this approach surface:

* The use case coordinating behavior becomes imbedded in the UI, making it more difficult to change the system.
* The same UI cannot be used in different use case realizations without difficulty.
* The UI becomes burdened with additional functionality, degrading its performance.
* The entity objects may become burdened with use-case specific behavior, reducing their generality.

To avoid these problems, control classes are introduced to provide behavior related to coordinating flows-of-events

Change probability

If the probability of changing flows of events is low, or the cost is negligible, the extra expense and complexity of additional control classes may not be justified.

Distribution and performance

The need to run parts of the application on different nodes or in different process spaces introduces the need for specialization of design model elements. This specialization is often accomplished by adding control objects and distributing behavior from the boundary and entity classes onto the control classes. In doing this, the boundary classes migrate toward providing purely UI services, and the entity classes toward providing purely data services, with the control classes providing the rest.

Transaction management:

Managing transactions is a classic coordination activity. Absent a framework to handle transaction management, one would have one or more transaction manager classes which would interact to ensure that the integrity of transactions is maintained.

Note that in the latter two cases, if the control class represents a separate thread of control it may be more appropriate to use an active class to model the thread of control.

### 4.7.5. Identify Persistent Classes

Classes which need to be able to store their state on a permanent medium are referred to as 'persistent'. The need to store their state may be for permanent recording of class information, for back-up in case of system failure, or for exchange of information. A persistent class may have both persistent and transient instances; labeling a class 'persistent' means merely that some instances of the class may need to be persistent.

Identifying persistent classes serves to notify the Database Designer that the class requires special attention to its physical storage characteristics. It also notifies the Software Architect that the class needs to be persistent, and the Designer responsible for the persistence mechanism that instances of the class need to be made persistent.

Because of the need for a coordinated persistence strategy, the Database Designer is responsible for mapping persistent classes into the database, using a persistence framework. If the project is developing a persistence framework, the framework developer will also be responsible for understanding the persistence requirements of design classes. To provide these people with the information they need, it is sufficient at this point to simply indicate that the class (or more precisely, instances of the class) are persistent. Also incorporate any design mechanisms corresponding to persistency mechanisms found during analysis.

Example

The analysis mechanism for persistency might be realized by one of the following design mechanisms:

* In-memory storage
* Flash card
* Binary file
* Database Management System (DBMS)

depending on what is required by the class.

Note that persistent objects may not only be derived from entity classes; persistent objects may also be needed to handle non-functional requirements in general. Examples are persistent objects needed to maintain information relevant to process control, or to maintain state information between transactions.

### 4.7.6. Define Class Visibility

For each class, determine the class visibility within the package in which it resides. A 'public' class may be referenced outside the containing package. A 'private' class (or one whose visibility is 'implementation') may only be referenced by classes within the same package.

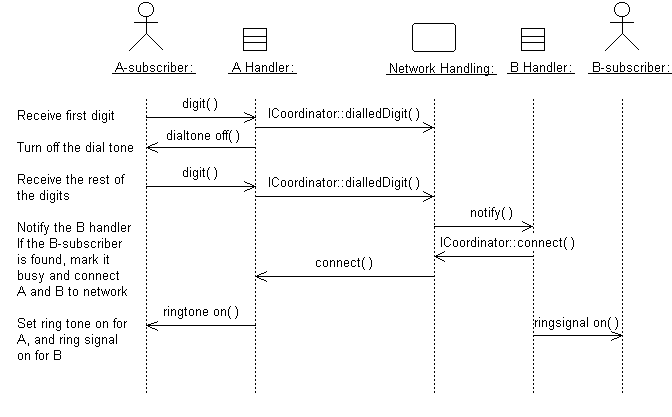
* Define Operations
* Identify Operations
* Name and Describe the Operations
* Define Operation Visibility
* Define Class Operations

Identify Operations

To identify Operations on design classes:

* Study the responsibilities of each corresponding analysis class, creating an operation for each responsibility. Use the description of the responsibility as the initial description of the operation.
* Study the use-case realizations in the class participates to see how the operations are used by the use-case realizations. Extend the operations, one use-case realization at the time, refining the operations, their descriptions, return types and parameters. Each use-case realization's requirements as regards classes are textually described in the Flow of Events of the use-case realization.
* Study the use case Special Requirements, to make sure that you do not miss implicit requirements on the operation that might be stated there.

Operations are required to support the messages that appear on sequence diagrams because scripts; messages (temporary message specifications) which have not yet been assigned to operations describe the behavior the class is expected to perform. An example sequence diagram is shown below:



Messages form the basis for identifying operations.

Do not define operations, which merely get and set the values of public attributes. These are generally generated by code generation facilities and do not need to be explicitly defined.

Name and Describe the Operations

The naming conventions of the implementation language should be used when naming operations, return types, and parameters and their types.

For each operation, you should define the following:

Operation name:

The name should be short and descriptive of the result the operation achieves.

The names of operations should follow the syntax of the implementation language. Example: find\_location would be acceptable for C++ or Visual Basic, but not for Smalltalk (in which underscores are not used); a better name for all would be findLocation.

Avoid names that imply how the operation is performed (example: Employee.wages() is better than Employee.calculateWages(), since the latter implies a calculation is performed. The operation may simply return a value in a database).

The name of an operation should clearly show its purpose. Avoid unspecific names, such as getData, that are not descriptive about the result they return. Use a name that shows exactly what is expected, such as getAddress. Better yet, simply let the operation name be the name of the property which is returned or set; if it has a parameter, it sets the property, if it has no parameter it gets the property. Example: the operation address returns the address of a Customer, while address(aString) sets or changes the address of the Customer. The 'get' and 'set' nature of the operation are implicit from the signature of the operation.

Operations that are conceptually the same should have the same name even if different classes define them, they are implemented in entirely different ways, or they have a different number of parameters. An operation that creates an object, for example, should have the same name in all classes.

If operations in several classes have the same signature, the operation must return the same kind of result, appropriate for the receiver object. This is an example of the concept of polymorphism, which says that different objects should respond to the same message in similar ways. Example: the operation name should return the name of the object, regardless how the name is stored or derived. Following this principle makes the model easier to understand.

The return type:

The return type should be the class of object that is returned by the operation.

A short description:

As meaningful as we try to make it, the name of the operation is often only vaguely useful in trying to understand what the operation does. Give the operation a short description consisting of a couple of sentences, written from the operation user's perspective.

The parameters. For each parameter, create a short descriptive name, decide on its class, and give it a brief description. As you specify parameters, remember that fewer parameters mean better reusability. A small number of parameters makes the operation easier to understand and hence there is a higher likelihood of finding similar operations. You may need to divide an operation with many parameters into several operations. The operation must be understandable to those who want to use it. The brief description should include the following:

* The meaning of the parameters (if not apparent from their names).
* Whether the parameter is passed by value or by reference
* Parameters which must have values supplied
* Parameters which can be optional, and their default values if no value is provided
* Valid ranges for parameters (if applicable)
* What is done in the operation.
* Which by reference parameters are changed by the operation.

Once you have defined the operations, complete the sequence diagrams with information about which operations are invoked for each message.

Define Operation Visibility

For each operation, identify the export visibility of the operation. The following choices exist:

* Public: the operation is visible to model elements other than the class itself.
* Implementation: the operation is visible only within to the class itself.
* Protected: the operation is visible only to the class itself, to its subclasses, or to friends of the class (language dependent)
* Private: the operation is only visible to the class itself and to friends of the class

Choose the most restricted visibility possible which can still accomplish the objectives of the operation. In order to do this, look at the sequence diagrams, and for each message determine whether the message is coming from a class outside the receiver's package (requires public visibility), from inside the package (requires implementation visibility), from a subclass (requires protected visibility) or from the class itself or a friend (requires private visibility).

Define Class Operations

For the most part, operations are 'instance' operations, that is, they are performed on instances of the class. In some cases, however, an operation applies to all instances of the class, and thus is a class-scope operation. The 'class' operation receiver is actually an instance of a metaclass, the description of the class itself, rather than any specific instance of the class. Examples of class operations include messages, which create (instantiate) new instances, which return all instances of a class, and so on.

To denote a class-scope operation, the operation string is underlined.

A method specifies the implementation of an operation. In many cases, methods are implemented directly in the programming language, in cases where the behavior required by the operation is sufficiently defined by the operation name, description and parameters. Where the implementation of an operation requires use of a specific algorithm, or requires more information than is presented in the operation's description, a separate method description is required. The method describes how the operation works, not just what it does.

* The method, if described, should discuss:
* How operations are to be implemented.
* How attributes are to be implemented and used to implement operations.
* How relationships are to be implemented and used to implement operations.

The requirements will naturally vary from case to case. However, the method specifications for a class should always state:

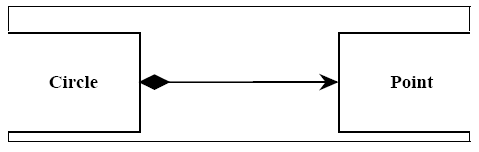
* What is to be done according to the requirements?
* What other objects and their operations are to be used?
* More specific requirements may concern:
* How parameters are to be implemented.
* Any special algorithms to be used.

Sequence diagrams are an important source for this. From these it is clear what operations are used in other objects when an operation is performed. A specification of what operations are to be used in other objects is necessary for the full implementation of an operation. The production of a complete method specification thus requires that you identify the operations for the objects involved and inspect the corresponding sequence diagrams.

### 4.7.7. Design Class Relationships

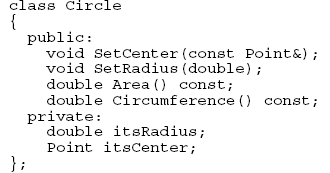
Composition Relationship

Each instance of type Circle seems to contain an instance of type Point. This is a relationship known as composition. It can be depicted in UML using a class relationship. Figure shows the composition relationship.



The black diamond represents composition. It is placed on the Circle class because it is the Circle that is composed of a Point. The arrowhead on the other end of the relationship denotes that the relationship is navigable in only one direction. That is, Point does not know about Circle. In UML relationships are presumed to be bidirectional unless the arrowhead is present to restrict them. Had I omitted the arrowhead, it would have meant that Point knew about Circle. At the code level, this would imply a #include “circle.h” within point.h. For this reason, I tend to use a lot of arrowheads.Composition relationships are a strong form of containment or aggregation. Aggregation is a whole/part relationship. In this case, Circle is the whole, and Point is part of Circle. How-ever, composition is more than just aggregation. Composition also indicates that the lifetime of Point is dependent upon Circle. This means that if Circle is destroyed, Point will be destroyed with it. For those of you who are familiar with the Booch-94 notation, this is the Has-by-value relationship.

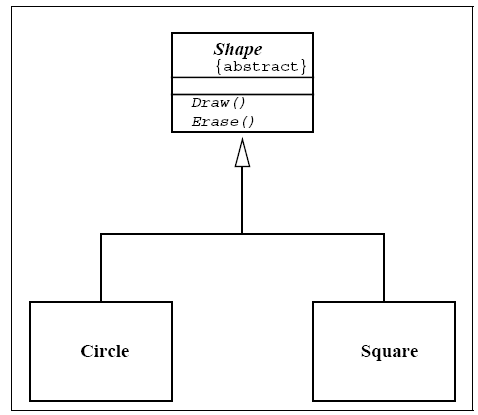
In C++ we would represent this as shown in Listing 1.In this case we have represented the composition relationship as a member variable. We could also have used a pointer so long as the destructor of Circle deleted the pointer.



**Inheritance Relationship**

A peculiar triangular arrowhead depicts the inheritance relationship in UML. This arrowhead, that looks rather like a slice of pizza, points to the base class. One or more lines proceed from the base of the arrowhead connecting it to the derived classes. Figure shows the form of the inheritance relationship. In this diagram we see that Circle and

Square both derive from Shape. Note that the name of class Shape is shown in italics. This indicates that Shape is an abstract class. Note also that the operations, Draw () and Erase () are also shown in italics. This indicates that they are pure virtual.



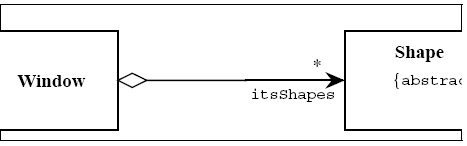
Italics are not always very easy to see. Therefore, as shown in Figure, an abstract class can also be marked with the {abstract} property. What’s more, though it is not a standard part of UML, I will often write Draw()=0 in the operations compartment to denote a pure virtual function.

**Aggregation / Association**

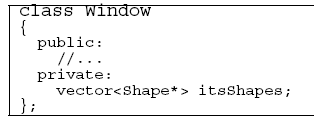
The weak form of aggregation is denoted with an open diamond. This relationship denotes that the aggregate class (the class with the white diamond touching it) is in some way the “whole”, and the other class in the relationship is somehow “part” of that whole.

Figure shows an aggregation relationship. In this case, the Window class contains many

Shape instances. In UML the ends of a relationship are referred to as its “roles”. Notice that the role at the Shape end of the aggregation is marked with a “ \*”. This indicates that the Window contains many Shape instances. Notice also that the role has been named. This is the name that Window knows its Shape instances by. i.e. it is the name of the instance variable within Window that holds all the Shapes.



Above figure might be implemented in C++ code as under:



There are other forms of containment that do not have whole / part implications. For example, each window refers back to its parent Frame. This is not aggregation since it is not reasonable to consider a parent Frame to be part of a child Window. We use the association relationship to depict this.

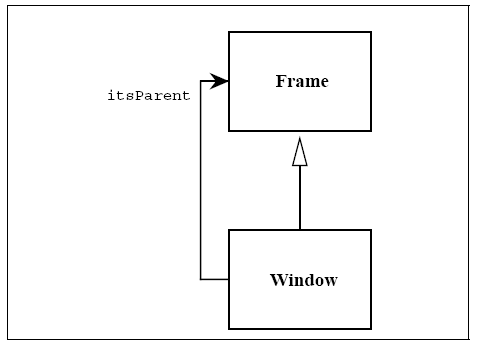


Figure shows how we draw an association. An association is nothing but a line drawn between the participating classes. In Figure 6 the association has an arrowhead to denote that Frame does not know anything about Window. Once again note the name on the role. This relationship will almost certainly be implemented with a pointer of some kind.

What is the difference between an aggregation and an association? The difference is one of implication. Aggregation denotes whole/part relationships whereas associations do not. However, there is not likely to be much difference in the way that the two relationships are implemented. That is, it would be very difficult to look at the code and determine whether a particular relationship ought to be aggregation or association. For this reason, it is pretty safe to ignore the aggregation relationship altogether. As the amigos said in the UML 0.8 document: “...if you don’t understand [aggregation] don’t use it.” Aggregation and Association both correspond to the Has-by-reference relationship from the Booch-94 notation.

Dependency

Sometimes the relationship between a two classes is very weak. They are not implemented with member variables at all. Rather they might be implemented as member function arguments. Consider, for example, the Draw function of the Shape class. Suppose that this function takes an argument of type Drawing Context.

