

This should be the first inner page of your document.

Title Of Final Project



Project ID: get this ID from the projects convener

Project Advisor: Mr/Ms. Name of Project Adviser

Submitted By

Name of Student
Name of Student

ID of Student, Session
ID of Student, Session

University of South Asia
Department of Computer Science

This should be the second inner page of your document.

Statement of acceptance

This is to certify that these students have submitted their final year project, which the department of computer science has accepted and evaluated. The final year project completion is one of the mandatory requirements to be fulfilled by the students towards the conclusion of their degree of

BS in Computer Science (BSCS).

Project ID: get this ID from the projects convener

Project Title

Name of Student

ID of Student

Name of Student

ID of Student

<write adviser/supervisor's name>

Project Adviser

Department of Computer Science

Masroor Hussain

Projects' Convener

Department of Computer Science

Dr. Rahat Hussain Bokhari

Dean Department of Computer Science.

University of South Asia.

This should be the third inner page of your document.

Proofreading Certificate

It is to certify that I have read the document meticulously and circumspectly. I am convinced that the resultant project does not contain any spelling, punctuation or grammatical mistakes. As much as expected from the graduating students at the bachelors' level. All in all I find this document well organized.

**XYZ, <name of proofreader resource>
Lecturer/Instructor, University of South Asia.**

Acknowledgement

Most of all we thank Allah our creator and Sustainer. Above all we thank Allah our Creator and Sustainer, for enabling and letting us to be what we are now. We truly acknowledge the cooperation and help given by our **Adviser, Designation and Address of Organization**. He has been a constant source of guidance throughout the course of this project. We would also like to thank **Sir/Mam <name> Designation, Name/Address of Organization** for their 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- Student name and roll number

2- Student name and roll number

Date:

April 1, 2019

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.

Next few pages give guidelines on typographic format and binding information.

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 “University of South Asia.” 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.
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:

$$f(x) = x+3 \quad (XX:YY)$$

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.

The Internet links shall be complete URLs to the final article.

- [1] <http://www.pu.edu.pk/ucit/projects/seminars.html>

Binding

All reports shall be bounded with an appropriate print on the backbone. Two copies should be submitted.

Color of the binding:

- BS project / thesis reports: Black
- MS project / thesis reports: Blue

Contents of the CD/DVD Attached

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

Top-level directories:

Doc	<p>All documents related to the project</p> <ul style="list-style-type: none">Instructions how to access the CD to the point to running the projectAll reports already submittedThe final project report in thesis formInstallation instructionsTrouble shooting instructions in case of problemsUser manualResearch material including URLsPapers consulted / referred toSlides of the presentations
Source	<p>All source files that will be needed to compile the project.</p> <p>Further subdirectories can be used.</p> <p>This must include sample data files as well.</p>
Project	<p>The running project including sample data files as well as sample output.</p> <p>This should be in a form that if copied to a machine runs without errors.</p> <p>This may an exe file of an entire project, an installer depending on the project or simply a running project.</p>

You can have sub directories with appropriate names.

Length

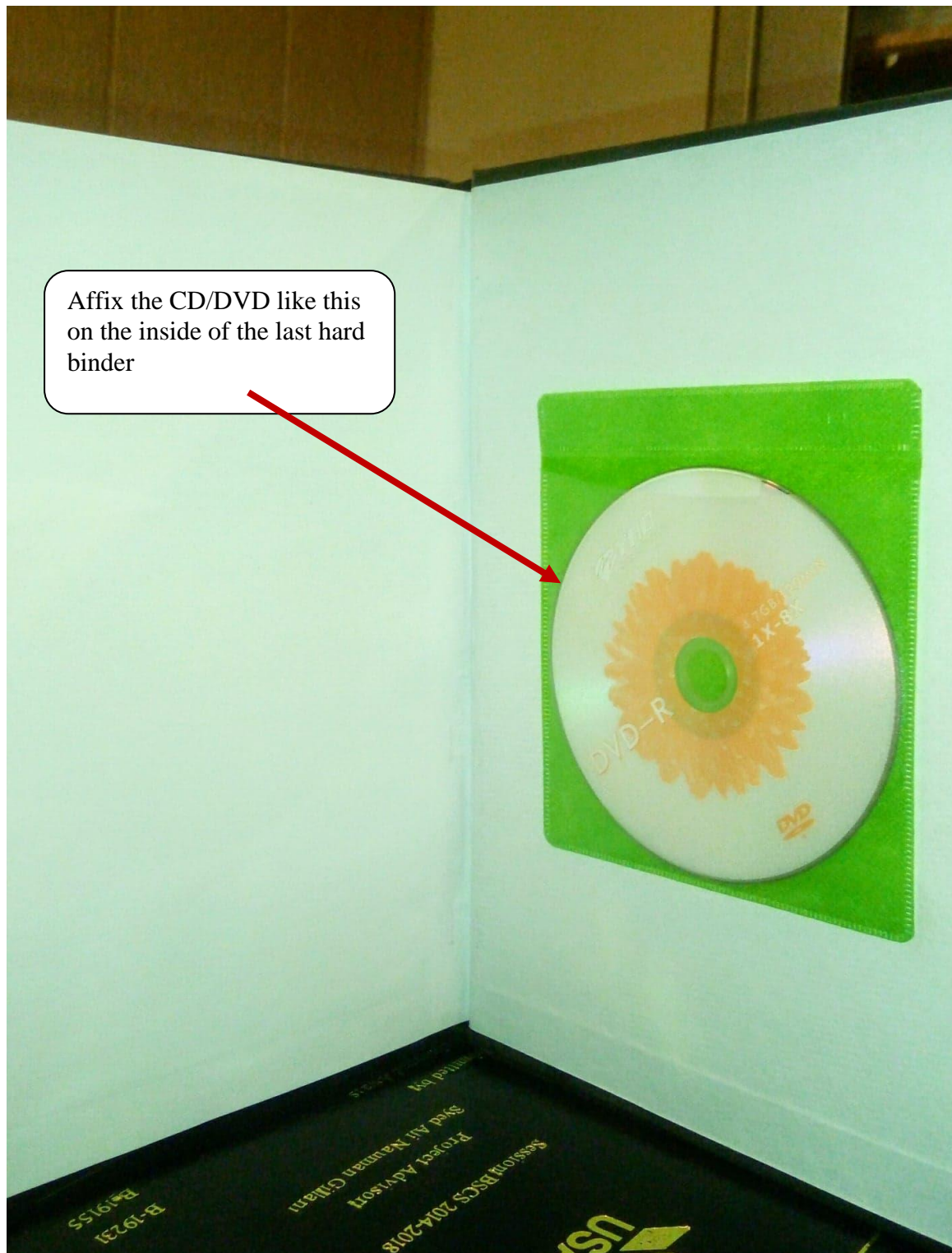
The length of your dissertation depends on the type of project you have selected. An excellent dissertation will often be brief but effective (its author will have said a lot in a small amount of space). Voluminous data can be submitted electronically on CD.

