3.6.3 Software Risk Management Steps and Techniques

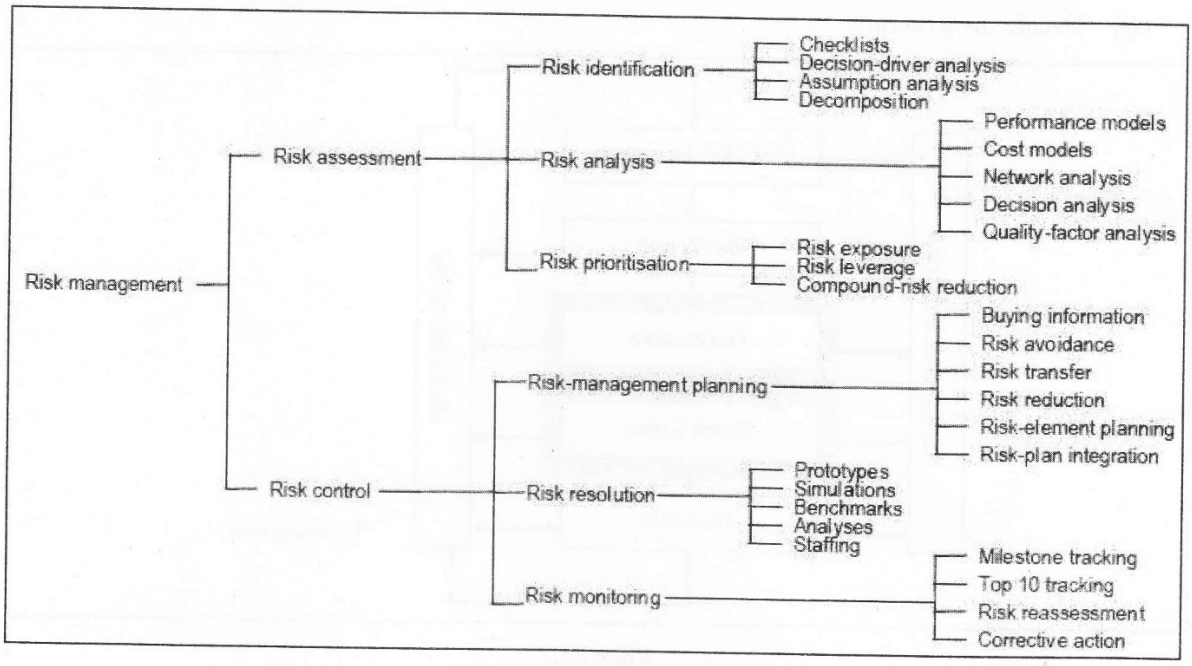


Figure 3.8: Software Risk Management Steps and Techniques

Risk Identification

Risk identification is used in risk management to answer the questions: What can happen? How can it happen? Risk identification is the process of recognizing the opportunities opened up by each activity or phase of the project and clarifying where the risk lies. The agreed tolerance of risk should help identify the amount of time should be spent in identifying risk, but, at least the 20% of the risks that would have 80% of the potential impact should be identified. There are many techniques to aid in risk identification and they generally fall under the heading of either quantitative or qualitative risk identification techniques.

Following are the main qualitative techniques of Risk Identification

- Assumptions Analysis
- Check Lists
- Prompt Lists
- Brainstorming
- Facilitated workshops
- Interviews

Assumptions Analysis

It is evitable that when you start planning or outlining your project you will be making assumptions. Making assumptions within a project will always create risks and a way to help manage this is to list all the assumptions of each phase or stage of your project against a timeline. Then think about the

consequence of the assumptions and how they affect the other parts of the project. Try creating a high level storyboard of the project showing risk and assumptions are associated, this can show the effect of decisions and should provide a better understanding about the risks in a program or project.

Check Lists

Risk checklists are often built upon a Project or Program Managers past experience or the Project Management experience of an organization. They will normally take into account

- Quality experience
- Formality of development
- Novelty of application
- Impact on business
- Requirements standards
- Software identification
- Projects concurrency
- Dependencies
- Project duration
- Flexibility of delivery
- Planning estimates
- Stability of suppliers
- Range of sites
- Impact upon status quo

Prompt Lists

Risks here will be identified by logical examination of each Program or Project aspect. Following is a list of common areas from which risk can arise.