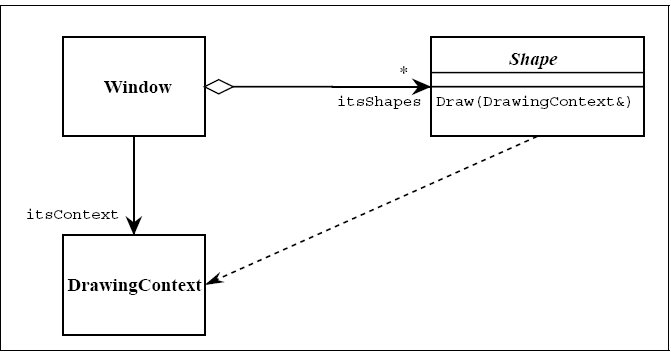
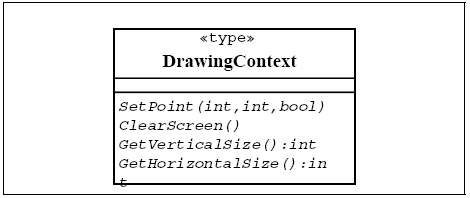
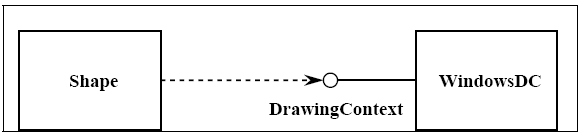


Figure shows a dashed arrow between the Shape class and the DrawingContext class. This is the dependency relationship. In Booch94 this was called a ‘using’ relationship. This relationship simply means that Shape somehow depends upon DrawingContext. In C++ this almost always results in a #include.

**Interfaces**

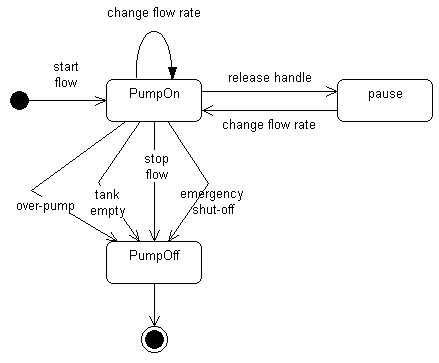
There are classes that have nothing but pure virtual functions. In Java such entities are not classes at all; they are a special language element called an interface. UML has followed the Java example and has created some special syntactic elements for such entities. The primary icon for an interface is just like a class except that it has a special denotation called a stereotype. Figure shows this icon. Note the «type» string at the top of the class. The two surrounding characters “«»” are called guillemots (pronounced Gee-**may**). A word or phrase surrounded by guillemots is called a “stereotype”. Stereotypes are one of the mechanisms that can be used to extend UML. When a stereotype is used above the name of a class it indicates that this class is a special kind of class that conforms to a rather rigid specification. The «type» stereotype indicates that the class is an interface. This means that it has no member variables, and that all of its member functions are pure virtual. UML supplies a shortcut for «type» classes. Figure 9 shows how the “lollypop” notation can be used to represent an interface. Notice that the dependency between Shape and DrawingContext is shown as usual. The class WindowsDC is derived from, or conforms to, the Drawingcontext interface. This is a shorthand notation for an inheritance relationship between





## 4.8. State chart diagram

For some operations, the behavior of the operation depends upon the state the receiver object is in. A state machine is a tool for describing the states the object can assume and the events that cause the object to move from one state to another. State machines are most useful for describing active classes. The use of state machines is particularly important for defining the behavior. An example of a simple state machine is shown below:



Each state transition event can be associated with an operation. Depending on the object's state, the operation may have a different behavior; the transition events describe how this occurs.

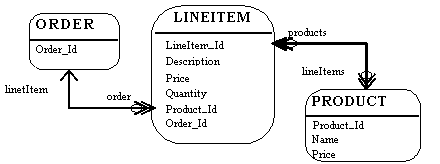
The method description for the associated operation should be updated with the state-specific information, indicating, for each relevant state, what the operation should do. States are often represented using attributes; the statechart diagrams serve as input into the attribute identification step.

## 4.9. Data Model

The data model is a subset of the implementation model, which describes the logical and physical representation of persistent data in the system.

**The Relational Data Model**

The relational model is composed of entities and relations. An entity may be a physical table or a logical projection of several tables also known as a view. The figure below illustrates LINEITEM and PRODUCT tables and the various relationships between them.



A relational model has the following elements:

An entity has columns. A name and a type identify each column. In the figure above, the LINEITEM entity has the columns LineItem\_Id (the primary key), Description, Price, Quantity, Product\_Id and Order\_Id (the latter two are foreign keys that link the LINEITEM entity to the ORDER and PRODUCT entities).

An entity has records or rows. Each row represents a unique set of information, which typically represents an object's persistent data. Each entity has one or more primary keys. The primary keys uniquely identify each record (for example, Id is the primary key for LINEITEM table).

Support for relations is vendor specific. The example illustrates the logical model and the relation between the PRODUCT and LINEITEM tables. In the physical model relations are typically implemented using foreign key / primary key references. If one entity relates to another, it will contain columns, which are foreign keys. Foreign key columns contain data, which can relate specific records in the entity to the related entity.

Relations have multiplicity (also known as cardinality). Common cardinalities are one to one (1:1), one to many (1:m), many to one (m:1), and many to many (m:n). In the example, LINEITEM has a 1:1 relationship with PRODUCT and PRODUCT has a 0:m relationship with LINEITEM.

Example

A company has several departments. Each department has a supervisor and at least one employee. Employees must be assigned to at least one, but possibly more departments. At least one employee is assigned to a project, but an employee may be on vacation and not assigned to any projects. The important data fields are the names of the departments, projects, supervisors and employees, as well as the supervisor and employee number and a unique project number.

1. Identify Entities

The entities in this system are Department, Employee, Supervisor and Project. One is tempted to make Company an entity, but it is a false entity because it has only one instance in this problem. True entities must have more than one instance.

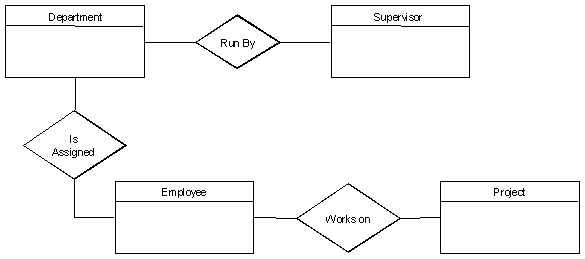
2. Find Relationships

We construct the following Entity Relationship Matrix:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Department** | **Employee** | **Supervisor** | **Project** |
| Department |  | is assigned | run by |  |
| Employee | belongs to |  |  | works on |
| Supervisor | runs |  |  |  |
| Project |  | uses |  |  |

3. Draw Rough ERD

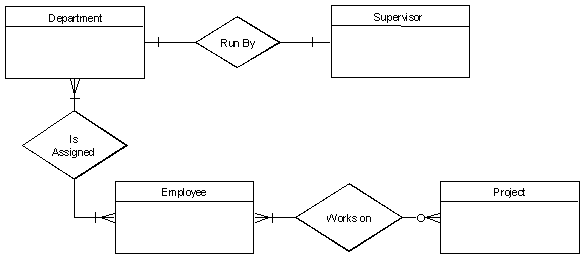
We connect the entities whenever a relationship is shown in the entity Relationship Matrix.



4. Fill in Cardinality

From the description of the problem we see that:

* Each department has exactly one supervisor.
* A supervisor is in charge of one and only one department.
* Each department is assigned at least one employee.
* Each employee works for at least one department.
* Each project has at least one employee working on it.
* An employee is assigned to 0 or more projects.

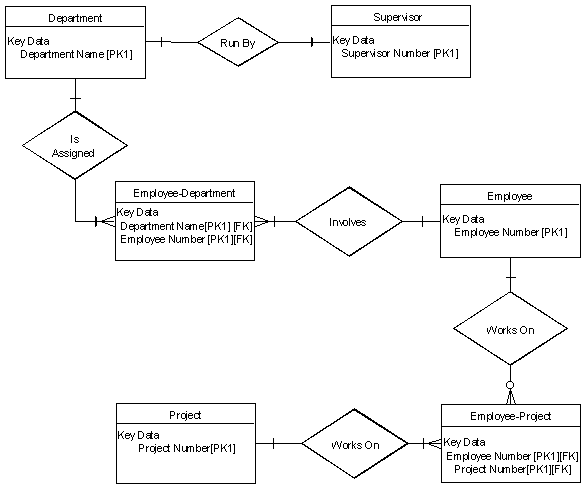


5. Define Primary Keys

The primary keys are Department Name, Supervisor Number, Employee Number, Project Number.

6. Draw Key-Based ERD

There are two many-to-many relationships in the rough ERD above, between Department and Employee and between Employee and Project. Thus we need the associative entities Department-Employee and Employee-Project. The primary key for Department-Employee is the concatenated key Department Name and Employee Number. The primary key for Employee-Project is the concatenated key Employee Number and Project Number.



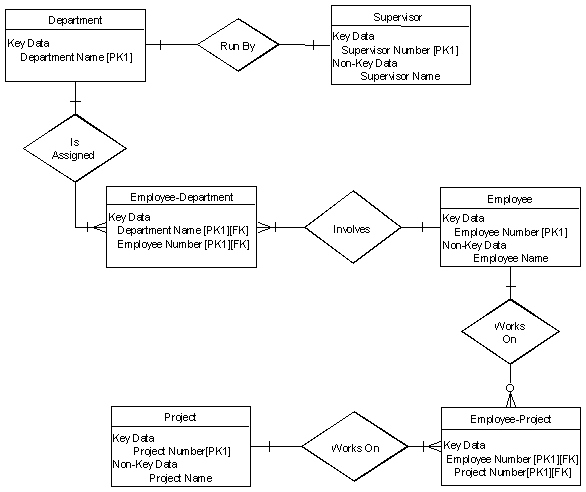
7. Identify Attributes

The only attributes indicated are the names of the departments, projects, supervisors and employees, as well as the supervisor and employee NUMBER and a unique project number.

8. Map Attributes

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Entity** | **Attribute** | **Entity** |
| Department Name | Department | Supervisor Number | Supervisor |
| Employee Number | Employee | Supervisor Name | Supervisor |
| Employee Name | Employee | Project Name | Project |
|  |  | Project Number | Project |

9. Draw Fully Attributed ERD



# Chapter 5: 2nd & 3rd Deliverable For structured Approach

## 5.1. Introduction:

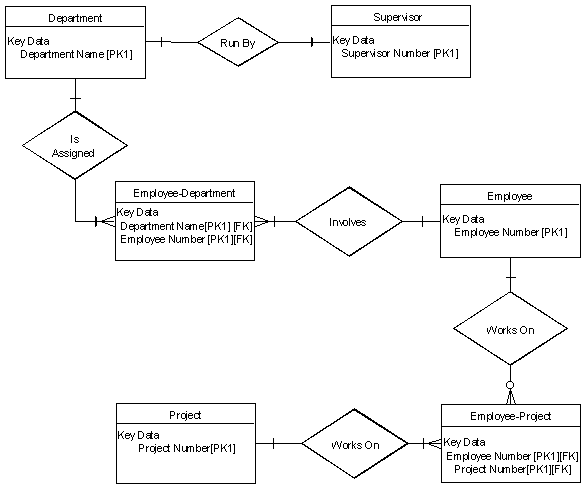
Analysis & Design Model for structured approach must contain following artifacts:

1. Entity Relationship Diagram
2. Data Flow Diagram (Functional Model)
3. State Transition Diagram (Behavioral Model)
4. Architecture Design
5. Component Level Design

## 5.2. Entity Relationship Diagram:

In the analysis model, Entity Relationship Diagram is used to understand the system under consideration with respect to entities involved and their relationships. Each entity is documented by extracted its attributes, cardinality, and modality.

**Example:**



#### Cardinality

From the description of the problem we see that:

Each department has exactly one supervisor.

A supervisor is in charge of one and only one department.

Each department is assigned at least one employee.

Each employee works for at least one department.

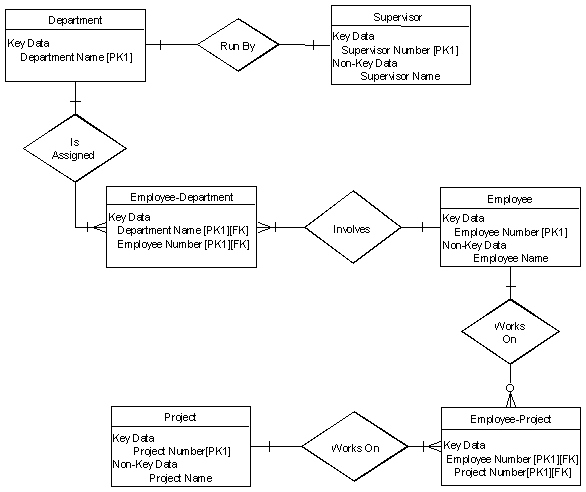
Each project has at least one employee working on it.

An employee is assigned to 0 or more projects.

#### Identify Attributes

The only attributes indicated are the names of the departments, projects, supervisors and employees, as well as the supervisor and employee NUMBER and a unique project number.

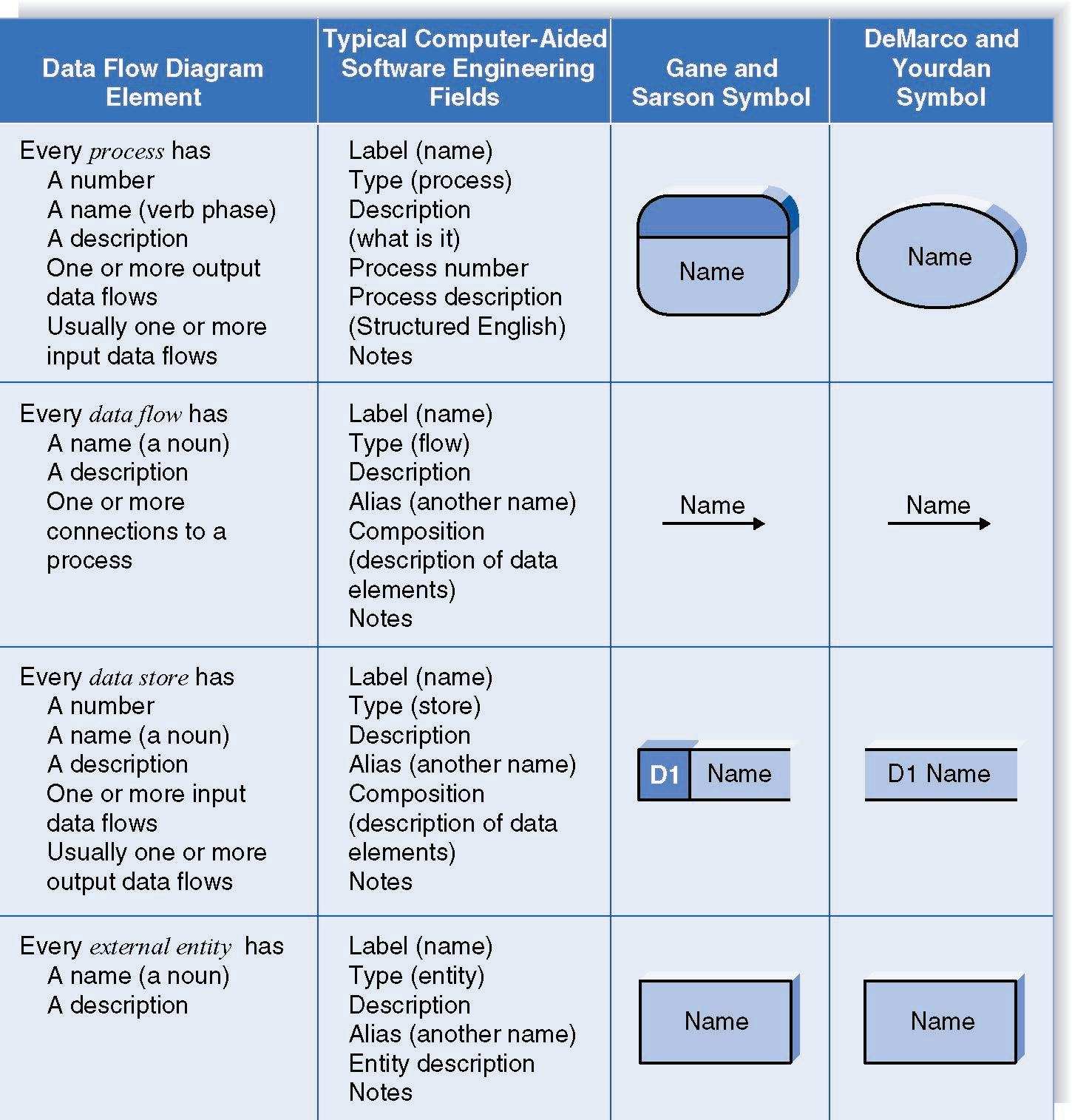
#### Fully Dressed ERD



## 5.3. Data flow diagram (Functional Model)

DFD is all about to identify the major processes in your system and develop Data Flow Diagram up to required level.