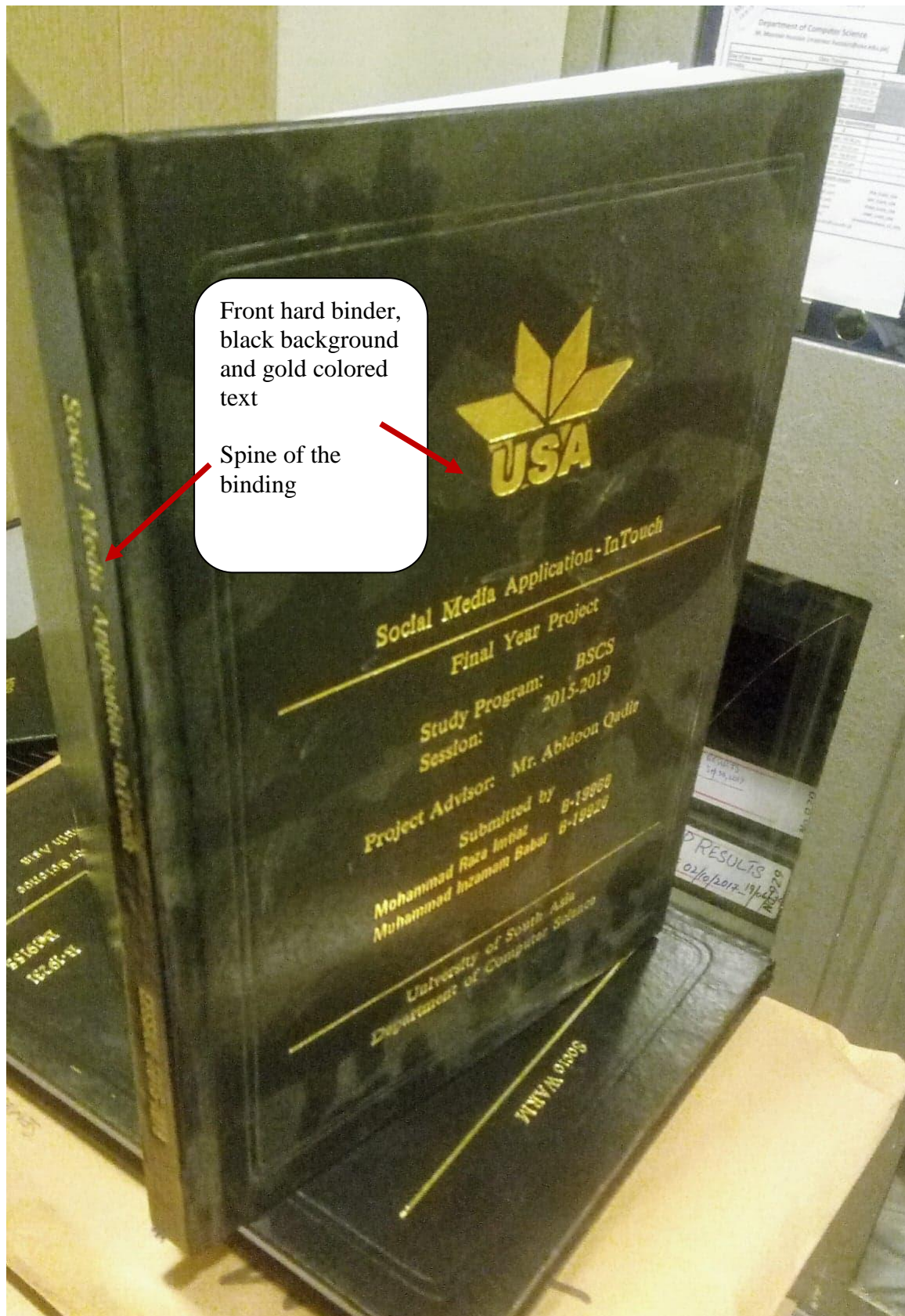
The hardcover binding should be black in color, and all the text on it should be printed in gold color.

The CD/DVD is to be attached on the inside of the last hard cover.

Do not attach along with the plastic hard-container.

Put the CD/DVD inside a thin envelope to attach using glue or scotch tape.





**IN THE FOLLOWING PAGES A FEW SAMPLE
CHAPTERS FOR YOUR DOCUMENT HAVE BEEN
SUGGESTED.**

TABLE OF CONTENTS

FIRST DELIVERABLE GUIDEERROR! BOOKMARK NOT DEFINED.

1 Introduction.....	15
1.1 Project/Product Feasibility Report.....	15
1.1.1 Technical Feasibility.....	15
1.1.2 Operational Feasibility.....	15
1.1.3 Economic Feasibility.....	15
1.1.4 Schedule Feasibility.....	16
1.1.5 Specification Feasibility.....	16
1.1.6 Information Feasibility.....	16
1.1.7 Motivational Feasibility.....	16
1.1.8 Legal & Ethical Feasibility.....	16
1.2 Project/Product Scope.....	16
1.3 Project/Product Costing.....	17
1.3.1 Project Cost Estimation By Function Point Analysis.....	17
1.3.2 Project Cost Estimation by using COCOMO'81 (Constructive Cost Model).....	20
1.4 CPM - Critical Path Method.....	21
1.5 Gantt chart.....	25
1.6 Introduction to Team member and their skill set.....	25
1.7 Tools and Technology with reasoning.....	25
1.8 Vision Document.....	25
1.9 Risk List.....	26
1 Introduction.....	27
1.1 Systems Specifications.....	28
1.2 Identifying External Entities.....	29
1.3 Context Level Data Flow Diagram.....	29
1.4 Capture "shall" Statements.....	29
1.5 Allocate Requirements.....	29
1.6 Prioritize Requirements.....	30
1.7 Requirements Trace-ability Matrix.....	30
1.8 Example.....	30
1.8.1 Introduction.....	30
1.8.2 Existing System Business Organization.....	30
1.8.4 Scope of the System.....	31
1.8.5 Summary of Requirements:(Initial Requirements).....	32
1.8.6 Identifying External Entities.....	33
1.8.9 Capture "shall" Statements and External Entities.....	35
1.8.10 Allocate Requirements.....	35
1.8.11 Priorities Requirements.....	37
1.8.12 Requirements Traceability Matrix.....	40
1.9 High Level Usecase Diagram.....	41

1 Introduction

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

- a. Project Feasibility
- b. Project Scope
- c. Project Costing
- d. Critical Path Method Analysis (CPM Analysis)
- e. Gantt Chart
- f. Introduction to team members
- g. Tools and Technologies
- h. Vision Document
- i. Risk List

1.1 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

1.1.1 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.

1.1.2 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.

1.1.3 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.

1.1.4 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.

1.1.5 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.

1.1.6 Information Feasibility

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

1.1.7 Motivational Feasibility

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

1.1.8 Legal & Ethical Feasibility

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

1.2 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.

1.3 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.

Cost estimation can be done by just one methodology.

1.3.1 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.

Type of Component	Complexity of Components			
	Low	Average	High	Total
External Inputs	x 3 =	x 4 =	x 6 =	
External Outputs	x 4 =	x 5 =	x 7 =	
External Inquiries	x 3 =	x 4 =	x 6 =	
Internal Logical Files	x 7 =	x 10 =	x 15 =	
External Interface Files	x 5 =	x 7 =	x 10 =	
Total Number of Unadjusted Function Points				
Multiplied Value Adjustment Factor				
Total Adjusted Function Points				

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 as 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:

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

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:

$$\text{Cost} / \text{FP} = \text{labor rate} / \text{productivity parameter}$$

$$\text{Total Project Cost} = \text{FP est.} * (\text{cost} / \text{FP})$$

$$\text{Total Estimated Effort} = \text{FP est.} / \text{productivity parameter}$$

1.3.2 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

T_{dev}= 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.

$$\text{People Required} = \text{Effort} / \text{Duration}$$

1.4 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.

$$ES(K) = \max [EF(J) : J \text{ is an immediate predecessor of } K]$$

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.

$$LF(K) = \min [LS(J) : J \text{ is a successor of } K]$$

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.

$$FS(K) = \min [ES(J) : J \text{ is successor of } K] - EF(K)$$

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.

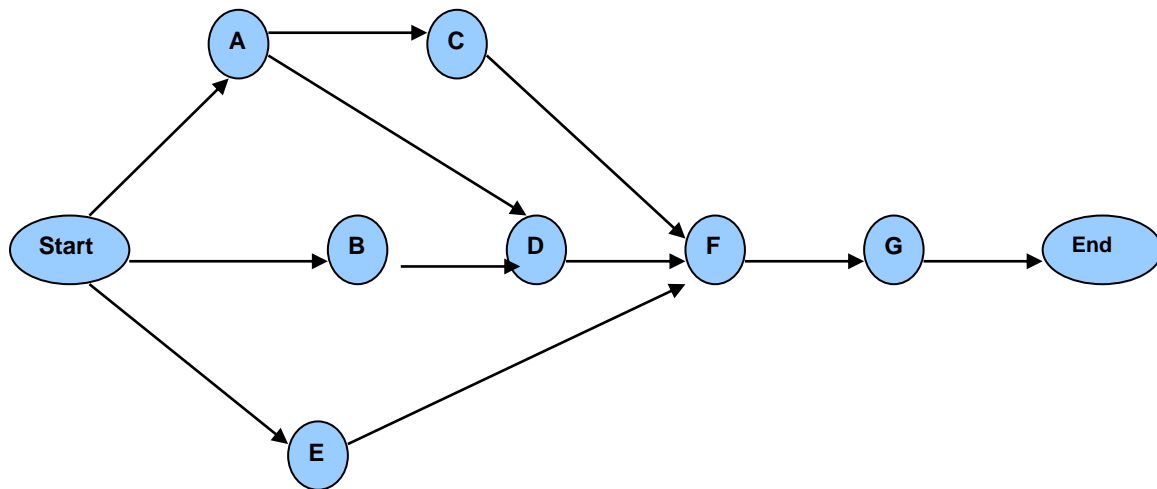
The critical path is the path through the project network in which none of the activities have slack, that is, the path for which $ES=LS$ and $EF=LF$ for all activities in the path. A delay in the critical path delays the project. Similarly, to accelerate the project it is necessary to reduce the total 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	22	0	0

The parameters and slacks are calculated as follows:

The critical path is:

A-> C-> F-> G, and Project Duration is _____ months

[Note: The project duration should be more than CoCoMo T_{dev}]

1.5 Gantt chart

The Gantt chart enumerates the activities to be performed on the vertical axis and their corresponding duration on the horizontal axis. The tasks identified and enlisted are based on task dependency table. 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.

Use MS Project to develop gantt chart.

1.6 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.

1.7 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

1.8 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?