Brainstorming or SWOT Analysis

Within SWOT analysis risks are identified by looking at the Strengths (or perceived Strength), Weakness, Opportunity & Threats to the success of the project or program. This is usually done within a Workshop environment.

Facilitated Workshops

Within SWOT analysis risks are identified by looking at the Strengths (or perceived Strength), Weakness, Opportunity & Threats to the success of the project or program. This is usually done within a Workshop environment.

Interviews

Interviews conducted to identify risk will only be successful where there is:

- Good preparation;
- Clear objectives;

- A positive & supportive environment;
- Proper time management;
- Use of open questions;
- Challenging not confrontational atmosphere.

The results of interviews should be well documented.

External	
Infrastructure	Relating to infrastructures such as computer networks, transport systems for staff, power supply systems
Economic	Relating to economic factors such as interest rates, exchange rates, inflation
Legal & Regulatory	Relating to the laws and regulations which if complied with should reduce hazards (E.g. - Health and Safety at Work Act)
Environmental	Relating to issues such as fuel consumption, pollution
Political	Relating to possible political constraints such as a change of government
Market	Relating to issues such as competition and supply of goods
Act of God	Relating to issues such as fire, flood earthquake
Financial	
Budgetary	Relating to the availability of resources or the allocation of resources
Fraud or Theft	Relating to the unproductive loss of resources
Insurable	Relating to the potential areas of loss which can be insured against
Capital Investment	Relating to the making of appropriate investment decisions
Liability	Relating to the right to sue or be sued in certain circumstances
Activity	
Policy	Relating to the appropriateness and quality of policy decisions
Operational	Relating to the procedures employed to achieve particular objectives
Information	Relating to the adequacy of information which is used for decision making
Reputational	Relating to the public reputation of the organisation and consequent effects
Transferable	Relating to risks which can be transferred or the transfer of risks at inappropriate cost
Technological	Relating to the use of technology to achieve objectives
Project	Relating to project planning and management procedures
Innovation	Relating to the exploitation of opportunities to make gains
Human Resources	
Personnel	Relating to the availability and retention of staff
Health and Safety	Relating to the well-being of people

Quantitative Risk Management Techniques

There are three main quantitative techniques;

- Decision trees
- Influence diagrams
- Monte Carlo Simulation

Decision Trees

Rather like flowchart diagrams these represent a method of looking at, for example, two options and making a decision. By analyzing the impact each decision will have, the risks of taking that decision

can be forecast and used to anticipate problems or inform the direction the project takes. This technique is best suited to simpler situations. In complex scenarios they can become confusing and complicated.

Influence Diagrams

This technique results in a diagram, which is similar to a project network diagram or Microsoft Project PERT charts. In this case each box will contain a variable or decision, which will have an influence on future progress. By analyzing the impact each variable will have, the risks of taking one path over another can be forecast and used anticipate problems or inform the direction the project takes.

Monte Carlo Simulation (or 3-Point Estimation)

Looking at both best and worse case scenarios as well as most likely scenario and then planning what the impact of each is. This can be plotted against the Project Baseline and the Critical Path to show the consequence of risk and allow you to anticipate suitable response to risk.

Risk Estimation

Risk, at the general level, involves two major elements: the occurrence probability of an adverse event and the consequences of the event. Risk estimation, consequently, is an estimation process, starting from the occurrence probability and ending at the consequence values.

Risk estimation involves following activities:

- Discussion of source, exposure issues
- Communication of results with stakeholders
- Assess changes in knowledge/perception in light of new information.

During the Risk Estimation step of risk management, the frequency and consequences associated with each risk scenario are estimated and communicated with stakeholders. Stakeholders may have important knowledge of sources and patterns of exposure that analysts will need to integrate into a risk assessment. However conflict is most likely to arise at this step as stakeholders are not typically involved in the risk estimation process, and the uncertainties and value assumptions associated with the methods may not be clearly communicated.