**DFD Constructs:**



Context Level DFD

A context diagram shows the context into which the business process fits. It also shows the overall business process as just one process and shows all the outside entities that receive information from or contribute information to the system.

Level 1 Diagram

This diagram shows all the processes that comprise the overall system and how information moves from and to each process. Data stores are added to it.

Level 2 Diagram

This diagram shows all the processes that comprise a single process on the level 1 diagram and how information moves from and to each of these processes. It also shows in more detail the content of higher-level process. Level 2 diagrams may not be needed for all level 1 processes.

Level 3 Diagram

This diagram shows all processes that comprise a single process on the level 2 diagram and how information moves from and to each of these processes. Level 3 diagrams may not be needed for all level 2 processes. Correctly numbering each process helps the user understand where the process fits into the overall system.

Integrating Scenario Descriptions

DFDs generally integrate scenario descriptions

Names of use cases become processes

Names of inputs and outputs become data flows

Combining “small” data inputs and outputs into a single flow

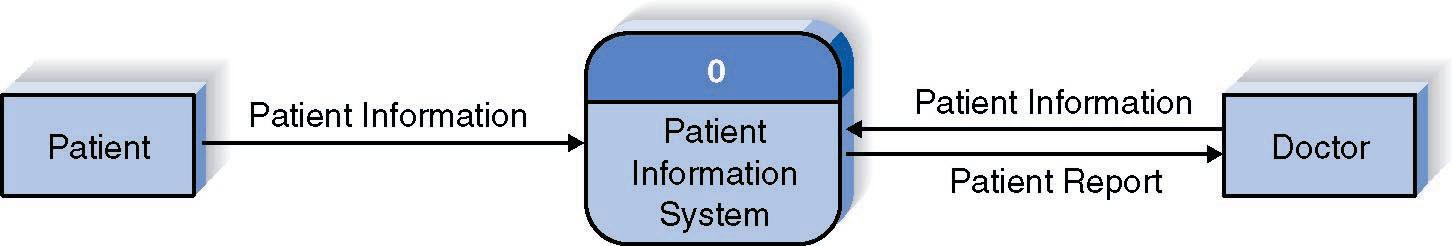
Steps in Building DFDs

* Build the context level DFD
* Create DFD fragments for each scenario
* Organize DFD fragments into level 1
* Decompose level 1 DFDs as needed
* Validate DFDs with user

DFD Fragment Tips

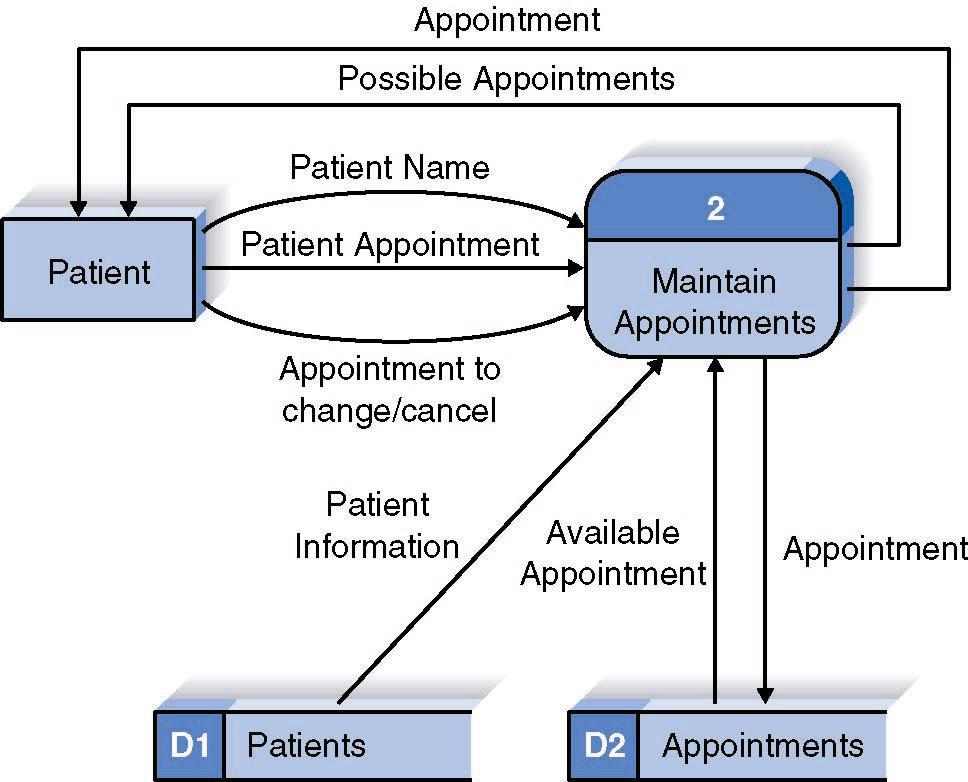
* All process names must be verb phrases
* Maintain organization’s viewpoint in naming processes
* Layouts often place
* Processes in the center
* Inputs from the left
* Outputs to the right
* Stores beneath the processes

A DFD Fragment Example



Context Level DFD

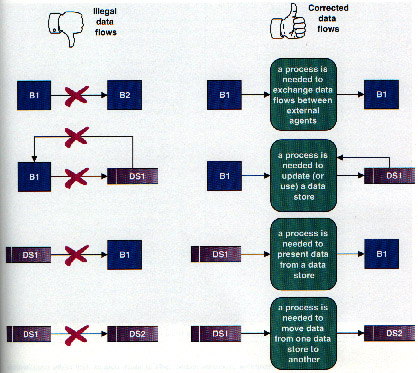
Context Level DFD



Context Level DFD

Level 1 DFD

Illegal Data Flows:



## 5.4. State Transition Diagram

State Transition Diagram is developed to represent the behavior of the system under consideration. The constructs of STD are as follows:

State

A set of observable circum-stances that characterizes the behavior of a system at a given time

State transition

The movement from one state to another

Event

An occurrence that causes the system to exhibit some predictable form of behavior

Action

Process that occurs as a consequence of making a transition

Guidelines for developing a state transition diagram

* Make a list of the different states of a system (How does the system behave?)
* Indicate how the system makes a transition from one state to another (How does the system change state?)
* Indicate event
* Indicate action
* Draw a state transition diagram

Example



State Transition Diagram for Microwave

## 5.5. Architectural design

The Focus of architecture design is the Mapping of Requirements into Software Architecture. DFD prepared in analysis model is analyzed to do it.

Two major structural patterns or two major alternatives are Transform (Flow) Analysis and Transaction (Flow) Analysis.

Beginning the Design Process

* Review the fundamental system model i.e. DFD and Software Requirement Specification document.
* Review and refine data flow diagrams for the software
* Determine whether DFD exhibits transform or transaction characteristics

Example

* There is a string conversion system.
* It has the ability to reverse strings, count number of characters, and append new strings with an old string.
* A user would be using this system and would be providing the string to it. The string would be validated. If approved the system would be displaying different choices including reversal of string, character counting and appending of strings.
* The user would select a choice and enter the appropriate selection. According to the selected choice the system would perform the required action.
* If “Reverse String” is selected the valid string is attained and reversed.
* If “Count Characters” is selected the valid string is attained and the number of characters are counted.
* If “Append String” is selected the valid string is attained and the user also enters a new string. Both the strings are appended together and the result produced.
* Whatever the choice selected the result is displayed.

See the diagrams on next page.

Context Level DFD

Level 1 DFD

Level 2 DFD

Program Structure/Design Architecture

Validate the Input

**String Conversion Executive**

Get User Selection

Reverse String

Controller

Reverse String

Processing Controller

Display Choices

Read Char.

Increment Count

Append String

Controller

Input String

Controller

Append Str

Get new Str

Display Output

Count String

Controller

Get Str

## 5.6. Component Level Design

Every component, which is appearing in program structure/design architecture, is logically analyzed and pseudocode or flow chart is prepared for each module. This flow chart is then given to programmer who translates it into a machine-readable code. The options available for component level design are:

* Flow chart
* Box Diagram
* Decision Table
* Psuedocode

# Chapter 6: 4th Deliverable (User Interface Design)

## 6.1. Introduction

A user interface design consists of three main parts:

Page elements should be visualized on paper before building them in the computer. Just as you draw a site map to plan the site, use cartoons and storyboards to begin blocking out the site’s appearance and navigational scheme.

1. Site maps
2. Storyboards
3. Navigational maps
4. Traceability Matrix

## 6.2. Site Maps

A site map's main benefit is to give users an overview of the site's areas in a single glance by dedicating an entire page to a visualization of the information architecture. If designed well, this overview can include several levels of hierarchy, and yet not be so big that users lose their ability to grasp the map as a whole. Some of the site maps we studied stretched over six screens on a standard 800x600 monitor. This is much too much. We recommend keeping the site map short; it should be no more than two-and-a-half times the window size most common among your users.

The greatest failures in our study came from site maps that attempted to lure the user into a dynamically twisting and expanding view, rather than presenting a simple, static representation of the information architecture. The site map's goal is to give users a single overview of the information space. If users have to work to reveal different parts of the map, that benefit is lost.

Dynamic site maps are basically an alternative way of navigating through the information space using a set of non-standard interaction techniques. For example, one site used a hyperbolic tree, where users had to click and drag clusters of links around the screen to expand areas of interest. Nobody could do this well.

A site map should not be a navigational challenge of its own. It should be a map.

As we have found again and again, users hated non-standard user interfaces that forced them to learn a special way of doing things for the sake of a single website. Site maps should be simple, compact layouts of links, and they should show everything in a single view.

If your site is large and complex, it is a good idea to include and index or site map that provides an outline of our site's content. This concept is something that we are all familiar with in the print world and it is very useful on the Web. An index is usually an alphabetical listing of key words that link to the appropriate pages in the site or it can be more like a table of contents. A site map is a graphical representation of the site's content. It doesn't usually have as much detailed information as the index has.

As web sites get more complicated, an index or site map is going to become more and more valuable and essential to the navigation of a good site.

## 6.3. Story boards

A storyboard is a sequence of single images, each of which represents a distinct event or narrative. It is also a visual representation of the script illustrating the interaction between the user and the machine. It can also be imagined as a film in visual-outline form.

A storyboard can be used in two ways. It describes the task, which are a series of images showing the user, environment and the machine. It also describes the interface, which represent series of screen images indicating the user’s representation and the computer’s response and work out interaction details when asking, “what happens next?” It also shows interaction sequence at a glance and helps develop usage scenarios to help develop tools & tasks.

All this can be done to construct a visual & verbal sequence that illustrates the interaction. Consider …

* Environment -- where system is used
* Visual cues -- what user can see
* Audible cues -- what user can hear
* Tactile cues -- what user can touch
* User input -- how the user communicates to the machine
* Machine output -- how the machine responds to the user
* User’s emotions -- how user perceives and responds to the interaction
* Technology -- what technology is involved in performing the task
* Quality of experience -- what benefit is perceived

HOW?

* Use a grid that puts the graphic representation above and the verbal description below
* Begin with loose thumbnail sketches and drawing for early design concepts. Refine with tighter drawings and screen designs for presentation and testing.
* Describe the interaction details and emotional responses verbally when no visual representation is effective
* Keep the medium loose and flexible in the conceptual design phase

These are the detailed screens, which pictorially represent the complete view of the screens. There would be symbols representing the different elements of the screens and in the end an index that would detail the symbols. Sample is given on the next page.

# S1: Microsoft Products

S2: Username

S3: Password

T1:

T2:

B1: **Submit**

S1: static button 1

S2: static button 2

S3: static button 3

T1: text box 1

T2: text box 2

B1: button

## 6.4. Navigational maps:

The next step is of navigational maps. In these maps, the storyboards are used as an input. The different display buttons or action buttons show the navigation from one screen to the other. In other words when one action button is pressed it would lead to other screens. This path and navigation would be shown.

## 6.5 Trace-ability Matrix

**Following columns are involved in the trace-ability matrix.**

Feature: Lists system features based on which use case are built.

Use Case ID: Write the ID of the use case for easy lookup.

UI ID: Write the user interface ID for this use case.

Priority: Give an appropriate rating to each use case according to its priority

Use Case Cross Ref: rite the related use cases separated with commas.

DB Table Id: write the relevant DB table ID’s

Elaborated Use-case ID: Elaboration of the related Use Cases separated by commas.

Dependent Classes: List of Classes separated by commas which are involved in achieving the Feature.

# Chapter 7: 5th Deliverable (Software Testing)

## 7.1 Introduction:

This deliverable is based on the IEEE standard of software testing i.e. IEEE SOFTWARE TEST DOCUMENTATION Std 829-1998. This standard describes a set of basic test documents that are associated with the dynamic aspects of software testing (i.e., the execution of procedures and code). The standard defines the purpose, outline, and content of each basic document. While the documents described in the standard focus on dynamic testing, several of them may be applicable to other testing activities (e.g., the test plan and test incident report may be used for design and code reviews). This standard may be applied to commercial, scientific, or military software that runs on any digital computer. Applicability is not restricted by the size, complexity, or criticality of the software. However, the standard does not specify any class of software to which it must be applied. The standard addresses the documentation of both initial development testing and the testing of subsequent software releases. For a particular software release, it may be applied to all phases of testing from module testing through user acceptance. However, since all of the basic test documents may not be useful in each test phase, the particular documents to be used in a phase are not specified. Each organization using the standard will need to specify the classes of software to which it applies and the specific documents required for a particular test phase.

The standard does not call for specific testing methodologies, approaches, techniques, facilities, or tools, and does not specify the documentation of their use. Additional test documentation may be required (e.g., code inspection checklists and reports). The standard also does not imply or impose specific methodologies for documentation control, configuration management, or quality assurance. Additional documentation (e.g., a quality assurance plan) may be needed depending on the particular methodologies used.