1.9 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

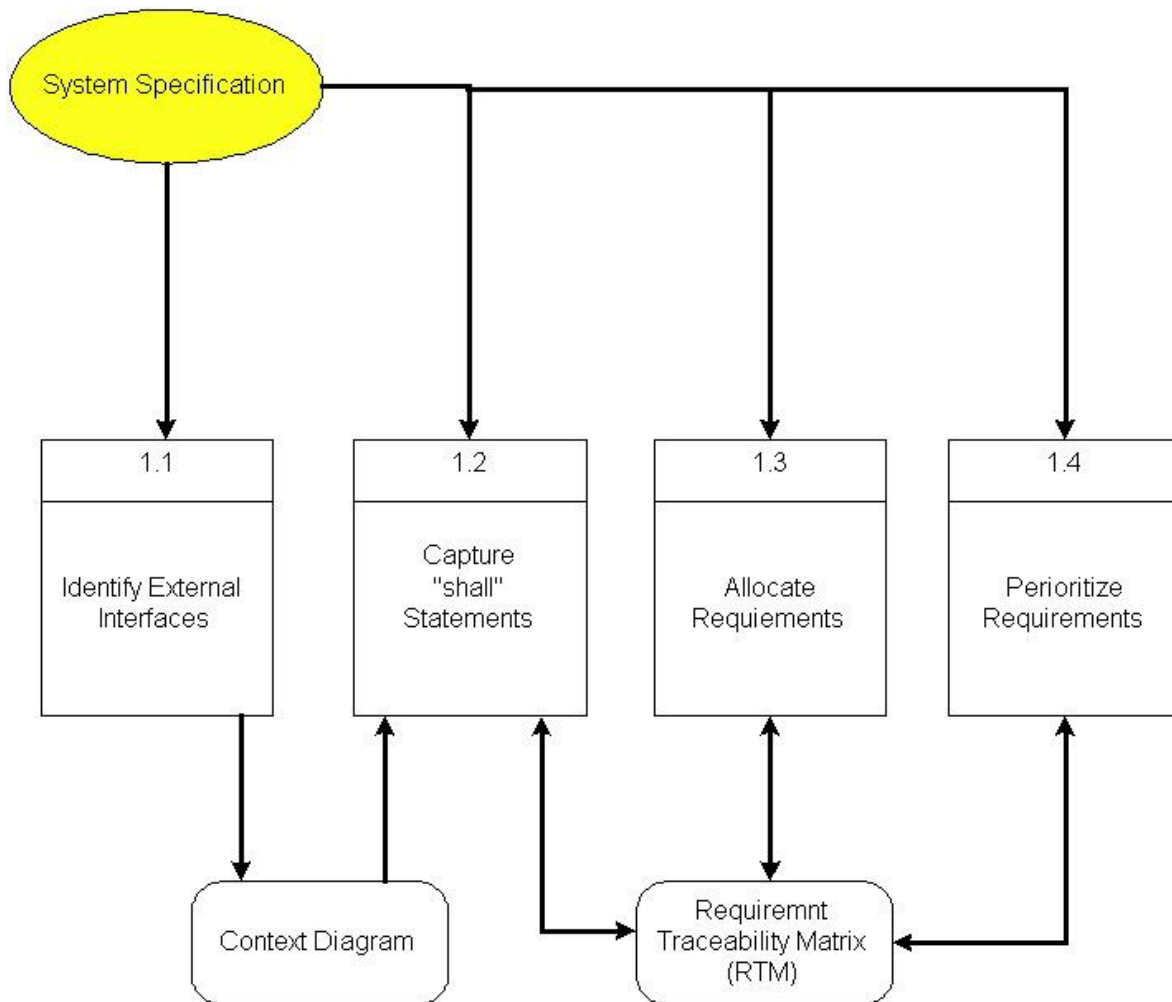
The Risk List is maintained throughout the project. It is created early in the Inception phase, and is continually updated as new risks are uncovered and existing risks are mitigated or retired. At a minimum, it is revisited at the end of each iteration, as the iteration is assessed.

*****REQUIREMENTS ENGINEERING*****

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

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.

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;

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.

1.4 Capture "shall" Statements

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

1.5 Allocate Requirements

Allocate the requirements in the use cases.

1.6 Prioritize Requirements

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

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.

1.8 Example

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

1.8.1 Introduction

Green Wood (GW) is a multinational company, which deals in manufacturing, delivery and selling of sports goods and sports ware throughout 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.

1.8.2 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:

1.8.2.1 Sport Goods Manufacturing Department

Deals in manufacturing of sport goods.

1.8.2.2 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

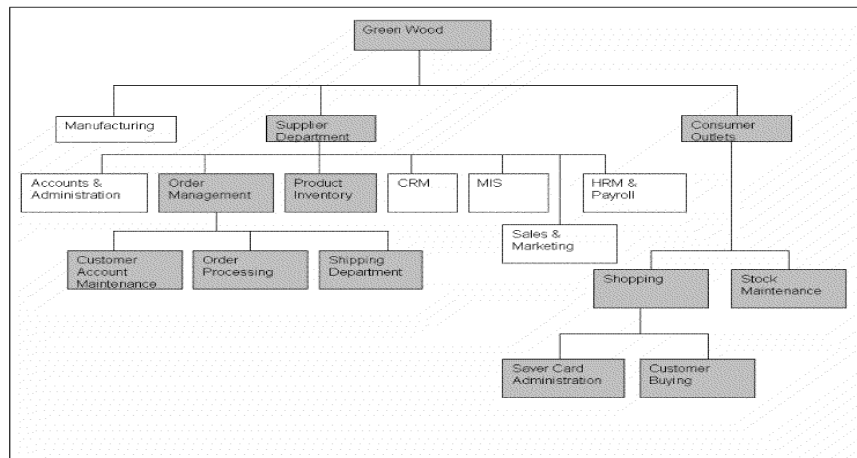
- Customer Account Maintenance
- Order Processing
- Shipping Department
- Product Inventory
- Accounts & Administration
- CRM
- MIS
- HRM & Pay Roll
- Sales & Marketing

1.8.2.3 GW Consumer Outlets & Showrooms

They directly deals with buying and selling of goods to customers

- Shopping Centre
- Stock Maintenance

1.8.3 Business Organization Chart



1.8.4 Scope of the System

The GW System is divided in to three phases.

1.8.4.1 Phase I

Phase I includes following business areas:

- Customer Account Maintenance
- Order Processing
- Product Inventory

1.8.4.2 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

1.8.4.3 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.

1.8.5 Summary of Requirements:(Initial Requirements)

The purposed system must fulfill following requirements as follow:

1.8.5.1 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.

1.8.5.2 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.

1.8.5.3 Consumer Dealing Department Requirements

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

1.8.5.4 Shopping Centre

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

1.8.5.5 Product Stock Maintenance

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

1.8.6 Identifying External Entities or Actors

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.

- An external entity (person or machine) that interacts with or uses the system
- Things that the project cannot control
- An actor is external to a system, interacts with the system, may be a human user or another system, and has goals and responsibilities to satisfy in interacting with the system. Actors address the question of who and what interacts with a system. In the UML, an actor is shown as a "stick figure" icon.

The Identification of External Entities or Actors is done in two phases.

1.8.7 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

1.8.8 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

1.8.9 Capture "shall" Statements and the external entities (Actors)

Para #	External Entity	Initial Requirements
1.0	customer	A customer "shall" place order for goods
1.0	customer	A customer "shall" register himself to the system
1.0	System	The system "shall" provide two types of registration process, normal and privileged
1.0	CA	CA "shall" accept, reject and temporarily waive the requests on the basis of credentials provided.
1.0	customer	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	customer	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	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	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	administrator	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	administrator	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

1.8.10 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_Privileged
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,

1.8.11 Priorities Requirements

<i>Para #</i>	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_Privileged
1.0	Medium	The system “shall” provide	UC_6	UC_Place_Order_Request

		two types of registration process, normal and privileged		
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	UC_15	UC_View_Customer_Details

		“shall” view his details for verification purposes		
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	UC_24	UC_View_Action,

		different actions, and correspondingly process these pending actions		
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

1.8.12 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	B1	UC_Update_Request	Business

		customers Request			
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

1.9 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.