During the Risk Estimation stage, stake holder's knowledge and perceptions are assessed in light of receiving new information resulting from the risk estimates and the stakeholder analysis is updated. Third party review by third party experts and explicit communication of the methods, assumptions and uncertainties will contribute to credibility and trust in the technical analyses.

Software Estimation Risks

The effects of inaccurate software estimation and schedule overruns are well known. The problem stems from an inability to accurately assess risks associated with various software development projects. Software estimation errors generally result from four major risk areas, which are:

1. The inability to accurately size the software project. This results in poor implementations, emergency staffing, and cost overruns caused by underestimating project needs.
2. The inability to accurately specify a development environment which reflects reality. This results in defining cost drivers which may be inappropriate, underestimated or overestimated.

3. The improper assessment of staff skills. This results in misalignment of skills to tasks and ultimately miscalculations of schedules and level of effort required, as well as either underestimating or overestimating project staffing requirements.
4. The lack of well defined objectives, requirements, and specifications, or unconstrained requirements growth during the software development life cycle. This results in forever changing project goals, frustration, customer dissatisfaction, and ultimately, cost overruns.

All potential risks associated with the proposed software development project should be defined and weighed, and impacts to project cost should be determined. This information should always be included in the software estimation process.

Risk Exposure

Quantifying the effects of a risk by multiplying the risk impact by the risk probability yields risk exposure.

Risk-exposure = Risk-impact x Risk-probability

For each risk, the *Risk Exposure* is defined as the probability of the undesirable outcome times the size of the loss involved. Risk exposure helps us to list the risks in priority order, with the risks of most concern given the highest priority. Next, we must take steps to control the risks. The notion of control acknowledges that we may not be able to eliminate all risks.

Identifying Risk Exposure

It is the responsibility of project manager to ensure that team has an understanding of the project's exposure to risk. Gaining this understanding can be achieved through identifying categories of risk and then answering questions associated with each of these categories. The next section provides project managers with a set of questions to ask about their projects to help categorize risk.

Risk Categories and Questions

- Business/Strategic
- External Factors
- Procurement
- Organizational Factors
- Management
- Technical

Strategic

1. Do the project objectives fit into the organizations overall business strategy?
2. When is the project due to deliver?
3. What would be the result of late delivery?
4. What would be the result of limited success (functionality)?
5. What is the stability of the business area?

External Factors

1. Is the project exposed to requirements due to international interests (foreign legal implications or foreign company involvement)
2. Could there be political implications of the project failure?
3. Is this project a part of the larger program? IF so, what constraints are set for the project by the program?

Procurement

1. Does the supplier have a reputation for delivery of high quality?
2. Is the contract sufficiently detailed to show what the supplier is going to provide?
3. Are the acceptance criteria clear to both the parties?
4. Is the contract legally binding/enforceable?

Organizational Factors

1. What consideration needs to be given to security of the project?
2. Does the project have wholehearted support from senior management?
3. What is the commitment of the user management?
4. Have training requirements been identified? Can these requirements be met?

Management

1. How clearly are the project objectives defined?
2. Will the project be run using well-documented approach to project management?
3. Does this approach cover aspects of quality management, risk management and development activities in sufficient depth?
4. How well does the project team understand chosen methodology?
5. What is the current state of project plans?
6. Is the completion of project dependent on completion of other projects?
7. Are the tasks in project plan interdependent?
8. Can a critical path through the project tasks be identified?
9. What is the availability of appropriate resources?
10. What are the skills and experience of project team?
11. Will people be available for training?
12. How many separate users are involved?
13. How much changes will there be for user's operation or organization?

Technical

1. Is the specification clear, concise accurate and feasible?
2. How have the technical options been evaluated?
3. What is the knowledge of equipment (hardware/software)?
4. Does the experience of project manager cover a similar application?
5. Is this a new application?
6. What is the complexity of system?
7. How many sites will the system are implemented in?
8. Is the proposed equipment new/leading edge?
9. Who is responsible for defining system testing?
10. Who is responsible for defining acceptance testing?
11. On what basis is the implementation planned?
12. What access will the project team have to development/testing facilities?
13. Will users of data processing staff use the system?
14. Have requirements for long-term operations, maintenance and support been identified?

With risk identified, it is much easier to develop a plan to eliminate or manage risk.