Following are standard artifacts, which must be included in this deliverable:

1. Test Plan
2. Test Design Specification
3. Test Case Specification
4. Test Procedure Specification
5. Test Item Transmittal Report
6. Test Log
7. Test Incident Report
8. Test Summary Report

## 7.2. Test plan

### 7.2.1. Purpose

To prescribe the scope, approach, resources, and schedule of the testing activities. To identify the items being tested, the features to be tested, the testing tasks to be performed, the personnel responsible for each task, and the risks associated with this plan.

### 7.2.2. Outline

A test plan shall have the following structure:

1. Test plan identifier
2. Introduction
3. Test items
4. Features to be tested
5. Features not to be tested
6. Approach
7. Item pass/fail criteria
8. Suspension criteria and resumption requirements
9. Test deliverables
10. Testing tasks
11. Environmental needs
12. Responsibilities
13. Staffing and training needs
14. Schedule
15. Risks and contingencies
16. Approvals

The sections shall be ordered in the specified sequence. Additional sections may be included immediately prior to Approvals. If some or all of the content of a section is in another document, then a reference to that material may be listed in place of the corresponding content. The referenced material must be attached to the test plan or available to users of the plan.

Details on the content of each section are contained in the following sub-clauses.

#### 7.2.2.1. Test plan identifier

Specify the unique identifier assigned to this test plan.

#### 7.2.2.2. Introduction

Summarize the software items and software features to be tested. The need for each item and its history may be included. References to the following documents, when they exist, are required in the highest-level test plan:

1. Project authorization;
2. Project plan;
3. Quality assurance plan;
4. Configuration management plan;
5. Relevant policies;
6. Relevant standards.

In multilevel test plans, each lower-level plan must reference the next higher-level plan.

#### 7.2.2.3. Test items

Identify the test items including their version/revision level. Also specify characteristics of their transmittal media that impact hardware requirements or indicate the need for logical or physical transformations before testing can begin (e.g., programs must be transferred from tape to disk).

Supply references to the following test item documentation, if it exists:

a) Requirements specification

b) Design specification

c) Users guide

d) Operations guide

e) Installation guide

Reference any incident reports relating to the test items. Items that are to be specifically excluded from testing may be identified.

#### 7.2.2.4. Features to be tested

Identify all software features and combinations of software features to be tested. Identify the test design specification associated with each feature and each combination of features.

#### 7.2.2.5. Features not to be tested

Identify all features and significant combinations of features that will not be tested and the reasons.

#### 7.2.2.6. Approach

Describe the overall approach to testing. For each major group of features or feature combinations, specify the approach that will ensure that these feature groups are adequately tested. Specify the major activities, techniques, and tools that are used to test the designated groups of features.

The approach should be described in sufficient detail to permit identification of the major testing tasks and estimation of the time required to do each one.

Specify the minimum degree of comprehensiveness desired. Identify the techniques that will be used to judge the comprehensiveness of the testing effort (e.g., determining which statements have been executed at least once). Specify any additional completion criteria (e.g., error frequency). The techniques to be used to trace requirements should be specified. Identify significant constraints on testing such as test item availability, testing resource availability, and deadlines.

#### 7.2.2.7. Item pass/fail criteria

Specify the criteria to be used to determine whether each test item has passed or failed testing.

#### 7.2.2.8. Suspension criteria and resumption requirements

Specify the criteria used to suspend all or a portion of the testing activity on the test items associated with this plan. Specify the testing activities that must be repeated, when testing is resumed.

#### 7.2.2.9. Test deliverables

Identify the deliverable documents. The following documents should be included:

1. Test plan;
2. Test design specifications;
3. Test case specifications;
4. Test procedure specifications;
5. Test item transmittal reports;
6. Test logs;
7. Test incident reports;
8. Test summary reports.

Test input data and test output data should be identified as deliverables.

Test tools (e.g., module drivers and stubs) may also be included.

#### 7.2.2.10. Testing tasks

Identify the set of tasks necessary to prepare for and perform testing. Identify all inter task dependencies and any special skills required.

#### 7.2.2.11. Environmental needs

Specify both the necessary and desired properties of the test environment. This specification should contain the physical characteristics of the facilities including the hardware, the communications and system software, the mode of usage (e.g., stand-alone), and any other software or supplies needed to support the test. Also specify the level of security that must be provided for the test facilities, system software, and proprietary components such as software, data, and hardware. Identify special test tools needed.

Identify any other testing needs (e.g., publications or office space). Identify the source for all needs that are not currently available to the test group.

#### 7.2.2.12. Responsibilities

Identify the groups responsible for managing, designing, preparing, executing, witnessing, checking, and resolving. In addition, identify the groups responsible for providing the test items identified in 7.2.2.3 and the environmental needs identified in 7.2.2.11.

These groups may include the developers, testers, operations staff, user representatives, technical support staff, data administration staff, and quality support staff.

#### 7.2.2.13 Staffing and training needs

Specify test-staffing needs by skill level. Identify training options for providing necessary skills.

#### 7.2.2.14. Schedule

Include test milestones identified in the software project schedule as well as all item transmittal events.

Define any additional test milestones needed. Estimate the time required to do each testing task. Specify the schedule for each testing task and test milestone. For each testing resource (i.e., facilities, tools, and staff), specify its periods of use.

#### 7.2.2.15. Risks and contingencies

Identify the high-risk assumptions of the test plan. Specify contingency plans for each (e.g., delayed delivery of test items might require increased night shift scheduling to meet the delivery date).

#### 7.2.2.16 Approvals

Specify the names and titles of all persons who must approve this plan. Provide space for the signatures and dates.

## 7.3. Test design specification

### 7.3.1. Purpose

To prescribe the scope, approach, resources, and schedule of the testing activities. To identify the items being tested, the features to be tested, the testing tasks to be performed, the personnel responsible for each task, and the risks associated with this plan.

### 7.3.2. Outline

A test plan shall have the following structure:

1. Test plan identifier;
2. Introduction;
3. Test items;
4. Features to be tested;
5. Features not to be tested;
6. Approach;
7. Item pass/fail criteria;
8. Suspension criteria and resumption requirements;
9. Test deliverables;
10. Testing tasks;
11. Environmental needs;
12. Responsibilities;
13. Staffing and training needs;
14. Schedule;
15. Risks and contingencies;
16. Approvals.

The sections shall be ordered in the specified sequence. Additional sections may be included immediately prior to Approvals. If some or all of the content of a section is in another document, then a reference to that material may be listed in place of the corresponding content. The referenced material must be attached to the test plan or available to users of the plan.

Details on the content of each section are contained in the following sub-clauses.

#### 7.3.2.1 Test plan identifier

Specify the unique identifier assigned to this test plan.

#### 7.3.2.2. Introduction

Summarize the software items and software features to be tested. The need for each item and its history may be included. References to the following documents, when they exist, are required in the highest-level test plan:

1. Project authorization
2. Project plan
3. Quality assurance plan
4. Configuration management plan
5. Relevant policies
6. Relevant standards

In multilevel test plans, each lower-level plan must reference the next higher-level plan.

#### 7.3.2.3. Test items

Identify the test items including their version/revision level. Also specify characteristics of their transmittal media that impact hardware requirements or indicate the need for logical or physical transformations before testing can begin (e.g., programs must be transferred from tape to disk). Supply references to the following test item documentation, if it exists:

1. Requirements specification
2. Design specification
3. Users guide
4. Operations guide
5. Installation guide

Reference any incident reports relating to the test items. Items that are to be specifically excluded from testing may be identified.

#### 7.3.2.4. Features to be tested

Identify all software features and combinations of software features to be tested. Identify the test design specification associated with each feature and each combination of features.

#### 7.3.2.5. Features not to be tested

Identify all features and significant combinations of features that will not be tested and the reasons.

#### 7.3.2.6. Approach

Describe the overall approach to testing. For each major group of features or feature combinations, specify the approach that will ensure that these feature groups are adequately tested. Specify the major activities, techniques, and tools that are used to test the designated groups of features.

The approach should be described in sufficient detail to permit identification of the major testing tasks and estimation of the time required to do each one. Specify the minimum degree of comprehensiveness desired. Identify the techniques that will be used to judge the comprehensiveness of the testing effort (e.g., determining which statements have been executed at least once).

Specify any additional completion criteria (e.g., error frequency). The techniques to be used to trace requirements should be specified. Identify significant constraints on testing such as test item availability, testing resource availability, and deadlines.

#### 7.3.2.7. Item pass/fail criteria

Specify the criteria to be used to determine whether each test item has passed or failed testing.

#### 7.3.2.8. Suspension criteria and resumption requirements

Specify the criteria used to suspend all or a portion of the testing activity on the test items associated with this plan. Specify the testing activities that must be repeated, when testing is resumed.

#### 7.3.2.9. Test deliverables

Identify the deliverable documents. The following documents should be included:

1. Test plan
2. Test design specifications
3. Test case specifications
4. Test procedure specifications
5. Test item transmittal reports
6. Test logs
7. Test incident reports
8. Test summary reports

Test input data and test output data should be identiÞed as deliverables. Test tools (e.g., module drivers and stubs) may also be included.

#### 7.3.2.10. Testing tasks

Identify the set of tasks necessary to prepare for and perform testing. Identify all inter task dependencies and any special skills required.

#### 7.3.2.11. Environmental needs

Specify both the necessary and desired properties of the test environment. This specification should contain the physical characteristics of the facilities including the hardware, the communications and system software, the mode of usage (e.g., stand-alone), and any other software or supplies needed to support the test. Also specify the level of security that must be provided for the test facilities, system software, and proprietary components such as software, data, and hardware. Identify special test tools needed.

Identify any other testing needs (e.g., publications or office space). Identify the source for all needs that are not currently available to the test group.

#### 7.3.2.12. Responsibilities

Identify the groups responsible for managing, designing, preparing, executing, witnessing, checking, and resolving. In addition, identify the groups responsible for providing the test items identified in 7.2.2.3 and the environmental needs identified in 7.3.2.11.

These groups may include the developers, testers, operations staff, user representatives, technical support staff, data administration staff, and quality support staff.

#### 7.3.2.13. Staffing and training needs

Specify test-staffing needs by skill level. Identify training options for providing necessary skills.

#### 7.3.2.14. Schedule

Include test milestones identified in the software project schedule as well as all item transmittal events. Define any additional test milestones needed. Estimate the time required to do each testing task. Specify the schedule for each testing task and test milestone. For each testing resource (i.e., facilities, tools, and staff), specify its periods of use.

#### 7.3.2.15. Risks and contingencies

Identify the high-risk assumptions of the test plan. Specify contingency plans for each (e.g., delayed delivery of test items might require increased night shift scheduling to meet the delivery date)

#### 7.3.2.16. Approvals

Specify the names and titles of all persons who must approve this plan. Provide space for the signatures and dates.

## 7.4. Test Case Specification

### 7.4.1. Purpose

To define a test case identified by a test design specification.

### 7.4.2. Outline

A test case specification shall have the following structure:

1. Test case specification identifier
2. Test items
3. Input specifications
4. Output specifications
5. Environmental needs
6. Special procedural requirements
7. Inter case dependencies

The sections shall be ordered in the specified sequence. Additional sections may be included at the end. If some or all of the content of a section is in another document, then a reference to that material may be listed in place of the corresponding content. The referenced material must be attached to the test case specification or available to users of the case specification. Since a test case may be referenced by several test design specifications used by different groups over a long time period, enough specific information must be included in the test case specification to permit reuse.

Details on the content of each section are contained in the following sub-clauses.

#### 7.4.2.1. Test case specification identifier

Specify the unique identifier assigned to this test case specification.

#### 7.4.2.2 Test items

Identify and briefly describe the items and features to be exercised by this test case.

For each item, consider supplying references to the following test item documentation:

1. Requirements specification
2. Design specification
3. Users guide
4. Operations guide
5. Installation guide

#### 7.4.2.3. Input specifications

Specify each input required to execute the test case. Some of the inputs will be specified by value (with tolerances where appropriate), while others, such as constant tables or transaction files, will be specified by name. Identify all appropriate databases, files, terminal messages, memory resident areas, and values passed by the operating system.

Specify all required relationships between inputs (e.g., timing).

#### 7.4.2.4. Output specifications

Specify all of the outputs and features (e.g., response time) required of the test items. Provide the exact value (with tolerances where appropriate) for each required output or feature.

#### 7.4.2.5. Environmental needs

##### 7.4.2.5.1. Hardware

Specify the characteristics and configurations of the hardware required to execute this test case (e.g., 132 character´ 24 line CRT).

##### 7.4.2.5.2. Software

Specify the system and application software required to execute this test case. This may include system software such as operating systems, compilers, simulators, and test tools. In addition, the test item may interact with application software.

##### 7.4.2.5.3. Other

Specify any other requirements such as unique facility needs or specially trained personnel.

#### 7.4.2.6. Special procedural requirements

Describe any special constraints on the test procedures that execute this test case. These constraints may involve special set up, operator intervention, output determination procedures, and special wrap up.

#### 7.4.2.7. Inter case dependencies

List the identifiers of test cases that must be executed prior to this test case. Summarize the nature of the dependencies.

## 7.5. Test procedure specification

### 7.5.1. Purpose

To specify the steps for executing a set of test cases or, more generally, the steps used to analyze a software item in order to evaluate a set of features.

### 7.5.2 Outline

A test procedure specification shall have the following structure:

1. Test procedure specification identifier
2. Purpose
3. Special requirements
4. Procedure steps

The sections shall be ordered in the specified sequence. Additional sections, if required, may be included at the end. If some or all of the content of a section is in another document, then a reference to that material may be listed in place of the corresponding content. The referenced material must be attached to the test procedure specification or available to users of the procedure specification.

Details on the content of each section are contained in the following sub clauses.

#### 7.5.2.1. Test procedure specification identifier

Specify the unique identifier assigned to this test procedure specification. Supply a reference to the associated test design specification.

#### 7.5.2.2. Purpose

Describe the purpose of this procedure. If this procedure executes any test cases, provide a reference for each of them. In addition, provide references to relevant sections of the test item documentation (e.g., references to usage procedures).

#### 7.5.2.3. Special requirements

Identify any special requirements that are necessary for the execution of this procedure. These may include prerequisite procedures, special skills requirements, and special environmental requirements.

#### 7.5.2.4. Procedure steps

Include the steps in 8.5.2.4.1. through 8.5.2.4.10 as applicable.

##### 7.5.2.4.1. Log

Describe any special methods or formats for logging the results of test execution, the incidents observed, and any other events pertinent to the test (see Clauses 9 and 10).

##### 7.5.2.4.2. Set up

Describe the sequence of actions necessary to prepare for execution of the procedure.

##### 7.5.2.4.3. Start

Describe the actions necessary to begin execution of the procedure.

##### 7.5.2.4.4. Proceed

Describe any actions necessary during execution of the procedure.

##### 7.5.2.4.5. Measure

Describe how the test measurements will be made (e.g., describe how remote terminal response time is to be measured using a network simulator).

##### 7.5.2.4.6. Shut down

Describe the actions necessary to suspend testing, when unscheduled events dictate.

##### 7.5.2.4.7. Restart

Identify any procedural restart points and describe the actions necessary to restart the procedure at each of these points.