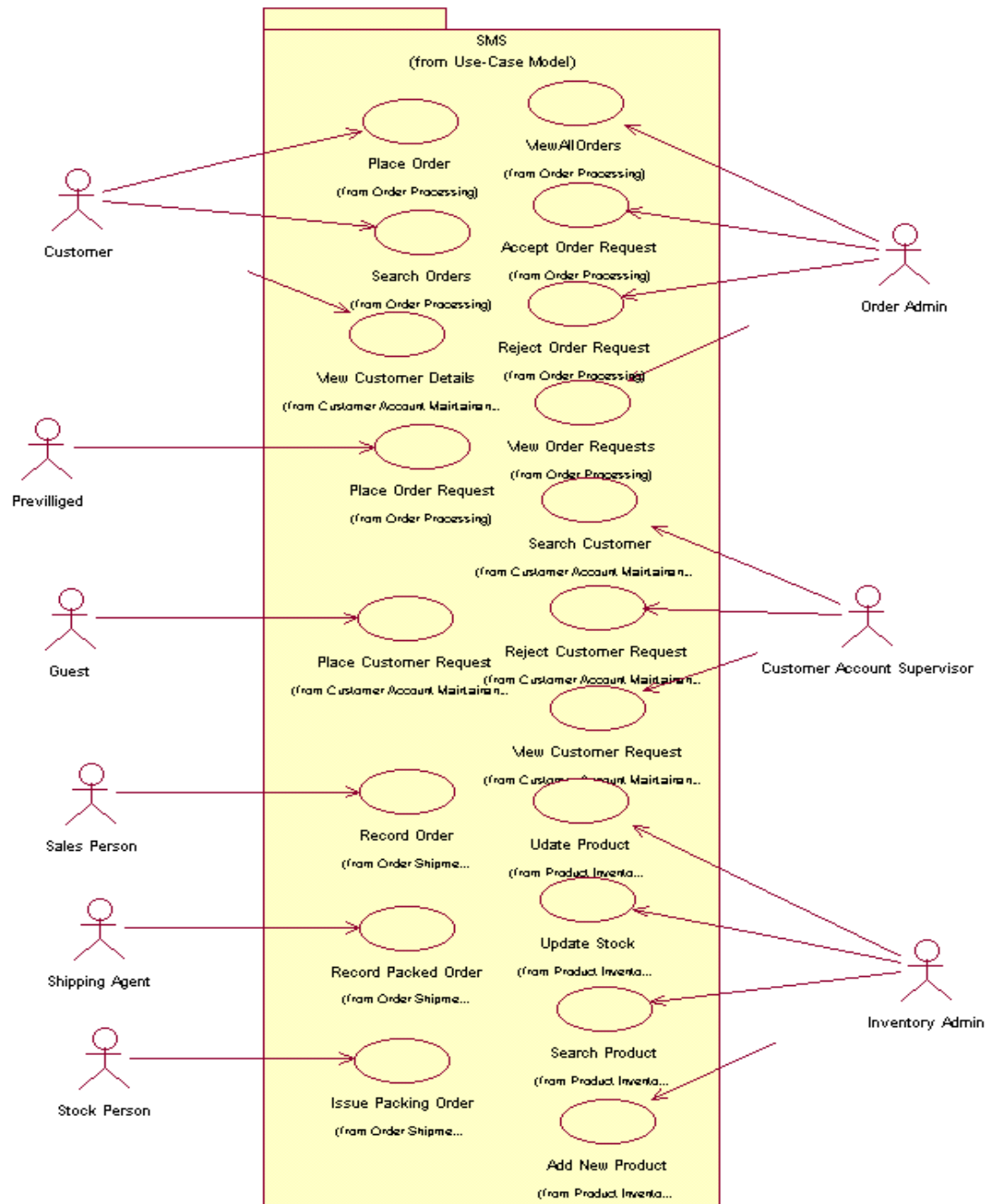
A use case is initiated by a user with a particular goal in mind, and completes successfully when that goal is satisfied. It describes the sequence of interactions between actors and the system necessary to deliver the service that satisfies the goal. It

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



1.1 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. Description of all use case's are written down. Use case description typically contains 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.

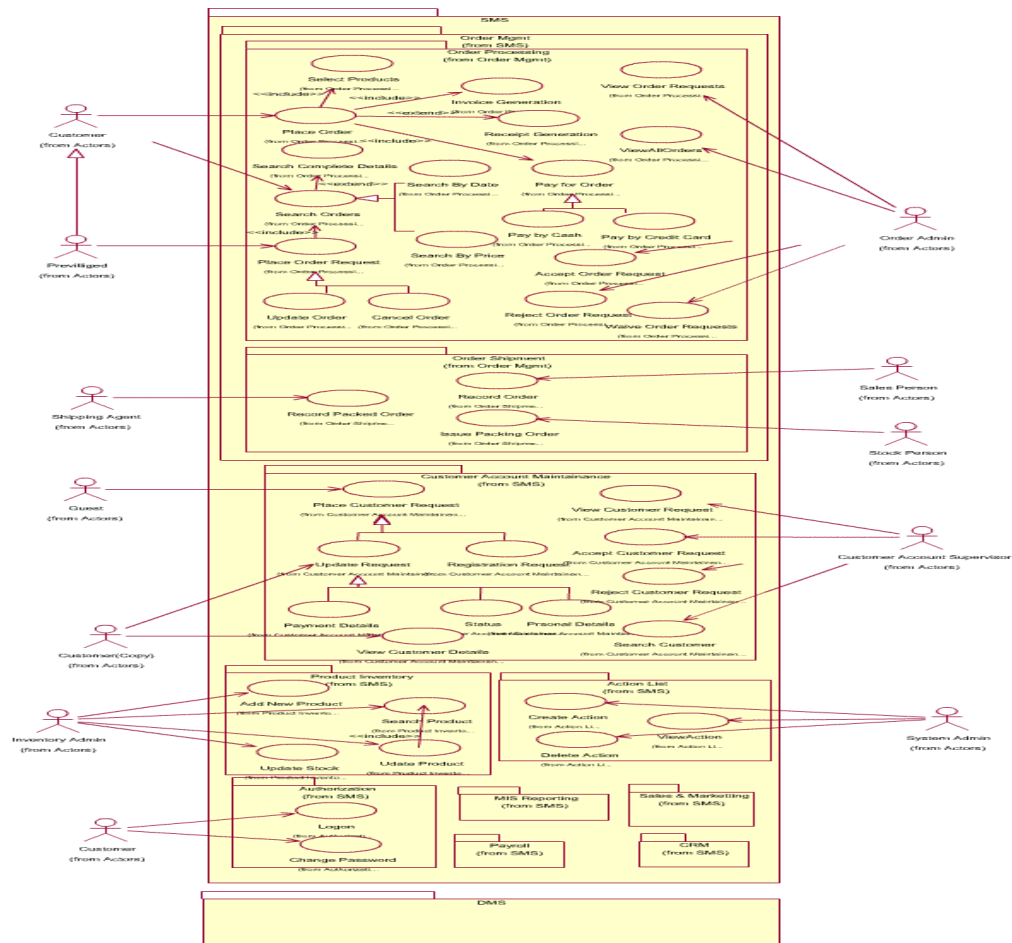
Post conditions

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.

1.2 Usecase Diagram (refined and updated)

Analysis level usecase diagram is a refined High level use case diagram and 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



1.3 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.

1.3.1 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

1.4 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.

1.4.1. Defining a Sequence diagram

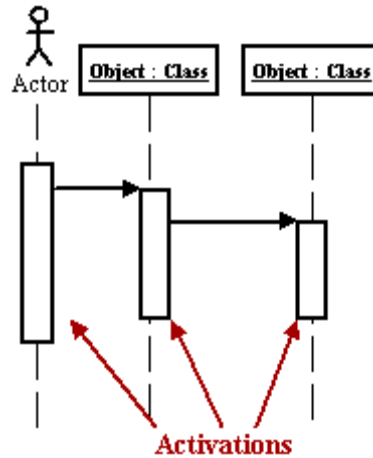
A sequence diagram is made up of objects and messages. Objects are represented exactly how they have been represented in all UML diagrams—as rectangles with the underlined class name within the rectangle.

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

1.4.2. 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.

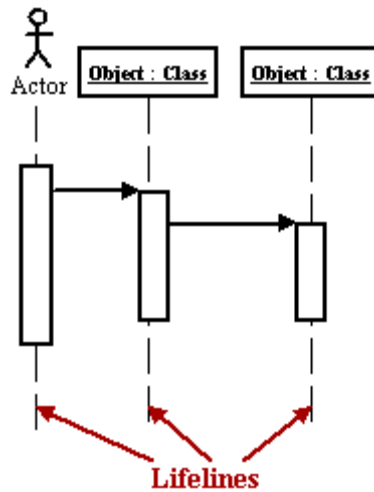
Messages

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.

Arrow	Message type
	Simple
	Synchronous
	Asynchronous
	Balking
	Time out

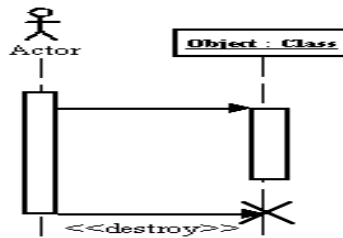
Lifelines

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



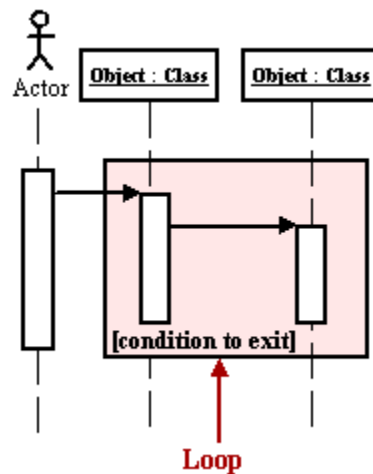
Destroying Objects

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.