Risk Migration

Risk Migration is the process of reducing risk exposure, either by decreasing the probability of the risk occurring, or by finding ways to reduce the possible impact if it does occur.

Risk Management Plans

The Risk Management Plan (RMP) presents the process for implementing proactive risk management as part of overall project management. The RMP describes techniques for identifying, analyzing, prioritizing & tracking risks; developing risk-handling methods; & planning for adequate resources to handle each risk, should they occur. The RMP also assigns specific risk management responsibilities & describes the documenting, monitoring & reporting processes to be followed.

Purpose

*Risk Management Plan d*efines how risks will be managed during the lifecycle of the program. It is used to plan the way risks are handled within the program. The Risk Strategy and supporting Plan must acknowledge actual and potential threats to the successful delivery of a project and determines the activities required to minimize or eliminate them. The risk plan needs to be capable of integration into or coordination with the project plan.

Check Your Progress

Discuss the following in brief:

1. Risk Migration.
2. Putnam's work in Putnam Resource Allocation Model.
3. Steps of Software Estimation.
4. Project Planning Document Structure.
5. Three levels of COCOMO.

3.7 LET US SUM UP

Software project planning plays a very important role in success of any Software implementation Project Scopes, Estimations, Resources are to be very critically calculated and monitored with the application of COCOMO model for successfully finishing project in time and in right cost. The overall process of developing a cost estimate for software is not different from the process for estimating any other element of cost.

3.8 KEYWORDS

COCOMO: Constructive Cost Model

RPM: Risk Management Plan

CSCI: Computer Software Configuration Item

R& D: Research and Development

DSI: Delivered Source Instructions

LRU: Line Replaceable Unit

3.9 QUESTIONS FOR DISCUSSION

1. Discuss the Putnam Resource Allocation Model.
2. What are the qualitative risk identification techniques?
3. Write briefly about risk estimation.
4. Explain the levels and modes of COCOMO model.
5. What is Cost Estimation?
6. Explain why the process of project planning is an iterative one and why a plan; must be continually reviewed during a software project.
7. Briefly explain the purpose of each of the sections in a software project plan.

Check Your Progress: Model Answers

1. The process of reducing risk exposure.
2. The problem of staffing of software projects.
3. Define Project Objectives and Requirements and plan the Activities.
4. Introduction (goals, constraints, etc.), project organization, risk analysis, hardware and software resource requirements etc.
5. Basic, intermediate and detailed.

3.10 SUGGESTED READINGS

R.S. Pressman, *Software Engineering—A Practitioner's Approach*, (5th edition), Tata McGraw Hill Higher Education.

Rajib Mall, *Fundamentals of Software Engineering*, PHI, 2nd Edition.

Sommerville, *Software Engineering*, Pearson Education, 6th Edition.

Richard Fairpy, *Software Engineering Concepts*, Tata McGraw Hill, 1997.

LESSON

4

SOFTWARE REQUIREMENT ANALYSIS AND SPECIFICATIONS

CONTENTS

4.0 AIMS AND OBJECTIVE

After studying this lesson, you should be able to:

- Discuss requirements analysis and engineering
- Explain problem analysis and approaches to problem analysis
- State software requirements specification
- Describe behavioral and non-behavioral requirements

4.1 INTRODUCTION

Software requirements gathering is a crucial step. It decides precisely what is to be built. No other work is as tough as fixing the detailed technical requirements. It can lead to a crippled system if done in a wrong manner. It is also difficult to rectify.

- The process of eliciting, analyzing, documenting, and validating the services required of a system and the constraints under which it will operate and be developed.
- Descriptions of these services and constraints are the requirements for the system.
- Requirements may be high-level and abstract, or detailed and mathematical.
- The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult... No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later - Fred Brooks, "No Silver Bullet..."

Why is Requirements Engineering so hard?

- Difficulty of anticipation.
- Unknown or conflicting requirements/priorities.
- Conflicts between users and procurers.
- Fragmented nature of requirements.
- Complexity/number of distinct requirements.

Some Analogies

- Working a dynamically changing jigsaw puzzle.
- Blind men describing an elephant.
- Different medical specialists describing an ill patient.