##### 7.5.2.4.8. Stop

Describe the actions necessary to bring execution to an orderly halt.

##### 7.5.2.4.9. Wrap up

Describe the actions necessary to restore the environment.

##### 7.5.2.4..10. Contingencies

Describe the actions necessary to deal with anomalous events that may occur during execution.

## 7.6. Test item transmittal report

### 7.6.1. Purpose

To identify the test items being transmitted for testing. It includes the person responsible for each item, its physical location, and its status. Any variations from the current item requirements and designs are noted in this report.

### 7.6.2. Outline

A test item transmittal report shall have the following structure:

1. Transmittal report identifier
2. Transmitted items
3. Location
4. Status
5. Approvals

The sections shall be ordered in the specified sequence. Additional sections may be included just prior to Approvals. If some or all of the content of a section is in another document, then a reference to that material may be listed in place of the corresponding content. The referenced material must be attached to the test item transmittal report or available to users of the transmittal report.

Details on the content of each section are contained in the following sub clauses.

#### 7.6.2.1. Transmittal report identifier

Specify the unique identifier assigned to this test item transmittal report.

#### 7.6.2.2. Transmitted items

Identify the test items being transmitted, including their version/revision level. Supply references to the item documentation and the test plan relating to the transmitted items. Indicate the people responsible for the transmitted items.

#### 7.6.2.3. Location

Identify the location of the transmitted items. Identify the media that contain the items being transmitted. When appropriate, indicate how specific media are labeled or identified.

#### 7.6.2.4. Status

Describe the status of the test items being transmitted. Include deviations from the item documentation, from previous transmittals of these items, and from the test plan. List the incident reports that are expected to be resolved by the transmitted items. Indicate if there are pending modifications to item documentation that may affect the items listed in this transmittal report.

#### 7.6.2.5. Approvals

Specify the names and titles of all persons who most approve this transmittal. Provide space for the signatures and dates.

## 7.7. Test log

### 7.7.1. Purpose

To provide a chronological record of relevant details about the execution of tests.

### 7.7.2. Outline

A test log shall have the following structure:

1. Test log identifier;
2. Description;
3. Activity and event entries.

The sections shall be ordered in the specified sequence. Additional sections may be included at the end. If some or all of the content of a section is in another document, then a reference to that material may be listed in place of the corresponding content. The referenced material must be attached to the test log or available to users of the log. Details on the content of each section are contained in the following sub clauses.

#### 7.7.2.1. Test log identifier

Specify the unique identifier assigned to this test log.

#### 7.7.2.2. Description

Information that applies to all entries in the log except as specifically noted in a log entry should be included here. The following information should be considered:

* Identify the items being tested including their version/revision levels. For each of these items, supply a reference to its transmittal report, if it exists.
* Identify the attributes of the environments in which the testing is conducted. Include facility identification, hardware being used (e.g., amount of memory being used, CPU model number, and number and model of tape drives, and/or mass storage devices), system software used, and resources available (e.g., the amount of memory available).

#### 7.7.2.3. Activity and event entries

For each event, including the beginning and end of activities, record the occurrence date and time along with the identity of the author. The information in 9.2.3.1 through 9.2.3.5 should be considered:

##### 7.7.2.3.1. Execution description

Record the identifier of the test procedure being executed and supply a reference to its specification. Record all personnel present during the execution including testers, operators, and observers. Also indicate the function of each individual.

##### 7.7.2.3.2. Procedure results

For each execution, record the visually observable results (e.g., error messages generated, aborts, and requests for operator action). Also record the location of any output (e.g., reel number). Record the successful or unsuccessful execution of the test.

##### 7.7.2.3.3. Environmental information

Record any environmental conditions specific to this entry (e.g., hardware substitutions).

##### 7.7.2.3.4. Anomalous events

Record what happened before and after an unexpected event occurred (e.g., A summary display was requested and the correct screen displayed, but response seemed unusually long. A repetition produced the same prolonged response). Record circumstances surrounding the inability to begin execution of a test procedure or failure to complete a test procedure (e.g., a power failure or system software problem).

##### 7.7.2.3.5. Incident report identifiers

Record the identifier of each test incident report, whenever one is generated.

## 7.8. Test incident report

### 7.8.1. Purpose

To document any event that occurs during the testing process that requires investigation.

### 7.8.2. Outline

A test incident report shall have the following structure:

1. Test incident report identifier
2. Summary
3. Incident description
4. Impact

The sections shall be ordered in the specified sequence. Additional sections may be included at the end. If some or all of the content of a section is in another document, then a reference to that material may be listed in place of the corresponding content. The referenced material must be attached to the test incident report or available to users of the incident report.

Details on the content of each section are contained in the following sub clauses.

#### 7.8.2.1. Test incident report identifier

Specify the unique identifier assigned to this test incident report.

#### 7.8.2.2. Summary

Summarize the incident. Identify the test items involved indicating their version/revision level. References to

the appropriate test procedure specification, test case specification, and test log should be supplied.

#### 7.8.2.3. Incident description

Provide a description of the incident. This description should include the following items:

1. Inputs
2. Expected results
3. Actual results
4. Anomalies
5. Date and time;
6. Procedure step;
7. Environment;
8. Attempts to repeat;
9. Testers;
10. Observers.

Related activities and observations that may help to isolate and correct the cause of the incident should be included (e.g., describe any test case executions that might have a bearing on this particular incident and any variations from the published test procedure).

#### 7.8.2.4.Impact

If known, indicate what impact this incident will have on test plans, test design specifications, test procedure specifications, or test case specifications.

## 7.9. Test summary report

### 7.9.1. Purpose

To summarize the results of the designated testing activities and to provide evaluations based on these results.

### 7.9.2. Outline

A test summary report shall have the following structure:

1. Test summary report identifier
2. Summary
3. Variances
4. Comprehensive assessment
5. Summary of results
6. Evaluation
7. Summary of activities
8. Approvals

The sections shall be ordered in the specified sequence. Additional sections may be included just prior to Approvals. If some or all of the content of a section is in another document, then a reference to that material may be listed in place of the corresponding content. The referenced material must be attached to the test summary report or available to users of the summary report.

Details on the content of each section are contained in the following sub clauses.

#### 7.9.2.1. Test summary report identifier

Specify the unique identifier assigned to this test summary report.

#### 7.9.2.2. Summary

Summarize the evaluation of the test items. Identify the items tested, indicating their version/revision level. Indicate the environment in which the testing activities took place.

For each test item, supply references to the following documents if they exist: test plan, test design specifications, test procedure specifications, test item transmittal reports, test logs, and test incident reports.

#### 7.9.2.3. Variances

Report any variances of the test items from their design specifications. Indicate any variances from the test plan, test designs, or test procedures. Specify the reason for each variance.

#### 7.9.2.4. Comprehensiveness assessment

Evaluate the comprehensiveness of the testing process against the comprehensiveness criteria specified in the test plan if the plan exists. Identify features or feature combinations that were not sufficiently tested and explain the reasons.

#### 7.9.2.5. Summary of results

Summarize the results of testing. Identify all resolved incidents and summarize their resolutions. Identify all unresolved incidents.

#### 7.9.2.6. Evaluation

Provide an overall evaluation of each test item including its limitations. This evaluation shall be based upon the test results and the item level pass/fail criteria. An estimate of failure risk may be included.

#### 7.9.2.7. Summary of activities

Summarize the major testing activities and events. Summarize resource consumption data, e.g., total staffing level, total machine time, and total elapsed time used for each of the major testing activities.

#### 7.9.2.8. Approvals

Specify the names and titles of all persons who must approve this report. Provide space for the signatures and dates.

# Appendixes:

# Appendix 1: User Interface

## 1.1. Introduction

What is user-centered design?

"A user interface is an interface that enables information to be passed between a human user and hardware or software components of a computer system."

There is no clear consensus on what constitutes user-centered design. However, John Gould and his colleagues at IBM developed an approach in the 1980’s called Design for Usability, which encompasses most commonly-accepted definitions. It developed from practical experience on a number of interactive systems, most notably IBM’s 1984 Olympic Messaging System. The approach has four main components as described below.

## 1.2. Focus on users

Gould suggests that developers should decide who the users will be and to involve them at the earliest possible opportunity. He suggests a number of ways of becoming familiar with users, their tasks and requirements:

|  |  |
| --- | --- |
|     Talk with users |     Visit customer locations |
|     Observe users working |     Videotape users working |
|     Learn about work organization |     Try it yourself |
|     Get users to think aloud while working |     Participative design |
|     Include expert users on the design team |     Perform task analysis |
|     Make use of surveys and questionnaires |     Develop testable goals |

In the Rational Unified Process (RUP), workshops are used at several key stages, but these must be complemented by the kinds of activities Gould describes if an accurate picture is to be differently from how they do it. Commonly performed tasks and seemingly unimportant details such as placement of work or the existence of "mysterious" scraps of paper are often forgotten, or omitted because they are not "officially" part of the current process.)

## 1.3. Integrated with design

Usability tasks should be performed in parallel early in development. These tasks would include sketching the user interface and drafting the user guides or online help. Gould also makes the point that usability should be the responsibility of one group.

An important feature of integrated design is that the overall approach – the framework – for detailed user-interface design is developed and tested at an early stage. This is an important difference between user-centered design and other purely incremental techniques. It ensures that incremental design carried out in later phases fits seamlessly into the framework and that the user interface is consistent in appearance, terminology and concept.

Within the RUP, this framework can be established by using a domain model to ensure that all terminology and concepts that will appear in the user interface are known and understood within the business in general and with users in particular. (There will also be subsets of the domain model that will be relevant only to specific groups of users. Care should be taken to ensure that the domain model is organized so that these subsets can be easily identified.) As user-interface design progresses in the requirements discipline, many of the domain classes will be represented as boundary classes in the interface. The boundary classes, and the relationships between them, should be consistent with the domain model and should be represented consistently through all parts of the system under design. (This not only assists users, but also improves reuse of user-interface components.)

## 1.4. Early user testing

Early user testing means early prototyping; typically drawings and mockups described as low-fidelity prototypes. Hi-fidelity prototypes will follow later in the process.

Mockups can be used in conjunction with use cases to write concrete scenarios of use for the system under design. These can take the form narrative, illustrated narrative (using the mockups for illustration), storyboards, walkthroughs (with users), and the basis of user focus groups. While these approaches are unfamiliar to many software developers, they are clearly more cost effective than the discovery of inappropriate design or misunderstood requirements once implementation is under way.

## 1.5. Iterative design

Object-oriented development has become synonymous with an iterative process. Iterative design is well suited to problems that need a refinement of understanding and have changing requirements. Not surprisingly, iterative design is a key component of user-centered design. This is partly due to the changing needs of users over time, but also the inherent complexity of producing design solutions that can deal with diverse needs.

Note that in user-centered methods, iterative design takes place within an integrated framework. We deliberately avoid incremental development, outside of an agreed framework, that might lead to a “patchwork” solution.

## 1.6. Guidelines

Following guidelines must be kept in mind while working on user interface design:

User Control

The user must feel like he or she is controlling the application, rather than feeling controlled by the application. To accommodate this perspective, the application must be able to respond to any appropriate task requests at any time. For example, if any process demands some processing time we can show progress bar on the status bar.

The user must control every thing about an application, including things like colors, fonts, and so on.

Directness

User should be able to see the direct results of actions that they take in your applications, for example, if your user wants to print a document, he or she will need to drag the document’s icon to a printer. The pictures on your icons and text on icons must explain functions to which that belong.

Consistency

An application must be consistent with the operating System. Windows provides numerous features that can aid in consistency, including standard menus, icons, and dialogs. For example file open and save file dialog are same most of the applications designed for windows.

An application must also be internally consistent. For example, fonts, text boxes, colors, messages, size, and optional marks action etc.

Forgiveness

Your software applications must be very forgiving to users who undoubtedly will make errors. For example user can click cross button on the title bar of the window by mistake word handles this situation by asking user once again.

Application must handle wrong entries to the application and guide students by displaying some messages.

Application must be terminated properly

Feedback

Your application should appropriately communicate with your users and inform them when the application is finished, busy, or in any other critical status.

These cues can be combination of both visual and audio, increasing the likelihood that users understand them.

We can use change in mouse pointer, color variation, Message boxes and progress bar.

Status bar with proper message is also a good way to do this act.

Aesthetics

Users accept an interface to be visually appealing.

Graphics are now an essential element of interface design.

Place alike elements together, watch spacing of the spacing of interface elements so that the screen does not appear too spread out.

Try to place graphics relevant to action on user interface.

Fonts, color and message style should be consistence

Window based Systems

Primary Window & Secondary Window

The primary window handles the major interaction with the user, and often contains an arbitrary number of objects.

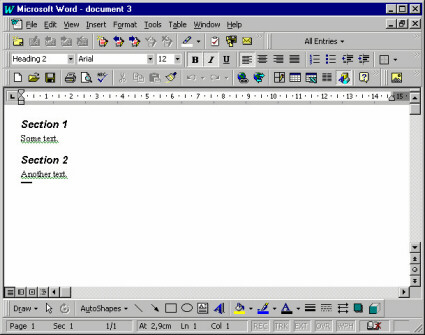
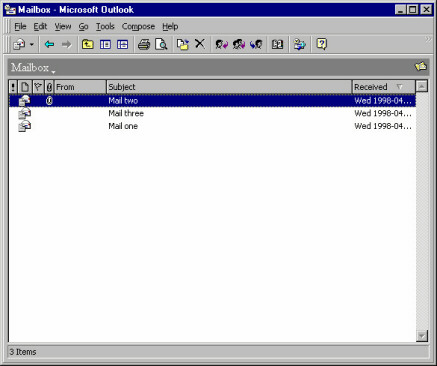
Secondary windows are used to support the interactions with primary windows by providing details about their objects and operations on those objects.

The user typically interacts with the system by first selecting one or several objects, for example by clicking on them, and then choosing an operation (for example, via a menu) that is executed on all the selected objects. Common operations are Cut, Copy, Paste, Delete, and View Properties.

The primary window normally contains a menu bar, from which users can choose operations. Users can also choose operations through pop-up menus (by pressing the right mouse button on the object itself) and by direct manipulation (by clicking and dragging the object).

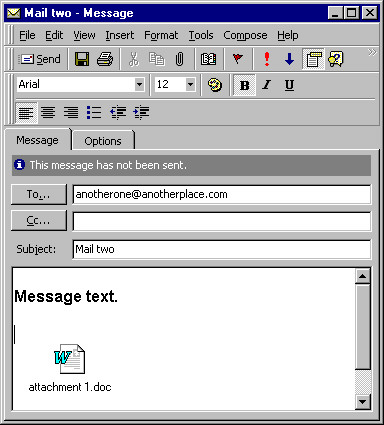
Since the total number of objects may not fit within the primary window, users can often scroll through the objects using a scroll bar, or resize the window.

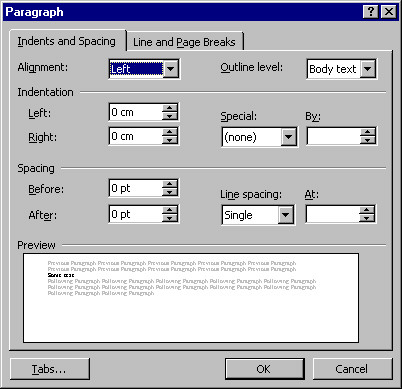
The primary window can often be divided into panes (defining sub-areas of the window), that the user can also resize.