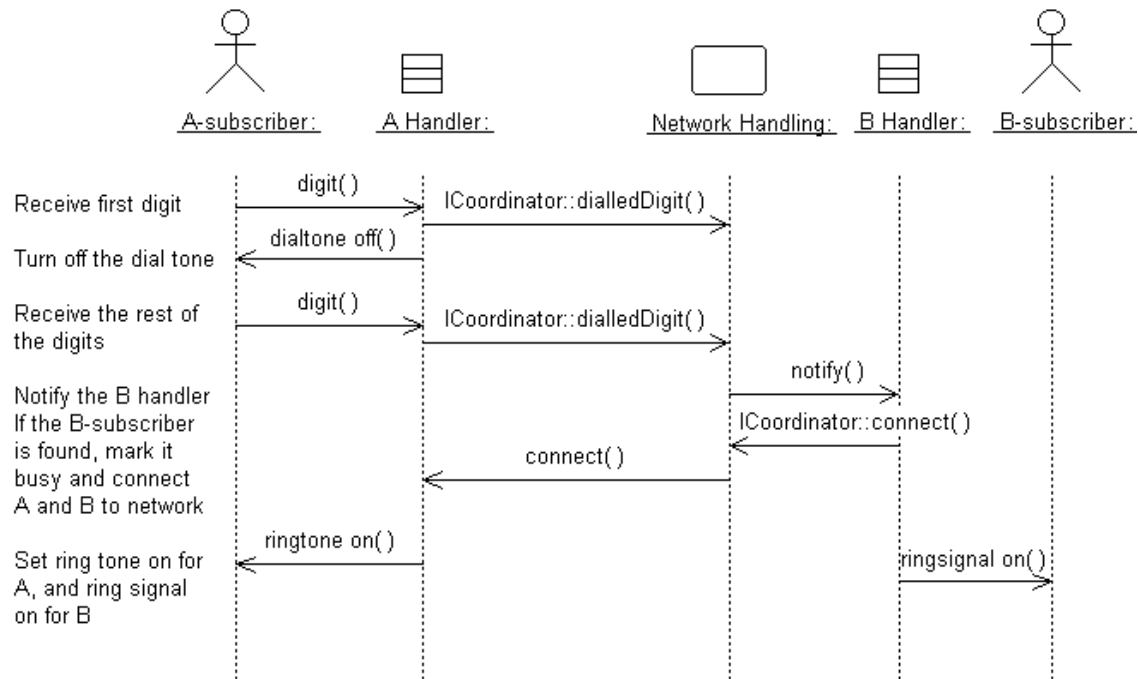
Scripts

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.

1.4.3 Example

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

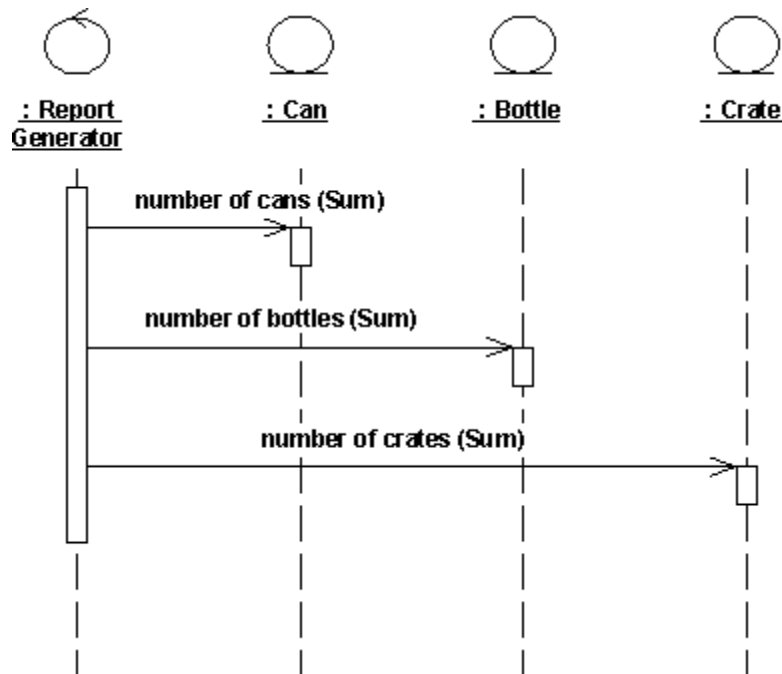


1.4.4 Distributing Control Flow in Sequence Diagrams

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.

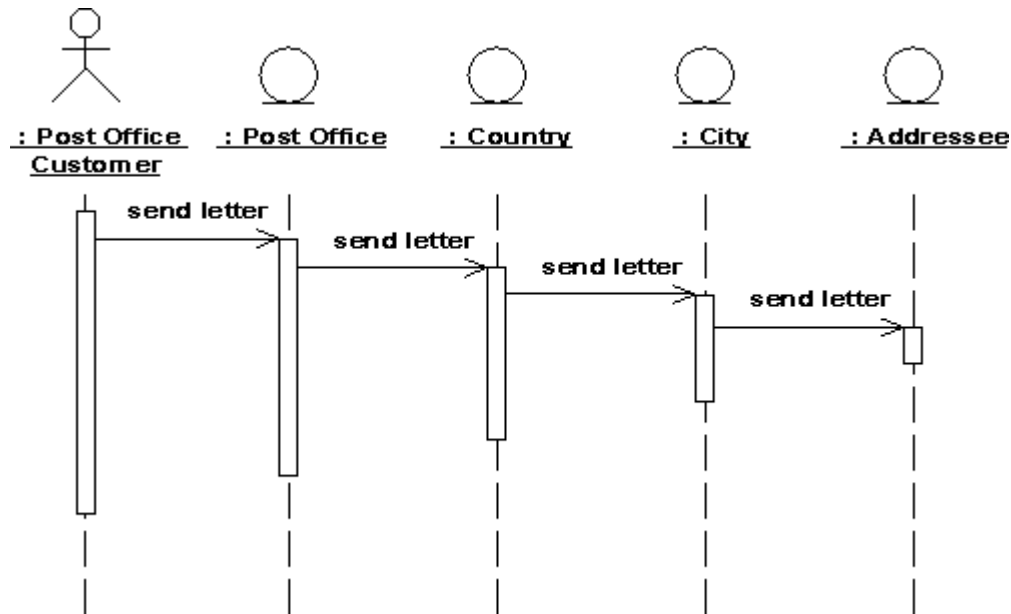
The use case behavior is a decentralized flow of events. The sub-event phases belong together. The sender of the letter speaks of "sending a letter to someone." He neither needs nor wants to know the details of how letters are forwarded in countries or cities. (Probably, if someone were mailing a letter 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.

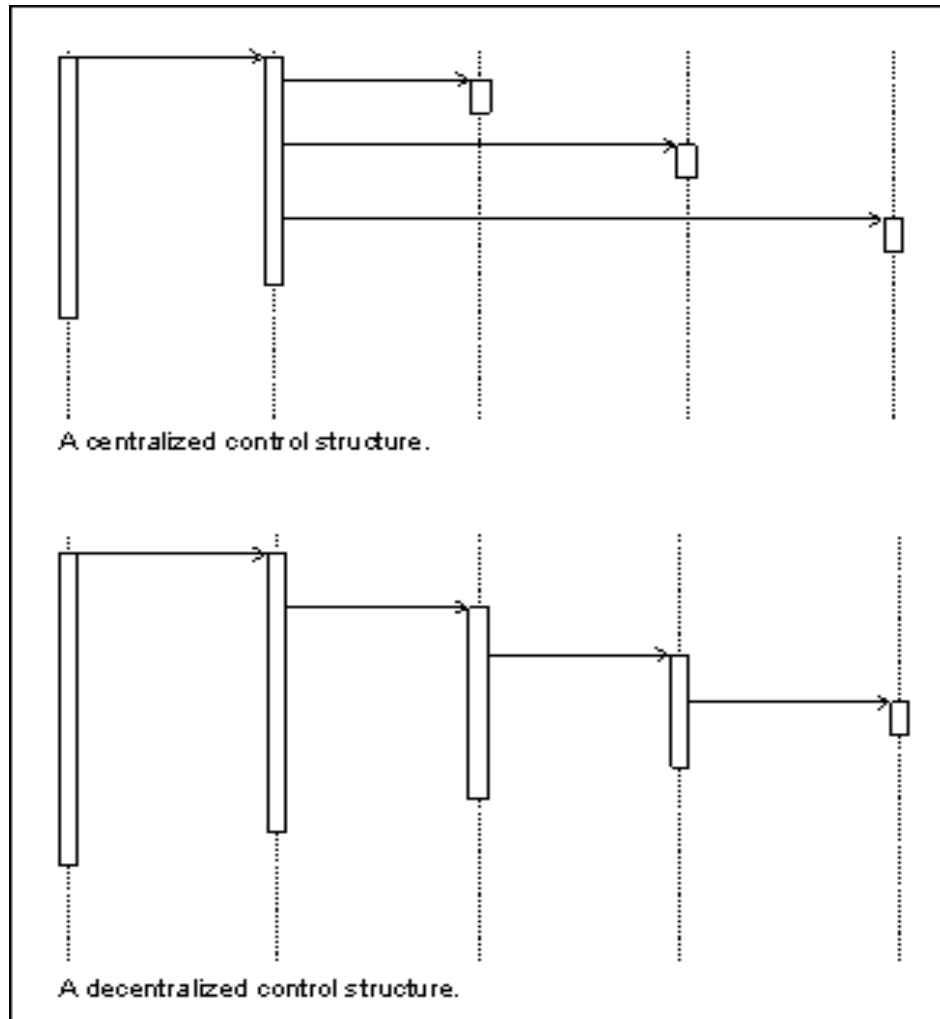
A centralized control structure in a flow of events produces a "fork-shaped" sequence diagram. A decentralized control structure produces a "stairway-shaped" sequence diagram.