4.2 SOFTWARE REQUIREMENT ANALYSIS

4.2.1 Types of Requirements

- Requirements range from being high-level and abstract to detailed and mathematical.

- This is inevitable as requirements may serve multiple uses.
  - ❖ May be the basis for a bid for a contract — must be open to interpretation.
  - ❖ May be the basis for the contract itself — must be defined in detail.
  - ❖ May be the basis for design and implementation - must bridge requirements.

Engineering and Design Activities

- ***User requirements:*** Statements in natural language plus diagrams of system services and constraints. Written primarily for customers.
- ***System requirements:*** Structured document setting out detailed descriptions of services and constraints precisely. May serve as a contract between client and developer.
- ***Software design specification:*** Implementation oriented abstract description of software design, which may utilize formal (mathematical) notations. Written for developers.

Functional and Non-functional Requirements

- ***Functional requirements*** – services the system should provide, how it should react to particular inputs, or how it should behave in particular situations.
- ***Non-functional requirements*** – constraints on services or functions (e.g., response time) or constraints on development process (e.g., use of a particular CASE toolset).
- ***Domain requirements*** – functional or non-functional requirements derived from application domain (e.g., legal requirements or physical laws

Examples of Functional Requirements

- The user shall be able to search either all of the initial set of databases or select a subset from it.
- The system shall provide appropriate viewers for the user to read documents in the document store.
- Every order shall be allocated a unique identifier (ORDER_ID), which the user shall be able to copy to the account's permanent storage area

Non-functional Requirements

- Define system attributes (e.g., reliability, response time) and constraints (e.g., MTTF5K $\geq$ transactions, response time $\leq$ 2 seconds).
- Attributes are often emergent system properties - i.e., only observable when the entire system is operational.
- Process constraints may mandate a particular CASE system, programming language, or development method.
- Non-functional requirements may be more critical than functional requirements. If not met, the system may be useless.

Non-functional Classifications

- ***Product requirements:*** Specify product behavior

- ***Organizational requirements:*** Derived from policies/procedures in customer's or developer's organization (e.g., process constraints)
- ***External requirements:*** Derived from factors external to the product and its development process (e.g., interoperability requirements, legislative requirements)

Examples

Product Requirement

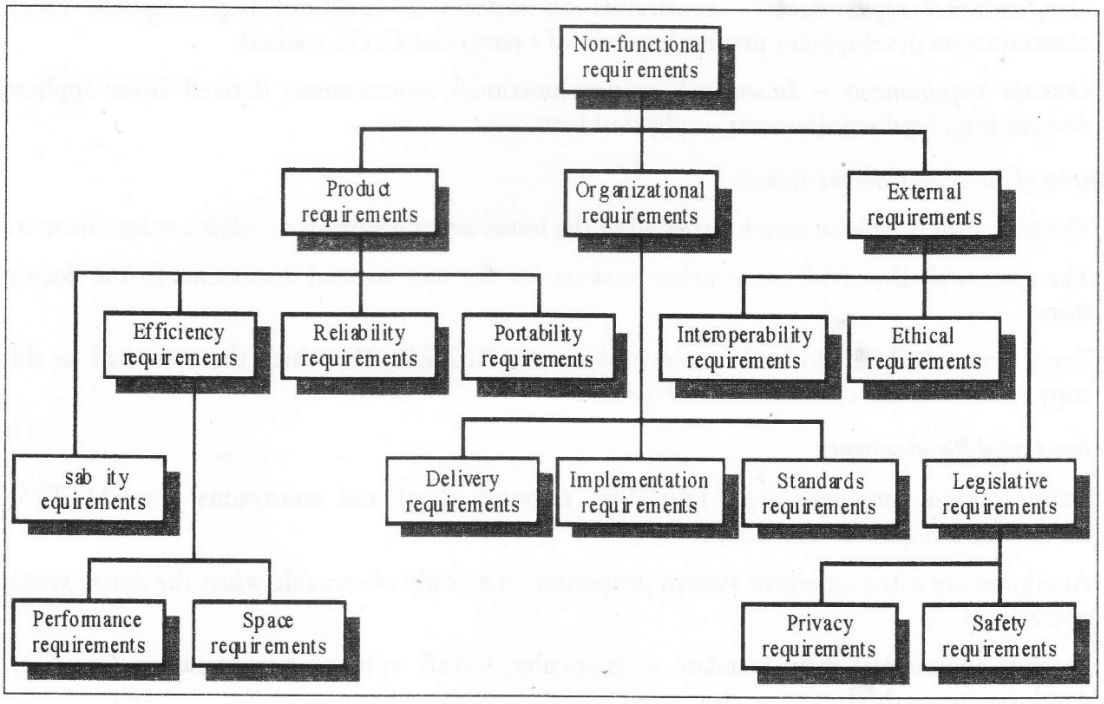
- It shall be possible for all necessary communication between the APSE and the user
- To be expressed in the standard Ada character set.