(Primary window in Microsoft® Word™ 97, showing a document. It contains objects like paragraphs and characters. (Although the examples illustrated here are from the Microsoft® platform, these guidelines are by no means intended to be specific to that particular platform.)

(A primary window in Microsoft® Outlook, showing a mail box. It contains objects like mail messages.)

* A composite object in a user interface is an object that is visually composed of other objects. For example, a paragraph is a composite of characters, or a complex drawing object is a composite of more primitive drawing objects.
* Secondary windows support the primary windows by providing details (such as properties) about their objects, and operations on those objects.
* Only a few of the objects' properties are normally shown in the primary window.
* Properties of an object can be viewed by opening a property window (which is a secondary window) that shows all the attributes of an object.
* The user can often change the attributes by controls such as toggle and radio buttons, scales, combo boxes, and text fields.





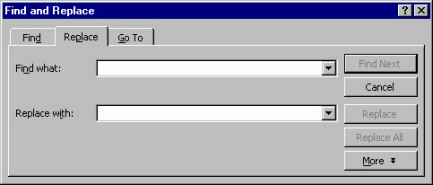
(A secondary window in Microsoft® Word™ 97, which is a property window showing the properties of a paragraph& A property window in Microsoft® Outlook, showing the properties of a mail message)

Note that there is a fine, and sometimes quite artificial, line between primary windows and secondary windows-they may display the same levels of complexity. For example, compare the document window shown above with the mail window: the document window is considered primary, whereas the mail window is considered secondary.

However, two main differences between primary and secondary windows are:

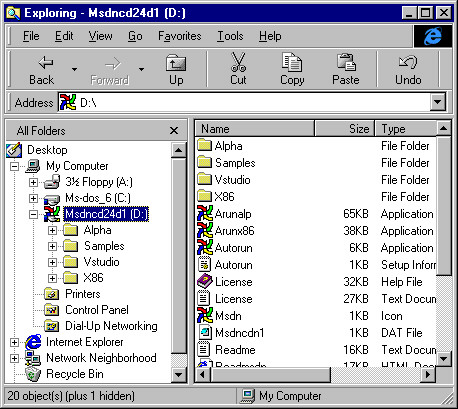
1. Primary windows are often considered to be more important to the application since they need to provide extensive usability. Therefore, development efforts tend to be more focused on the primary windows.
2. Secondary windows are often displayed by navigating through primary windows, and not vice versa.

In addition to property windows, there are other types of secondary windows, such as dialog boxes, message boxes, palettes, and pop-up windows.



(A dialog box in Microsoft® Word™ 97, providing a find operation among paragraphs and characters)

Many applications are filebased. Users can start these applications with the Open operation on a file object (for example, by double-clicking a file icon in a folder). Their primary window shows the objects stored in that file. Common operations on files are Save, Save As, Open, New, which can usually be selected through a file menu in the primary window. The primary window can also usually display multiple files (also called Multiple Document Interface, or MDI), thereby allowing the user to switch between different files.



(A file management window in Microsoft® Windows 95, showing files and folders.)

Visual Dimensions

The key to really usable primary windows is to use the visual dimensions when visualizing the contained objects and their attributes. The advantages of presenting more attributes than are necessary for identification are that:

* The user avoids window navigation overhead since you decrease the number of windows that must be shown (when the user needs to see an attribute that is presented in the primary window).
* The user can see the different aspects (of different objects) at the same time, which is often useful for comparisons and for starting to recognize patterns. A good use of the visual dimensions can encourage users to develop an entirely new fingertip feeling for their work.

The visual dimensions are:

* Position
* Size
* Shape
* Color

These dimensions are presented below. However, beware of the available screen area when designing the visualization of the objects. Try to make the overhead when exploiting the screen area as small as possible, and consider if using several visual dimensions is worth the extra expenditure of screen area. Maybe the user is better served by just a list of names, because what the user really needs is to see as many objects as possible.

Note that it is important to use these visual dimensions, or extend them, to be able to uniquely identify objects.

Also note that the visual dimensions can be used in correlation with the time dimension, for example by moving objects (their position is changed through time), or by changing the shape or color of objects (their state is changed through time).

Position

The most intuitive aspects that position can present are real-world positions. Examples are:

* Geographical Information Systems (GIS) that display a map on which you present the objects on the same longitude and latitude as they have in the real world;
* Computer Aided Design (CAD) programs that present the objects and their environment exactly according to their real-world coordinates;

"What You See Is What You Get" (WYSIWYG) editors that display the objects (characters) in the same location on the window as they will appear on a paper printout.

Sometimes it is relevant to show real-world size (the CAD-program and WYSIWYG editor examples), and sometimes it is not (for example, when the size of the objects is much smaller than the distance between the objects).

For example, imagine we have a flight booking system where the user must enter destinations. A possible presentation for this would be to display a map containing the different airports (where an airport is an object). Naturally, since the real-world sizes of the airports are irrelevant (as well as too small to be seen), all airports are shown as icons that are the same size.

This example also illustrates that real-world positions can be used even if they are not relevant, as long as they help the user to identify the objects. In the example, the user doesn't need to know the location of an airport. But, if the user is familiar with geography, it can be easier to find destinations on a map than in a list.

You can also use position to represent "virtual" real-world positions. For example, imagine a home shopping system where the users can buy things from different stores. A possible presentation for this would be to display a schematic picture of a (virtual) mall on which the different stores are positioned (where a store is an object). This schematic picture has nothing to do with the real locations of these stores-it only exploits the user's spatial memory: it is easier to remember an x-y position than it is to remember an item in a list or hierarchy.

Another alternative use for position is to show associations between objects: all objects that have the same vertical position are associated in one way, and all objects that have the same horizontal position are associated in another way. Spreadsheets are an example of this.

A similar alternative is to let one axis represent the value range of some attribute. For example, in a travel booking system, booked flights (where a flight is an object) could be presented along a horizontal time axis showing their relation in time, how long they will last, and the length of time the user will stay at each destination. These are all things that the user doesn't have to know, but they are nice to see if they can be presented unobtrusively.

If you don't want to use so much screen area by presenting the whole value range, you can collapse the distances between the objects. In the travel booking example, this would mean that all booked flights are laid out horizontally with no spaces in between, but the first flight is to the left, the second flight is immediately to the right of the first flight, and so on. Users wouldn't see the length of time they could stay at each destination, but they could see how long the flights would last.

Size

In many cases "size" must represent the same thing as position. In a CAD-system, for example, size must naturally represent real-world extent. Sometimes, however, we are free to choose what size should represent, for example the airports on the map that supported the destination selection.

In these cases, size should represent what is most intuitively perceived as the real-world size of the object. For a file, object size should represent amount of disk space occupied. For a bank account, object size should represent balance. For most sizes, a logarithmic scale is better than a proportional scale, since a proportional scale normally consumes too much screen area.

Size is actually so intuitive that you can consider showing it even if it is not relevant. After all, in the real world, different things (objects) occupy different proportions of our visual field because of their different size. And that is not obtrusive; it only helps us discriminate between the things. Similarly, using different sizes in the user interface will often help users discriminate between different objects.

Size should normally be used to present only one attribute, even though it would be possible to let horizontal extent present one attribute and vertical extent present another (which is rather non-intuitive, and might confuse the user).

Either horizontal extent or vertical extent should be (logarithmically) proportional to the attribute that size is to illustrate-the other extent should be fixed (or dependent on the length of the name, for example). If both horizontal and vertical extent is proportional to the same attribute, it seldom adds any value: it seems obtrusive and just consumes more screen area.

Shape

Icons in a graphical user interface normally represent shapes; shape is best used to represent type because it is more intuitive to map out a difference in looks than it is to map out a difference in type. In the real world, different objects of the same type of thing normally look similar, while objects of different types look different. For example, different objects of chair look similar (they all have four legs, a seat and a backrest), while a car looks very different from a chair.

So, what are the criteria for when different objects are of different types? Well, different classes should certainly be considered as different types. Also, some attributes are "type-like." These attributes must have a limited set of possible values and their value normally determines what can be done with the object (in terms of operations and possible values of other attributes). This is the same as in the real world-the most important difference between chair and car is how they are used: a chair is used for rest and a car is used for transportation.

However, when you analyze what should be considered different types, remember that the most important thing is: which attribute will the user most likely perceive as a type.

If you don't have multiple classes or any "type"-like attribute, you can use icons to represent the different values for some other limited-value attribute, but only if this attribute is of central interest to the user.

Icons are also often used to show different states of the object (in addition to showing the type). When you select an object, it is usually displayed in either of two ways: the color changes to black, or it displays a rectangle around it. Another possible state is that you have opened a property window for the object. Normally, you also have other application specific states that could be displayed, such as whether or not e-mail has been read. Just make sure that the presentation of state doesn't make it harder for the user to perceive the type and vice versa.

Color

Color can be divided into three components, based on visual perception. These are: hue (i.e., red, blue, brown, etc.), saturation, and darkness. However, you should not use different components to represent different attributes, since this will be too difficult for the user to perceive.

Hue could be used to represent type or attributes with a limited set of possible values. However, it is better to use an icon for this, because the icon can be designed so that the user understands what value it represents, while there is no such intuitive mapping out between color content and (most types of) values. Hue can thus be used instead of icons, if no intuitive icons can be found. An alternative if you have many type icons is to use hue for categorizing the type icons (so that some icons with a similar meaning are red, some with another meaning are blue, etc.).

Saturation could be used to represent an attribute with a value range, but this will lead to a rather ugly and obtrusive user interface-using different saturation is unsettling to the eye and using high saturation is rather obtrusive.

Darkness is the most usable component of color. It can be used to represent an attribute with a value range, and it is so unobtrusive that it can be used also for attributes of secondary importance. For darkness to be unobtrusive, you should not go from no darkness (white) to full darkness (black) but only from low darkness (light gray) to high darkness (dark gray). For many systems where the users create most of the objects, it is very useful to present objects according to age (e.g., the amount of time since the last change). This helps users identify the object they want to work with (which is often the object with the shortest "time since last change"). So, if you don't have a value-range attribute that you really need to present to the user, consider presenting age.

Often color is used to make the icons more esthetically appealing and that also helps the user quickly discriminate between the icons. If you provide multicolored icons, you should probably not use color for other purposes.

Since some people are color blind, and since not all screens support color, you should not use color as the only means of showing some vital information. On the other hand, a well-planned and non-obtrusive use of color makes the user interface more esthetically appealing.

Identification

The user must be able to uniquely identify each object. Sometimes the other visual dimensions are enough for identification, but most often they are not. Displaying a name within or close to the icon is the most popular technique for supporting identification. The advantage of names is that a very small screen area can display a large number of distinctly different names.

It is best if a name can be generated from an attribute value (that is normally textual). The alternative is to let users specify the names when they create the objects, but this takes some time, and thus reduces usability.

Sometimes you can shape the icon so that the name can be contained within the icon. This saves screen area and provides a stronger indication of the relation between the icon and the name. However, this can create the following problems:

* The icon has to be empty in the middle (where the name appears);
* Names have variable lengths, which means that either the icon's horizontal extent must depend on the length of the name, or that some names must be truncated;
* The icon must be much wider than it is high, since all text of reasonable length is longer than it is wide.

As a result, you often have to display the name below or to the right of the icon, which has the advantage that it consumes less screen area but the disadvantage that the object (icon + name) becomes even wider than it is high. If you don't have enough space to display the name at all (which is possible, because you can usually identify an icon without naming it), you can display the name through pop-up windows that display when the cursor is above the icon.

The font of the name can be used to display a limited-choice attribute, if you can find an intuitive mapping between font and attribute values (for example, you could use bold or italics to distinguish the object, or emphasize importance). In most cases, however, it is not appropriate to use the font, since it's rather obtrusive and seldom intuitive.

If you show the name (or, for that matter, any other text that the user is allowed to change), you should support editing the name directly in the primary window. The alternative would be for the user to request a rename-operation and then enter the new name, or to open the property window and edit the name there. Not only is it faster to edit the name directly in the primary window, but it also supports the principle "where you see it is where you change it."

Power Find and Select

If the group of objects that should be changed/operated on is composed so that the user can express selection criteria identifying them, the search tool of the primary window can solve the problem by always selecting all criteria matches.

There are two possible ways of managing the search:

* All objects to which the search criteria apply are selected in the primary window. If you cannot guarantee that all found objects are shown simultaneously in the primary window (because they may be too far apart), you can also display a hit list in the search window. After a search, the user either specifies additional search criteria or performs an operation on the selected objects. The advantage of this approach is that it enables the user to order some operation on all objects conforming to the search criteria.
* You provide a "Search" button in the search window that selects the next object conforming to the search criteria and scrolls the contents of the primary window so that this object is visible. After a search, the user can perform an operation on the selected object and then continue to search sequentially through the objects conforming to the search criteria. The advantage of this approach is that the user can see each found object in its surroundings (in the primary window rather than in a separate hit list).

In many cases, you will want to combine the two cases, for example by including a "Select All" button in the sequential search window or a "View Next" button in the parallel search window.

Sorting

An example of sorting may be that the system arranges all objects vertically, in alphabetical order by name or according to the value of an attribute. The user then browses the objects by scrolling. This is the simplest possible browsing support both with respect to implementation and to user operation. Sorting works best when the user always knows the name (or the attribute that we sorted according to) of the object that is wanted. An example of a system that should be implemented this way is a telephone book. The primary window should often have an operation for changing the sorting order and/or criteria.

User-Controlled Inheritance

An example of user-controlled inheritance is WYSIWYG-editors where you define what "style" each paragraph belongs to and then define how this style (i.e., every character belonging to this style) should be laid out.

A disadvantage compared to a search tool is that user-controlled inheritance supports only change of attributes (and possibly associations) for multiple objects, but not the performing of operations. Also user-controlled inheritance adds overhead in that the user must explicitly define and maintain the groups (that is, the available styles). It is also a more complicated concept.

However, if search criteria cannot be specified for the objects, or if the user needs to make relative changes to the attribute values (like increase by two), then providing user-controlled inheritance may be a solution.

For user-controlled inheritance to be useful, the nature of the class must be such that the objects can be categorized into groups (that have some logical meaning to the user) in which most of the attribute values are the same.

An advantage compared to a search tool is that user-controlled inheritance supports override (e.g., change the attribute value but only if it has not been explicitly defined in the object). Also user-controlled inheritance can enable the user to make more generic (and thus powerful) attribute value definitions (e.g., inherit the font from this style, but make it two pixels bigger). User-controlled inheritance is particularly useful when the groups have no easy-to-specify search criteria.

The class for which you will support user-controlled inheritance can either inherit itself or you can create a new class from which purpose is to be inherited. Making the class inherit itself is a little bit more powerful, since the same object can be used both to inherit from and to do the things originally intended for the object, like being an invoice, being an account, etc. This leads to fewer classes for the user (and the system) to manage. On the other hand, creating a new class to inherit from has the advantage of being easier to comprehend since inheritance is clearly separated from the normal operation of the class. Creating a new class is the best solution in most cases, especially if the users have not great experience with computers and object-oriented models. The new class you create should preferably inherit itself to support multiple levels of inheritance.

For most systems, the user often has to change the inheritance group for particular objects since the user does not know in advance exactly how the inheritance groups should be structured. Provide an operation for that.