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.



1.5 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 determine class responsibilities and interfaces.

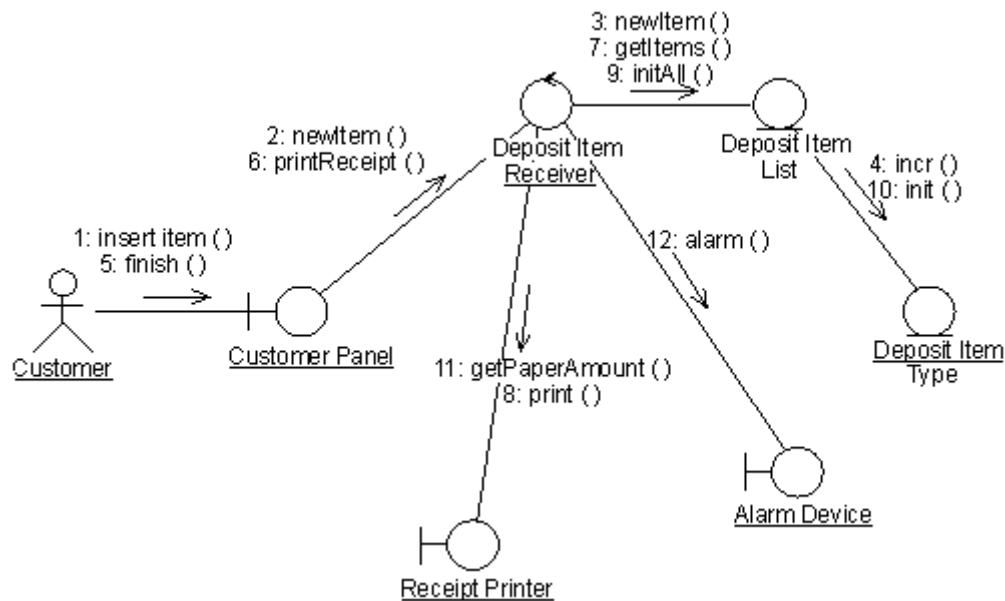
Unlike a sequence diagram, a collaboration diagram shows the relationships among the objects. Sequence diagrams and collaboration diagrams express similar information, but show it in different ways. Collaboration diagrams show the relationships among objects and are better for understanding all the effects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

Because of the format of the collaboration diagram, they tend to be 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.

1.5.1 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.

1.5.2 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.

1.6 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 postcondition 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.

1.7 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.

1.7.1 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.

1.7.2 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.

1.7.3 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.

1.7.4 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.

1.7.5 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.

1.7.6 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.

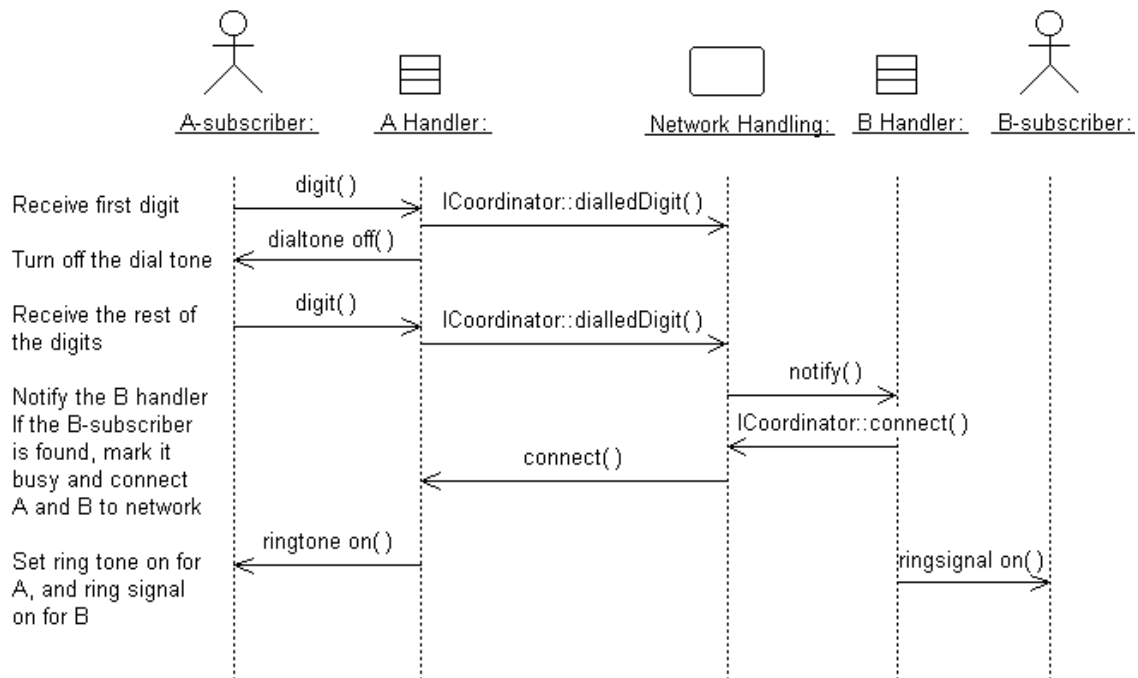
- Define Operations
- Identify Operations
- Name and Describe the Operations
- Define Operation Visibility
- Define Class Operations

1.7.7 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.

1.7.8 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.

1.7.9 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).

1.7.10 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:

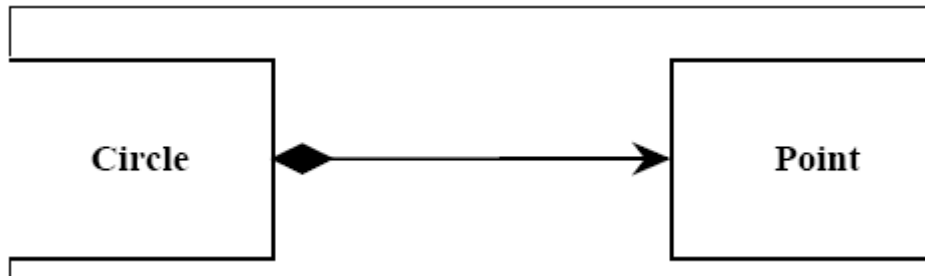
- What is to be done according to the requirements?
- What other objects and their operations are to be used?
- More specific requirements may concern:
- How parameters are to be implemented.
- Any special algorithms to be used.

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.

1.7.11 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`. However, 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.