Organizational Requirement

- The system development process and deliverable documents shall conform to the process and deliverables defined in XYZCo-SP-STAN-95.

External Requirement

- The system shall not disclose any personal information about customers apart from their name and reference number to the operators of the system.



4.2.2 Goals versus Requirements

General goals such as "system should be user friendly" or "system should have fast response time" are not verifiable.

Goals should be translated into quantitative requirements that can be objectively tested.

Examples

- *A system goal:* The system should be easy to use by experienced controllers and should be organized in such a way that user errors are minimized.
- *A verifiable non-functional requirement:* Experienced controllers shall be able to use all the system functions after a total of two hours training. After this training, the average number of errors made by experienced users shall not exceed two per day.

Requirements Measures

Property	Measure
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	K Bytes Number of RAM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Requirement Interactions

Competing/conflicting requirements are common with complex systems. Spacecraft system example:

- To minimize weight, the number of chips in the unit should be minimized.
- To minimize power consumption, low-power chips should be used.
- But using low-power chips means that more chips have to be used. Which is the most critical requirement?

For this reason, preferred points in the solution space should be identified.

4.2.3 Domain Requirements

- Derived from the application domain rather than user needs.
- May be new functional requirements or constraints on existing requirements.
- If domain requirements are not satisfied, the system may be unworkable.

Library System Domain Requirements

There shall be a standard user interface to all databases, which shall be based on the Z39.50 standard.

Because of copyright restrictions, some documents must be deleted immediately on arrival. Depending on the user's requirements, these documents will either be printed locally on the system server for manually forwarding to the user or routed to a network printer.

Domain Requirements Problems

- *Understandability:* Requirements are expressed in the language of the application domain and may not be understood by software engineers.
- *Implicitness:* Domain experts may not communicate such requirements because they are so obvious (to the experts).

User requirements "should"

- Should be understandable by system users who don't have detailed technical knowledge.
- Should only specify external system behavior.
- Should be written using natural language, forms, and simple intuitive diagrams.

Some potential problems with using natural language:

- *Lack of clarity* - expressing requirements precisely is difficult without making the document wordy and difficult to read.
- *Requirements confusion* - functions, constraints, goals, and design info may not be clearly distinguished.
- *Requirements amalgamation* - several different requirements may be lumped together.

Guidelines for Writing User Requirements

- Adopt a standard format and use it for all requirements.
- Use language in a consistent way. E.g., use shall for mandatory requirements, should for desirable requirements.
- Use text highlighting to identify key parts of the requirement.
- Avoid the use of computer jargon.

System Requirements

- More detailed descriptions of user requirements
- May serve as the basis for a contract
- Starting point for system design & implementation
- May utilize different system models such as object or dataflow

System Requirements and Design Information

- In principle, system requirements should state what the system should do, and not how it should be designed.
- In practice, however, some design info may be incorporated, since:
- Sub-systems may be defined to help structure the requirements.
- Interoperability requirements may constrain the design.
- Use of a specific design model may be a requirement

More Potential Problems with using Natural Language

- *Ambiguity:* The readers and writers of a requirement must interpret the same words in the same way. NL is naturally ambiguous so this is very difficult.
- *Over-flexibility:* The same requirement may be stated in a number of different ways. The reader must determine when requirements are the same and when they are different.
- *Lacks of modularization:* NL structures are inadequate to structure system requirements sufficiently.