If you decide to support user-controlled inheritance in your system, analyze what things (attributes, associations, class) need to be inherited and then support inheritance only for these things. This will lead to a less generic but easier way (for both users and developers) to manage functionality. Model those things that should be inherited in your new class. Many attributes will then be modeled both in the inheriting class and in the inherited class. Remember that user-controlled inheritance is meant to save time for the user, not for you. If the class inherits itself, this implies that everything is inheritable.

Decide if the user really needs to create new objects of the inherited class or if the system can provide a sufficient number of objects once and for all. Prohibiting the user from creating new objects will greatly decrease the flexibility of inheritance but on the other hand it will make it easier to operate.

Also decide if changes to numerical attributes in the inheriting objects should be interpreted as relative to the inherited value or as fixed. Say, for example, that an object inherits font size 12 and user changes it to 14. By relative interpretation, the system will remember the object's font size as inherited value +2; that is, if the font size of the inherited object changes the font size, the inheriting object will also change the font size. If you support relative interpretation, it should be noted on the attribute of the inherited object (because that's where you look when you want to examine inheritance). It is important that the relative interpretation is presented to the user (e.g., "font size: 12+2=14," rather than just "font size: 14"). You can explore with scenarios to find situations in favor of relative or fixed interpretation. You may have to support both.

Since user-controlled inheritance is only for intermediate and power-users, you must design it so that it will not interfere with normal use (e.g., when the user doesn't use inheritance); otherwise, novice users will be intimidated.

Remember that the user-controlled inheritance you construct is intended to make life easier for the user; it doesn't have to be generic or pure, but it has to be usable.

Browsing Hierarchies

A browsing hierarchy allows the user (or possibly the system) to categorize the objects into primary windows or composites, which are organized hierarchically. Browsing hierarchies ensures that the user only has to search one (or a few) categories. This reduces the number of objects that have to be displayed at a given point in time. A drawback is that the user (usually) has to manage the categorization. An example of this technique is file browsers: the reason for having directories or folders is to help the user find files.

Window Management

Window size and position is usually in complete user control. You can, however, consider reducing windowing overhead by letting the system influence size and position of windows.

The bigger a primary window is, the more objects can be shown, but the more screen area is also consumed. A primary window should normally show as many objects as possible but without unnecessary consumption of screen area.

Make each primary window big enough that all objects can be shown, but not bigger than the screen. Make each primary window big enough to show the whole objects but avoid areas that don't show anything useful like the margins in a desktop publisher. Even if you have space for showing these empty areas, they might obscure other applications.

Remember that a user resizes between sessions. If the number of objects increases, increase window size so much that all objects are visible, unless it is already full screen height or if the user has chosen a size that is smaller than the default. If the number of objects decreases, decrease the size, unless the user has chosen a size greater than the default. This rule ensures that you follow the intention of the user's resizing operations.

A possible further limitation on the size of a primary window is if you often need to use the application in parallel with other applications. Then you might maximize default size of the window to half screen (as opposed to full screen).

Make the default position of a primary window so that it obscures as little as possible of other applications. If you have to obscure some windows, chose those that have been unused for longest time, and try to leave at least a little bit of the windows visible so that the user can easily activate them.

A disadvantage with applying the rules above is that it will take some amount of control away from the user (the system will resize a window without being asked, and not remember user repositioning between sessions). Therefore, if you apply these rules, you should allow the user to switch them off (with a control).

For secondary windows, their size and position should be such that they don't obscure the window they were called from and possibly so that they don't obscure other secondary windows. If they must obscure the window they were called from, try to make sure that they don't obscure selected objects. Obscuring vital things, like selected objects, is a common usability flaw for secondary windows.

For primary windows other than the main primary window, you should also apply the sizing rule of the last paragraph.

Dialog boxes, however, should be placed so that they obscure the active window. Since they are normally temporary and small, the user usually doesn't need to see the active window while the dialog window is open. Placing dialog boxes over the active window makes sure that the user acknowledges them, and decreases necessary mouse movement since the cursor is normally already over the active window.

For property windows, the number of attributes determines the size. If the size is too big (approximately 1/4 of the screen), you should use more tabs.

Session Information

All application configurations should be saved between sessions (without the user having to specify it). The size and position of windows, which view is selected, and the positions of scroll bars should also be saved. When users restart an application, it should look exactly as when they exited it the last time. The motive for this is that usually the first thing users will do when starting a session is to work back to where they were when they exited the last session.

Online Help

On-line help is a very important part of the system. A well-designed help system should even be able to replace the user manuals for most systems. Most projects spend considerable efforts on constructing and producing manuals when it is a known fact that most users never use them. You should consider investing these efforts in a good help system instead.

There are a number of possible help tools you should consider:

Help-on-subject is the most important help tool. It lets the user enter a subject or browse an existing subject and provides help on these subjects. The key is to provide a large help index with lots of synonyms. Remember: the user may not know the correct term when needing help.

Help-on-object is context-sensitive help. It displays text that explains a specific part (object) of the user interface. The user requests context-sensitive help and then selects the part of the user interface where help is needed. This type of help should be supported for every part of the user interface, if it is to be usable. Another alternative is to provide implicit help in pop-up windows- a condensed form of context sensitive help that the system presents adjacent to the cursor when the user lingers for a few seconds. Using implicit help in pop-up windows has the advantage that it doesn't interfere with the normal operation of the user interface.

Message area is an area (usually in the main window) where the system prints unsolicited "comments" on the user's actions. It should be optional if provided.

Wizards is a popular technique you should consider providing when the user asks for help on how to do something. A wizard guides the user through a (non-trivial) task using a "hand-holding" technique. It shows descriptive text in conjunction with operations (buttons) that let the user carry out the parts of task explained in the text. Alternatively, a wizard will ask questions, and, based on the user's responses, automatically carry out the task. Wizards are excellent for tasks that are non-trivial and infrequently used.

The need for context-sensitive help and wizards is likely to be identified during use testing. If, during use testing, users don't understand what different portions of the user interface are, it is an indication to the need for context-sensitive help. If they have difficulties performing a certain task, it is an indication to the need for wizards.

The problem with many help systems is that they are either written for novices (spending an enormous amount of text explaining the obvious) or for experts (reference manuals that anticipate the user knows almost as much as the programmer who made the application). For most systems, most users are "improving intermediates." Write the help text for them.

Undo

Undo is a very useful feature, although it is hard to achieve (implement) in general. It enables users to learn faster, since they will not have to be afraid of destroying things. It also reduces the risk of losing information. An alternative solution for avoiding loss of information is to require that the user confirms all operations that might result in loss of information. This is usually a bad solution, however, since it adds considerable interaction overhead and the users soon learn to confirm unconsciously, thus rendering this solution inadequate.

An ambitious option is to also provide redo and possibly multiple levels of undo/redo. However, the first undo level achieves most of the increased usability.

Macro Agent

If you provide macros, it may be very useful to employ an agent that continuously monitors the user's actions, looking for repeated interaction sequences. As soon as a repeated interaction sequence is found, the agent creates a macro for it (after asking the user for permission). Let's say the user has ordered "Underline" for two text paragraphs and both times the user has also changed the text color to blue immediately after ordering "Underline." Then the agent should ask the user if the user wants a macro that does both "Underline" and "Set color to blue" for the selected text paragraph. If so, the agent should create such a macro and a push-button (or a menu item) that executes the macro.

If the user selects an object during recording, this should normally be interpreted as a "delta" specification, that is, what object has been selected in relation to previous selection (like "select next", "select first child," etc.).

Whether you should interpret the changing of an object's attributes as a delta specification (for example, interpreting the change of an attribute value from 12 to 14 as an increase by 2 rather than as a setting to 14) is not as obvious. Interpreting it as a delta specification is usually more powerful, since changing an attribute to a fixed value for multiple objects can often be accomplished by selecting multiple objects and then opening an attribute window for them, in which you set the attribute (to 14) once and for all.

Dynamic Highlighting

Quite often, associations between classes are bi-directional, meaning that in the real user interface, the association is shown on both objects. If a user, focusing on object A, can see that A is associated to object B, then the reverse is normally also interesting for the user (that is, when focusing on object B, the user can see that B is associated to A). The association is normally shown in the property windows of the objects, identifying the associated object by name.

In general, visualizing associations between objects in a primary window is tricky. Visualizing the associations as arrows or lines often leads to a rather unappealing and obtrusive "snake pit." A nice way of visualizing associations is to highlight all associated objects when the cursor is above an associating object. An example of this is when footnotes are associated with characters in a document editor, and the footnotes are highlighted when the cursor is above the associated character.

# Appendix 2: Guidelines for Research Projects

**1.0 Introduction**

This document aims at giving guidelines to students who are undertaking the research project for the B.Sc. /M.Sc. in Computer Science. Section 2 discusses the expected content for both the mid-term and final reports while Section 3 describes the due dates and some important notes.

**2.0** **Expected Contents of Report**

There are 2 reports to be produced throughout the duration of this research, a mid-term report and a final report. The mid term report is also referred to as a mini thesis because of the extension of research involved. This section will only highlight the expected contents of the report and thesis. For a description of the format of thesis writing, students should refer to the document "Guide to the Preparation of Master thesis " prepared by Project co-ordination Office.

**2.1 Mid-Term Report (Mini-Thesis)**

This report is your detailed proposal, which consists of the background information, related work and the proposed solution.

Suggested chapter outline:

Chapter 1 – Introduction

This will contain background information, which should mainly be relevant to later discussion of your work. Finish with a brief statement of the aim of the research you have undertaken.

Chapter 2 – Related Work

This consists of a critical review of the other research work, which is related to what you are doing. Problem Statement Declaration should be the output of this Chapter.

Chapter 3 – Proposed Design Methodology/Framework

Show how you intend to solve the problem.

Bibliography

**2.2 Final Thesis (Final Report)**

Chapter 1, 2, 3 as in the Mini-thesis (revised based on feedback received from your first seminar).

Chapter 4 – Implementation and Results

Present a discussion of the results that you obtained.

Chapter 5 – Summary and Future Work

Evaluate your findings/achievements and how this work can be further enhanced.

Bibliography

Both the above reports must have the following:

(a) Table of Contents

(b) Declaration Page

"This report is prepared as a partial fulfillment towards graduation requirements for Masters of Science in Computer Science, Division of Science & Technology, University of Education, Lahore. This report and all the products of the project (source codes, system/application, user manual etc.) are the copyright of Division of Science & Technology, University of Education, Lahore. No part of this report and project shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without permission from Division of Science & Technology, University of Education, Lahore."

(e) Abstract

Indicate concisely what you have done, your methodology and the important conclusion that you have drawn.

3 copies of each report (for supervisor, examiner and office) must be handed in on the due date.

# Appendix 3: Final Documentation Format Guidelines

**Typographical Format and Binding**

**Page Format:**

Page size: A4

Top margin: 1.00 inch

Bottom margin: 1.00 inch

Left margin: 1.25 inch

Right margin: 1.00 inch

Page numbering: Bottom right - part of the footnote

Title page not numbered

All other pages before the page of chapter one numbered in lower roman numerals (i, ii, iii, …)

All other pages starting from first page of chapter one to last page of the report numbered in integers (1, 2, 3, …)

Footer: Each page shall have a footnote “Division of Science & Technology, University of Education, Lahore”

Left aligned

In case of long titles shorter versions should be used.

There shall be a line over the footnote.

Header: Each page shall have a header “Project Name”

Left aligned

In case of long titles shorter versions should be used.

There shall be a line under the footnote.

Chapter Startup: Each chapter shall be numbered as Chapter 1, Chapter 2, etc. The name of the chapter shall be written immediately below. Both shall be centered horizontally as well as vertically.

The actual chapter content shall start from the next page.7

Text: Only one side of the paper shall be used.

The other side shall be blank.

When a report is opened the right side would contain text, figures, or tables and the left side would be blank.

Tables and Figures: Tables and figures shall be placed on one side only

Separate pages shall be used for figures and tables.

One page may contain more than one figure or table but text will not be combined or interlaced with figure or table.

Each table / figure shall be numbered.

For example "Table 1.2: Population distribution in Asia" or "Figure 3.2: Temperature distribution"

The table number or figure number shall be placed as normal text centered at the bottom of the table or figure or sideways with table / figure title coming on the opening side of the paper and note on the binding side.

**Paragraph:**

Single-spaced.

Line entered paragraph.

DONOT put indents at the beginning of the paragraph.

Left aligned or justified.

**Text Format**

Normal and plane text:

Font Type: Times New Roman

Font Size: 12

Headings:

Chapter Heading: Times New Roman Bold Size 16 Title Case normal

Heading 1: Times New Roman Bold Size 14 Title Case normal

Heading 2: Times New Roman Bold Size 12 Title Case normal

Heading 3: Times New Roman Bold Size 12 Title Case italic

**Sections and Subsections**

In case of sections and subsections follow this format:

1 Section

1.1 Sub Section

1.1.1 Nested Sub Section

a

b

i

ii

The subsequent reference to a any section shall be made using the section and its number. For example **section 2.1.3** means chapter 2 section 1 subsection 3.

**Mathematical Equations**

The following numbering scheme should be used to number the equations:

f(x) = x+3 (XX:YY)

Where XX is the chapter number and YY is the sequence number of that equation in that chapter.

If an equation is previously quoted in an earlier chapter, say as equation 4:5 and need to be re-quoted in chapter 5, its number will remain as equation 4:5.

**References**

References are to be placed in square brackets and interlaced in the text. For example "A comprehensive detail of how to prevent accidents and losses caused by technology can be found in the literature [1]. A project report / thesis cannot be accepted without proper references. The references shall be quoted in the following format:

The articles from journals, books, and magazines are written as:

[1] Abe, M., S. Nakamura, K. Shikano, and H. Kuwabara. Voice conversion through vector quantization. *Journal of the Acoustical Society of Japan*, April 1990, E-11 pp 71-76.

[2] Hermansky, H. Perceptual linear predictive (PLP) analysis for speech. *Journal of the Acoustical Society of America*, January 1990, pp 1738-1752.

The books are written as:

[1] Nancy G. Leveson, Safeware System Safety and Computers, A guide to preventing accidents and losses caused by technology, Addison-Wesley Publishing Company, Inc. America, 1995.

[2] Richard R. Brooks, S. S. Iyengar, *Multi-Sensor Fusion Fundamentals and Applications with Software*, The Prentice-Hall Inc. London, 1998.

**Binding**

All reports shall be bounded with an appropriate print on the backbone.

Two copies should be submitted.

**Color of the binding:**

* BSc project / thesis reports: black
* MSc project / thesis reports: blue

Contents of the CD Attached

All reports / theses must accompany a CD whose contents will have the following:

Top-level directories:

Doc All documents related to the project

Instructions how to access the CD to the point to running the project

All reports already submitted

The final project report in thesis form

Installation instructions

Trouble shooting instructions in case of problems

User manual

Research material including URLs

Papers consulted / referred to

Slides of the presentations

Source All source files that will be needed to compile the project.

Further subdirectories can be used.

This must include sample data files as well.

Project The running project including sample data files as well as sample output.

This should be in a form that if copied to a machine runs without errors.

This may an exe file of an entire project, an installer depending on the project or simply a running project.

You can have sub directories with appropriate names.