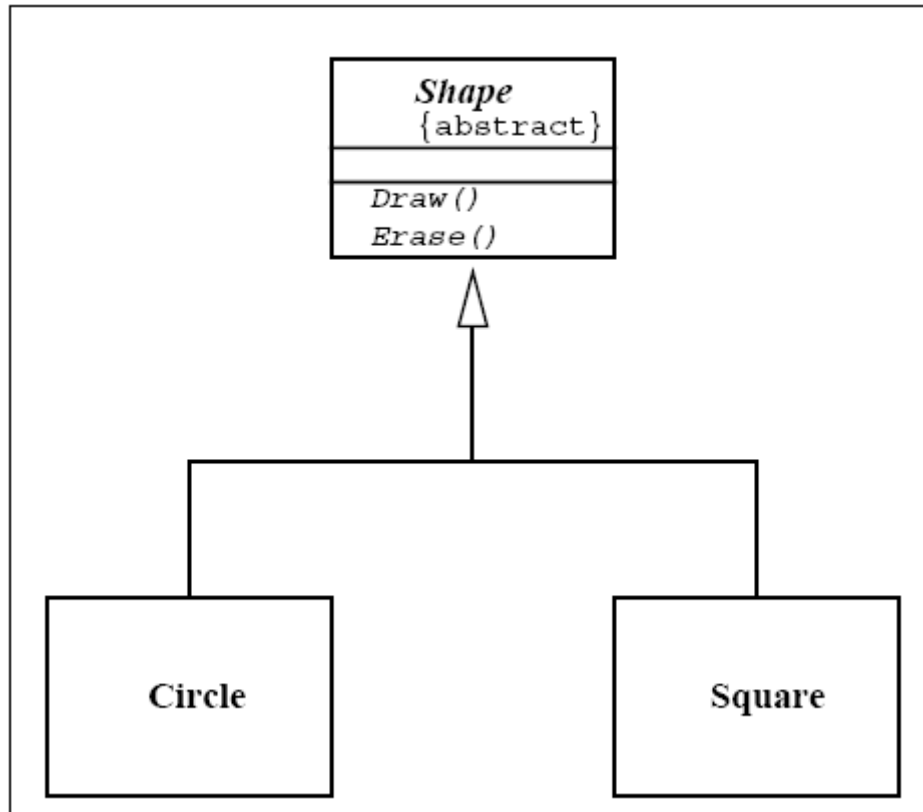
In C++ we would represent this as shown in Listing 1. In this case we have represented the composition relationship as a member variable. We could also have used a pointer so long as the destructor of `Circle` deleted the pointer.

```
class Circle
{
public:
    void SetCenter(const Point&);
    void SetRadius(double);
    double Area() const;
    double Circumference() const;
private:
    double itsRadius;
    Point itsCenter;
};
```

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

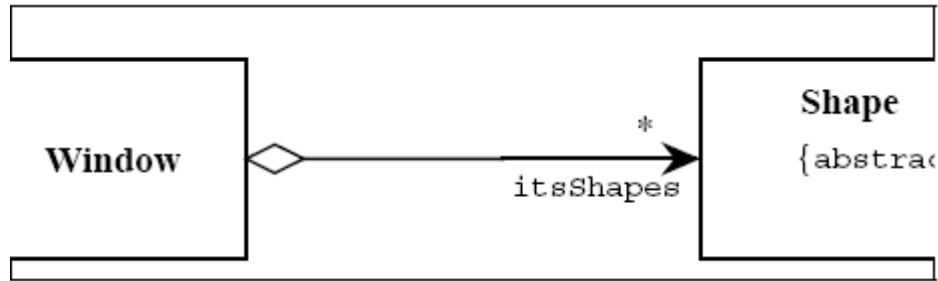
Circle and *Square* both derive from *Shape*. Note that the name of class *Shape* is shown in italics. This indicates that *Shape* is an abstract class. Note also that the operations, *Draw ()* and *Erase ()* are also shown in italics. This indicates that they are pure virtual.



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:

```

class Window
{
public:
    //...
private:
    vector<Shape*> itsShapes;
};
  
```

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.

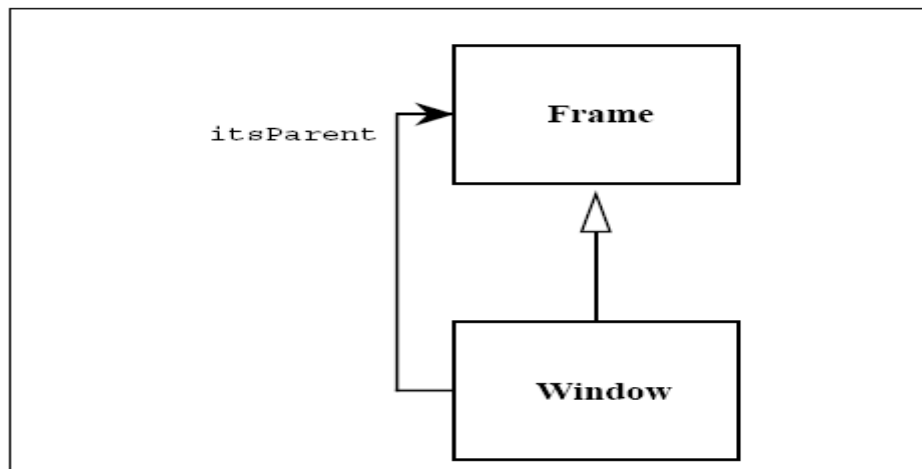


Figure shows how we draw an association. An association is nothing but a line drawn between the participating classes. In Figure 6 the association has an arrowhead to denote that Frame does not know anything about Window. Once again note the name on the role. This relationship will almost certainly be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

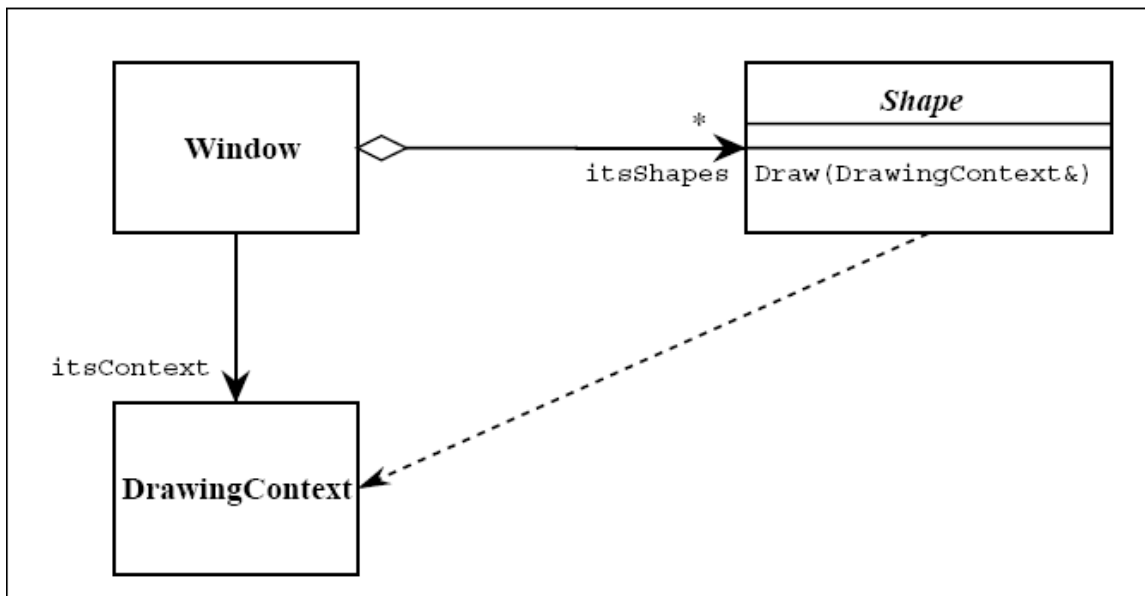
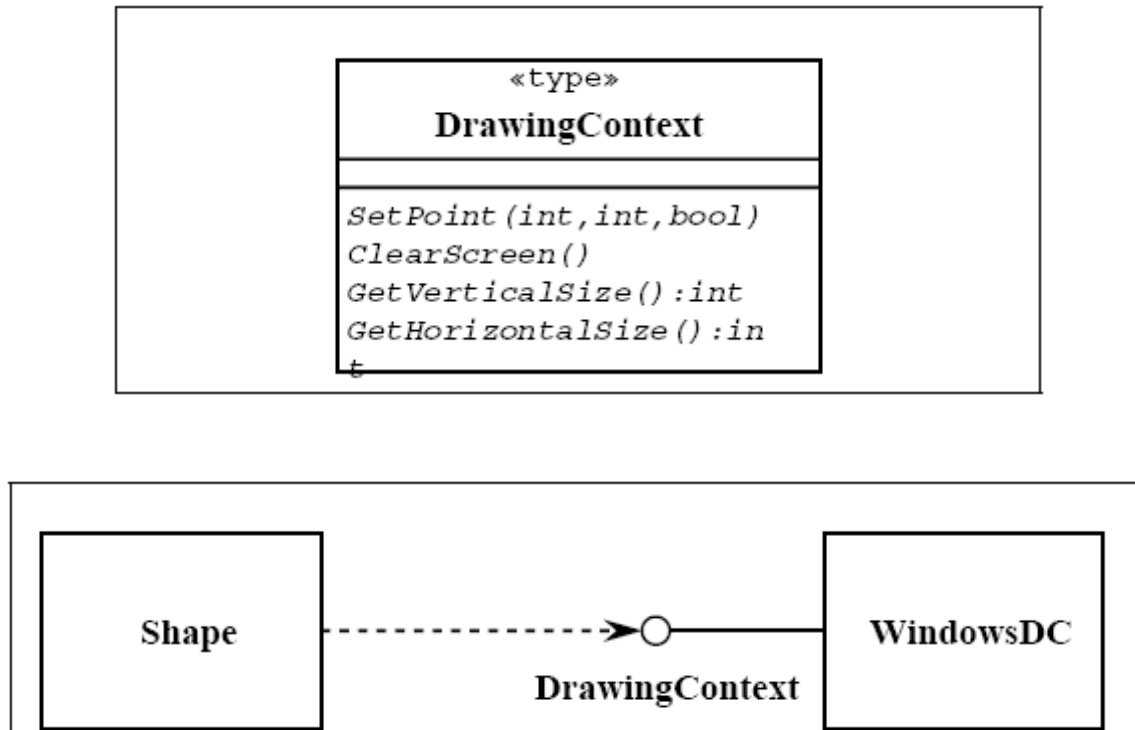


Figure shows a dashed arrow between the Shape class and the DrawingContext class. This is the dependency relationship. In Booch94 this was called a ‘using’ relationship. This relationship simply means that Shape somehow depends upon DrawingContext. In C++ this almost always results in a #include.