Requirements Document Requirements

- Specify external system behavior
- Specify implementation constraints
- Easy to change (!)
- Serve as reference tool for maintenance
- Record forethought about the life cycle of the system i.e. predict changes
- Characterize responses to unexpected events

IEEE Requirements Standard

- Introduction
- General description
- Specific requirements
- Appendices
- Index

This is a generic structure that must be instantiated for specific systems

Requirements Document Structure

- Preface (readers, version, change history)
- Introduction
- Glossary
- User requirements definition
- System requirements specification
- System models
- System evolution
- Appendices
- Index

4.2.4 Tools for Developing System Requirement Documents

Data Flow Diagram (DFD)

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system. It differs from the system flowchart as it shows the flow of data through processes instead of hardware.

A data flow diagram can also be used for the visualization of data processing (structured design).

It is common practice for a designer to draw a context-level Data flow diagram first which shows the interaction between the system and outside entities. The DFD is designed to show how a system is divided into smaller portions and to highlight the flow of data between those parts. This context-level Data flow diagram is then "exploded" to show more detail of the system being modeled.

Data flow diagrams were invented by Larry Constantine, the original developer of structured design, based on Martin and Estrin's "data flow graph" model of computation.

Data Flow Diagrams (DFDs) are one of the three essential perspectives of Structured Systems Analysis and Design Method SSADM. The sponsor of a project and the end users will need to be briefed and consulted throughout all stages of a system's evolution. With a dataflow diagram, users are able to visualize how the system will operate, what the system will accomplish, and how the system will be implemented. The old system's dataflow diagrams can be drawn up and compared with the new system's dataflow diagrams to draw comparisons to implement a more efficient system. Dataflow diagrams can be used to provide the end user with a physical idea of where the data they input ultimately has an effect upon the structure of the whole system from order to dispatch to restock. How any system is developed can be determined through a dataflow diagram.

Developing a Data flow diagram helps in identifying the transaction data in the data model.

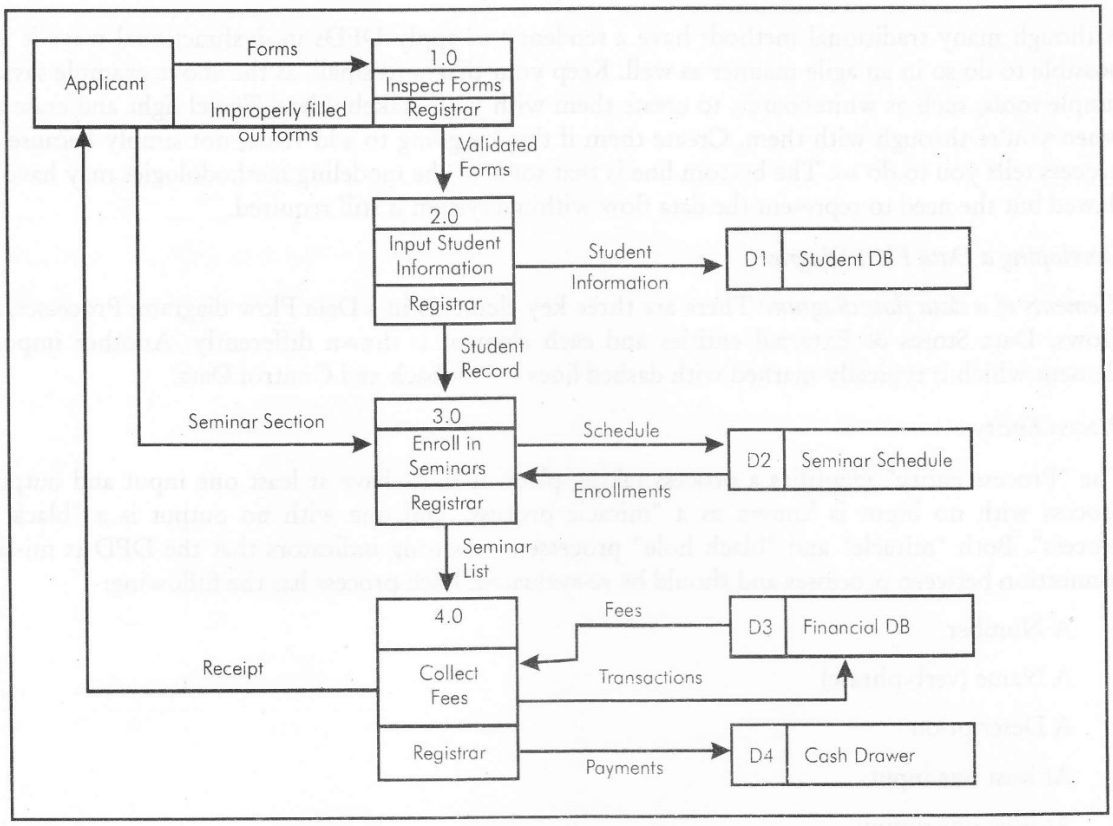
There are different notations to draw data flow diagrams, defining different visual representations for processes, data-stores, dataflow, and external entities.