Length

The length of your dissertation depends on the type of project you have selected. An excellent dissertation will often be brief but effective (its author will have said a lot in a small amount of space). Voluminous data can be submitted electronically on CD.

# Appendix 4:Final Documentation Intial Pages

STUDY ON DIGITAL SUBSCRIBER LINE



Session : BSCS Spring 2008-2012

**Project Advisor: Dr. M. A. Pasha**

Submitted By

|  |  |
| --- | --- |
| Imran Arshad | BSCSS09A041 |
| Asif Khan | BSCSS 09A104 |
|  |  |
|  |  |

Department of Computer Science & Information Technology

The Govt. Sadiq College Women University Bahawalpur

STATEMENT OF SUBMISSION

This is certify that Imran Arshad Roll No. BSCSS09A041 and Asif KhanRoll No. BSCSS 09A104 successfully completed the final project named as: **Online Banking System**, at the Department of Computer Science & Information Technology, The Govt. Sadiq College Women University Bahawalpur, to fulfill the requirement of the degree of **BS in Computer Science**.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Project Supervisor Project Coordination Office

Saeed Ahamed DOCS&IT -GSCWU

Lecturer

DOCS&IT -GSCWU

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

External Examiner Dr. M. Anwar Ur Rehman Pasha

Khurram Shehzad Chairman

System Analyst DOCS&IT -GSCWU.

Net sol, Lahore

Acknowledgement

We truly acknowledge the cooperation and help make by Dr. M A Pasha, Chairman, Department of Computer Science & Information Technology, The Govt. Sadiq College Women University Bahawalpur, Sargodha. He has been a constant source of guidance throughout the course of this project. We would also like to thank Mr. Muammer Qureshi from National Telecommunication Corporation, Lahore for his help and guidance throughout this project. We are also thankful to our friends and families whose silent support led us to complete our project.

1- M. Aslam

2- M. Akram

Date: September 26, 2012

**Abstract**

Due to ever increasing demand of transporting huge amount of information generated from various sources such as voice, data, video, etc., modern telecommunication networks have been transformed into all digital and broadband. Depending on the characteristics of information sources and the availability of facility, the mode of transportation can be either constant bit rate (CBR) using circuit switched networks or variable bit rate (VBR) using packet switched networks. For efficient utilization of the network, all kinds of information can be transported using BISDN (Broadband Integrated Services Digital Network) and ATM (Asynchronous Transfer Mode) technology. One important research area in Network Technology is the design of high-speed digital network with good performance. The issues need to be investigated include modeling of Variable Bit Rate video traffic, efficient assignment of different traffic classes with diverse quality of services, optimal bandwidth allocation, routing and call admission control etc. This project not only relates to study of Digital Subscriber Line, which is a Broadband technology to provide high-speed data, voice and video but also addresses the above-mentioned issues. What are the provisions made in DSL implement QoS quality of service.

# Appendix 5: Final Evaluation Matrix for Development Projects

Department of CS & IT, The Govt. Sadiq College Women University Bahawalpur

(Software Evaluation Metrics)

Project ID:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Title of the Project:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dated:\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Students ID’s** |  |  |  |  |  | **Remarks** |
| Individual Evaluation | Please award marks for individual students between 1(low) – 10(High) | | | | | |
| 1) Presenter Appearance |  |  |  |  |  |  |
| 2) Presentation Style |  |  |  |  |  |  |
| 3) Communication Skill |  |  |  |  |  |  |
| 4) Answer to the Questions |  |  |  |  |  |  |
| 5) Individual Contribution |  |  |  |  |  |  |
| 6) Knowledge about the Project |  |  |  |  |  |  |
| 7) Knowledge about the limitation of the software |  |  |  |  |  |  |
| Individual Total | | | | | | |
| Standard of Deliverables Group Remarks | | | | | | |
| 1) Software Requirement Specification |  |  |  |  |  |  |
| 2) Software Design Modeling |  |  |  |  |  |
| 3) Software Architecture Modeling |  |  |  |  |  |
| 4) Detailed Design |  |  |  |  |  |
| 5) Implementation of Modeling Technique |  |  |  |  |  |
| 6) Quality of the Text |  |  |  |  |  |
| Software Deliverables Total | | | | | | |
| Overall Software as Group Group Remarks | | | | | | |
| 1) Software Complexity |  |  |  |  |  |  |
| 2) Fulfillment of the Proposal |  |  |  |  |  |
| 3) Programming Technique Used |  |  |  |  |  |
| 4) User interface Design Standards |  |  |  |  |  |
| 5) Software Performance |  |  |  |  |  |
| 6) Implementation of the Project |  |  |  |  |  |
| 7) Integrated form of the software |  |  |  |  |  |
| Overall Software Total |  |  |  |  |  |

# 

# Appendix 6: Final Evaluation Matrix for Research Projects

Project ID:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Title of the Project:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dated:\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Student’s ID** |  |  |  |  |  | **Remarks** |
| Individual Evaluation | Please award marks for individual students between 1(low) – 10(High) | | | | | |
| 1. Presenter Appearance |  |  |  |  |  |  |
| 1. Presentation Style |  |  |  |  |  |  |
| 1. Communication Skill |  |  |  |  |  |  |
| 1. Answer to the Questions |  |  |  |  |  |  |
| 1. Individual Contribution |  |  |  |  |  |  |
| 1. Knowledge about the Project |  |  |  |  |  |  |
| 1. Knowledge about the limitation of the Project |  |  |  |  |  |  |
| Individual Total | | | | | | |
| Standard of Deliverables Group Remarks | | | | | | |
| 1. Research Scope & Feasibility Report |  | | | |  | |
| 1. Research Preliminary Investigation Report |  | | | |
| 1. Validity of Review of Literature |  | | | |
| 1. Soundness of Problem Definition |  | | | |
| 1. Correctness and Validity of Proof Methodology |  | | | |
| 1. Soundness and Correctness of Research Outcomes |  | | | |
| 1. Strength and Validity of References |  | | | |
| 1. Margin of Future Research Enhancement |  | | | |
| 1. Overall Complexity of the Research |  | | | |
| 1. Quality of the Text |  | | | |
| Overall Total |  | | | |

# Appendix 7: Final Evaluation Matrix for Network Projects

Title of the Project:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Project ID:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dated:\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Student’s ID** |  |  |  |  |  | **Remarks** |
| Individual Evaluation | Please award marks for individual students between 1(low) – 10(High) | | | | | |
| 1. Presenter Appearance |  |  |  |  |  |  |
| 1. Presentation Style |  |  |  |  |  |  |
| 1. Communication Skill |  |  |  |  |  |  |
| 1. Answer to the Questions |  |  |  |  |  |  |
| 1. Individual Contribution |  |  |  |  |  |  |
| 1. Knowledge about the Project |  |  |  |  |  |  |
| 1. Knowledge about the Limitation of the Project |  |  |  |  |  |  |
| Individual Total | | | | | | |
| Standard of Deliverables (Group Remarks) | | | | | | |
| 1. Network Scope & Feasibility Report |  | | | |  | |
| 1. Network Solution Architecture |  | | | |
| 1. Network Technical Aspects (Protocol, etc.) Report |  | | | |
| 1. Network Design & Implementation details |  | | | |
| 1. Network Security/Performance Issues Report |  | | | |
| 1. Quality of the Text |  | | | |
| Software Deliverables Total | | | | | | |
| Overall Network Implementation as Group (Group Remarks) | | | | | | |
| 1. Network Complexity |  | | | |  | |
| 1. Fulfillment of the Proposal |  | | | |
| 1. Programming Scripts utilities for Networks |  | | | |
| 1. User Interface/Protocols, Design Standards |  | | | |
| 1. Network Performance |  | | | |
| 1. Implementation of the Project |  | | | |
| 1. Tools and Technologies used (Fiber optics, Radio Links, etc.) |  | | | |
| Overall Total |  | | | |

# Appendix 8: Final Evaluation Metric

**Metrics:**

Individual Evaluation: ∑ ((Individual i) x (Weight i)

Deliverables Evaluation: ∑ ((Deliverables i) x (Weight i)

Individual Evaluation: ∑ ((Overall Software i) x (Weight i)

**Success Criteria:**

**Individual Evaluation:**

A-Grade 80%

B-Grade 70%

C-Grade 60%

Repeat 50%

**Fail bellow then 50%**

**Deliverables Evaluation:**

A-Grade 80%

B-Grade 70%

C-Grade 60%

Repeat 50%

**Fail bellow than 50%**

**Overall Project Evaluation:**

A-Grade 80%

B-Grade 70%

C-Grade 60%

Repeat 50%

**Fail bellow than 50%**

**Total % Marks in Project:**

#### Marks = (Individual %age + Deliverables %age + overall Project %age) / 3

**Note:**

* If any person is fail in the individual Evaluation will be considered Fail in overall project
* If as Group Deliverables/ overall Project Evaluation is less than 60% then individual Evaluation will not be considered.
* Group will repeat the any activity in case of failure

# Appendix 9: Project Registration Form

FINAL PROJECT REGISTRATION FORM

**Degree** \_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Session** \_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Project Title:** | | | | | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | | | | | | | | | | |
| **External Supervisor Information:** | | | | | | | | | | | |  | | | | | | |
|  | | Name: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | | | | | | | | | | | | |
|  | | Address: | | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | | | | | | | | | | | |
|  | | Designation: | | | | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | | | | Email: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | |
|  | | Ph. No. Res. | | | | | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | Office | | | \_\_\_\_\_\_\_\_\_\_\_\_ | | Mobile: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | |
| **Group Members:** | | | | | | | | | |  | | |  | | | | | | |
|  |  | | Roll Number | | | | | | Name | | | | | Email | | | Signatures | | |
|  | 1. | |  | | | | | |  | | | | |  | | |  | | |
|  | 2. | |  | | | | | |  | | | | |  | | |  | | |
|  | 3. | |  | | | | | |  | | | | |  | | |  | | |
|  | 4. | |  | | | | | |  | | | | |  | | |  | | |
|  | 5. | |  | | | | | |  | | | | |  | | |  | | |
|  | 6. | |  | | | | | |  | | | | |  | | |  | | |
| Supervisor Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | | | | | | | | | | | | | | | | | | |

-----------------------------------------------------------------------------------------------------------------------

(For Official Use Only)

Recommended

Meeting Required: Date: \_\_\_\_\_\_\_\_\_\_\_ Time: \_\_\_\_\_\_\_\_\_\_\_ Place: \_\_\_\_\_\_\_\_\_\_\_

Rejected

|  |
| --- |
| Remarks: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Project Title (Revised): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Recommended / Not Recommended Final Project :\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Approved / Not Approved Group ID: \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Start Date: \_\_\_\_\_\_\_\_\_\_

Project Manager

Appendix 10: Evaluation Request Form

**Project Information**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Project ID |  | | | | | | | |
| Project Name |  | | | | | | | |
| Team Size |  | | | | | | | |
| Project Category | Research |  | Java Platform |  | Microsoft Platform |  | Network |  |
| Student Note / Request |  | | | | | | | |

**Required Hardware Resources**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **Resources/ Equipment** | **Required** | **Number** |
| 1 | LAN Cards |  |  |
| 2 | Modems |  |  |
| 3 | Telephone Lines |  |  |
| 4 | Telephone Sets |  |  |
| 5 | Speakers |  |  |
| 6 | Others | | |

**Time Required for System Configuration:** Minutes

**Special Requirements** (e.g. Networking projects in LAB)

* Lab
* Room

**Note:** Following guidelines should be followed to fulfill this form.

1. If your project/ software is related to lab environment then tick Lab otherwise Room option
2. It is recommended that you bring your personal computers (System Unit(s)) but if you required clearly mention the number of computers required

# Appendix 11: Evaluation Delay Request Form

**Evaluation Delay Request Form**

**Project Information**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Project ID |  | | | | | Dated: | | | |
| Project Name |  | | | | | | | | |
| Personal-ID |  | | | | | | | | |
| Project Category | Research |  | Java Platform |  | Microsoft Platform | |  | Network |  |
| Subject: |  | | | | | | | | |
|  |

**Extension for**

|  |  |
| --- | --- |
| Proposal | Evaluation |
| 1st Deliverable |
| 2nd Deliverable | Submission |
| 3rd Deliverable |
| Final | Inspection |
| Other |

**Reason for Extension**

|  |
| --- |
|  |

**Students Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_,\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_,\_\_\_\_\_\_\_\_\_\_\_\_**

Project Office Use Only

**Remarks:**

|  |
| --- |
|  |

Name of Project Manager: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Decision:**

|  |
| --- |
|  |

# 

# Appendix 12: Show Cause Form

**Show-Cause Form**

**Project Information**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Project ID |  | | | | | Dated: | | | |
| Project Name |  | | | | | | | | |
| Personal-ID |  | | | | | | | | |
| Project Category | Research |  | Java Platform |  | Microsoft Platform | |  | Network |  |
| Subject: |  | | | | | | | | |
| Notice Number |  | | | | | | | | |

**Notice Abstract**

|  |
| --- |
|  |

**Personal/ Group Statement**

|  |
| --- |
|  |

**Students Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_,\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_,\_\_\_\_\_\_\_\_\_\_\_\_**

Project Office Use Only

**Remarks:**

|  |
| --- |
|  |

Name of Project Manager: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Decision:**

|  |
| --- |
|  |

# Appendix 13: Project Renrollemnt Form

**Project Re-Enrollment Form**

**Team Information**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Level | BS MSc | **Dated:** | | **Form Number** | |
| **Student – ID** | **Student Name** | **E-Mail Address** | | **Signature** | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
| Project Category | Research | Java  Platform | Microsoft  Platform | | Network |
| Old Project ID |  | Old Evaluation Comments |  | | |

**New Project Abstract /Extension**

|  |
| --- |
|  |

**Tools / Technologies**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Project Office Use Only

**Remarks:**

|  |
| --- |
|  |

**Signature:**

Accounts:-------------------------------------- Academic Coordinator:-------------------------------------

|  |
| --- |
| Name of Project Manager: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **Decision:** |

# Appendix 14: Change Request Form

**Change Request Form**

**Project Information**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Project ID |  | | | | Dated: | | | | |
| Project Name |  | | | | | | | | |
| Team Size |  | | | | Change Risks to : Project Project Team | | | | |
| Project Category | Research |  | Java Platform |  | | Microsoft Platform |  | Network |  |
| Subject for Change |  | | | | | | | | |
| Reason (Abstract) |  | | | | | | | | |

This portion is to be filled by the Students

|  |  |  |  |
| --- | --- | --- | --- |
| **Change Description** | Category  (New / Change / Withdrawal) | Priority  (Must be done has high value) | Status  (Decision by Project office) |
|  |  |  |  |

Students Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***This portion is to be filled by Project Coordination Office***

Analysis:

|  |  |  |
| --- | --- | --- |
| Effected Documents | | |
| Document Name | Change Required | Chang Description |
| Proposal  Analysis Deliverables  Design Deliverable  Detailed Design  Deliverable  Implementation |  |  |

|  |  |
| --- | --- |
| Effected Modules | |
| Module name | Description |
|  |  |

|  |  |
| --- | --- |
| Effected GUIs | |
| GUI ID | Description |
|  |  |

**Detailed comments about change effects:**

**Estimated time:**

**Delivery Date:**

**Comments/Suggestions:**