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 “lollipop” 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

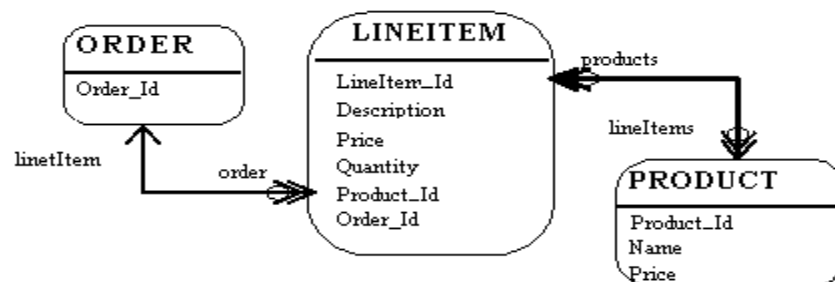


1.8 Data Model

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.

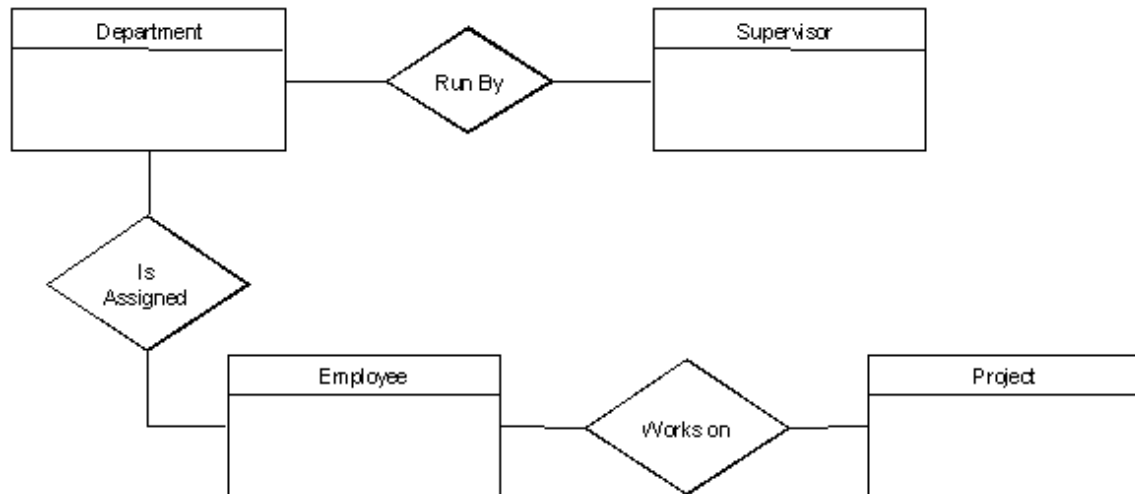
2. Find Relationships

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

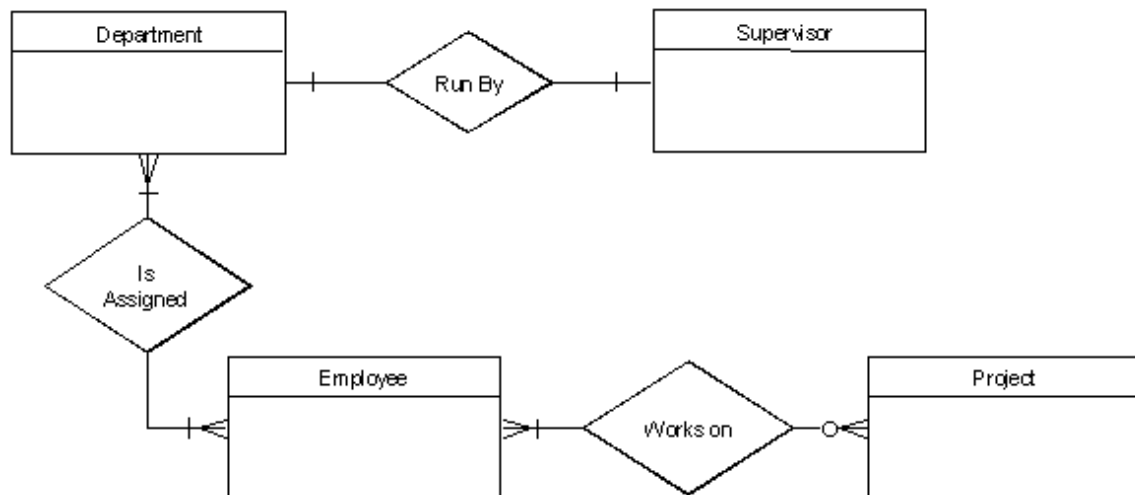
We connect the entities whenever a relationship is shown in the entity Relationship Matrix.



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.



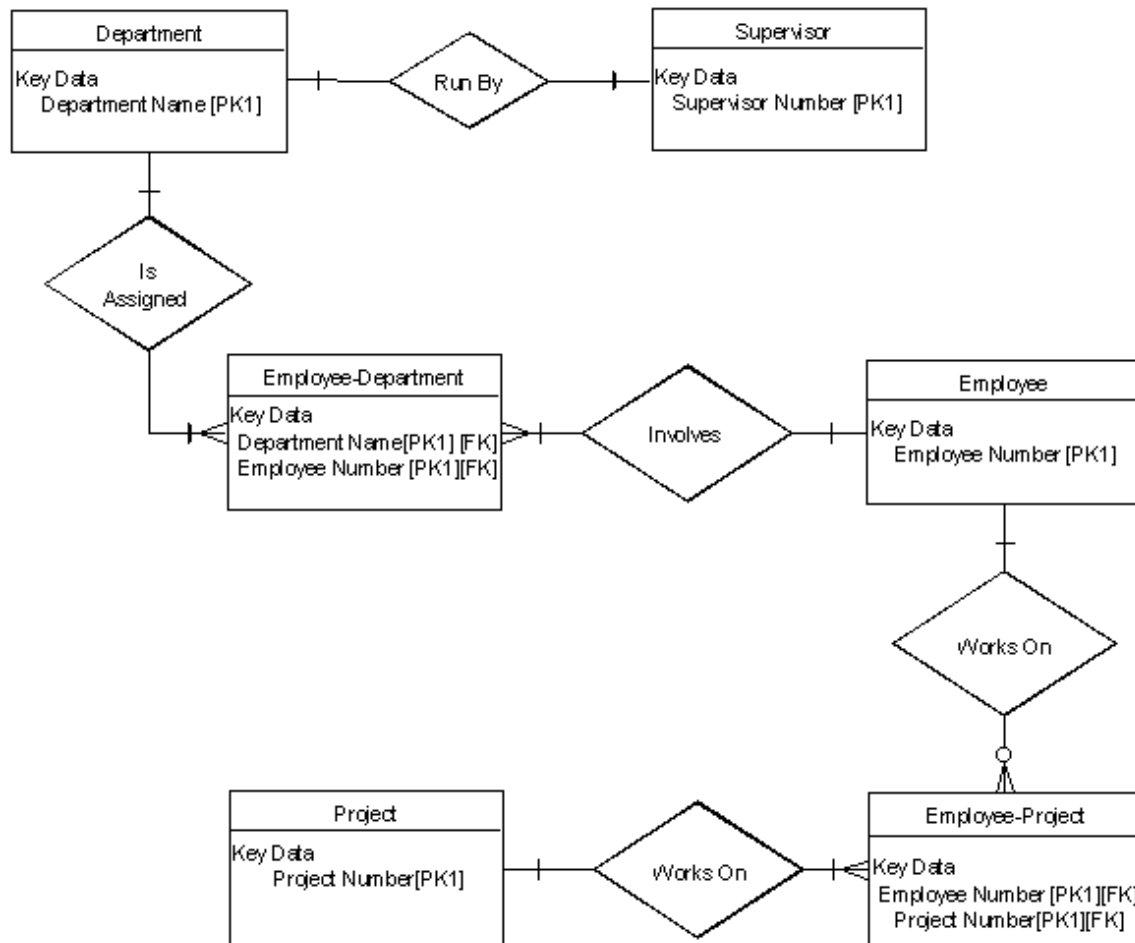
5. Define Primary Keys

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