Let us see the concept of DFD through a usage scenario, in this case the use case logic described in the Enroll in University system use case. On actual projects it's far more common just to stand at a whiteboard with one or more project stakeholders and simply sketch as we talk through a problem.

In this case start with the applicant, the external entity in the top left corner, and simply followed the flow of data throughout the system. Now introduce the Inspect Forms process to encapsulate the initial validation steps. Again assign this process identifier 1.0, indicating that it's the first process one the top level diagram. A common technique with DFDs is to create detailed diagrams for each process to depict more granular levels of processing. Were to do this for this process, lets put number the subprocesses 1.1, 1.2, and so on. Subprocesses of 1.1 would be numbered 1.1.1, 1.1.2, and so on. At this stage we shouldn't bother to expand this process to more detailed DFD as it is fairly clear what is happening in it and therefore the new diagram wouldn't add any value. The diagram also indicates who/what does the work in the bottom section of the process bubble, in this case the registrar. This information is optional although very useful in our experience. One can see how the improperly filled out forms are returned to the applicant if required.

Then continue to follow the logic of the use case, concentrating on how the data is processed by each step. The second process encapsulates the logic for creating a student record, including the act of checking to see it the person is eligible to enroll as well as if they're already in the database. Notice

how each data flow on the diagram has been labeled. Also notice that the names of the data change to reflect how it's been processed.



Now look closely at the diagram the arrow between the Input Student Information process and the Student DB data store should be two-way because this process searches the database for existing student records. Unfortunately I've erased this diagram from my whiteboard so it isn't easy to address this minor problem. Yes, I could use a drawing program to update the arrowhead but its more important to make the point that agile models don't need to be perfect, they just need to be good enough. AM recommends that you follow the practice Update Models Only When it Hurts and in this case this issue doesn't hurt enough to invest the two or three minutes it would take to fix the diagram.

The Collect Fees process is interesting because it interacts with an electronic data store, Financial DB, as well as a physical one, Cash Drawer. DFDs can be used to model processes that are purely physical, purely electronic, or more commonly a mix of both. Electronic data stores can be modeled via data models, particularly if they represent a relational database. Physical data stores are typically self explanatory.

There are several common modeling rules that to be followed while creating DFDs:

- All processes must have at least one data flow in and one data flow out.
- All processes should modify the incoming data, producing new forms of outgoing data.
- Each data store must be involved with at least one data flow.

- Each external entity must be involved with at least one data flow.
- A data flow must be attached to at least one process.

Although many traditional methods have a tendency to apply DFDs in dysfunctional ways it is still possible to do so in an agile manner as well. Keep your diagrams small, as the above example says. Use simple tools, such as whiteboards, to create them with your stakeholders. Travel light and erase them when you're through with them. Create them if they're going to add value, not simply because your process tells you to do so. The bottom line is that some of the modeling methodologies may have been flawed but the need to represent the data flow within a system is still required.

Developing a Data Flow Diagram

Elements of a data flow diagram: There are three key elements in a Data Flow diagram; Processes, Data Flows, Data Stores & External entities and each element is drawn differently. Another important element which is typically marked with dashed lines is Feedback and Control Data.

Process Entity

The "Process entity" identifies a process taking place, it must have at least one input and output. A process with no input is known as a "miracle process" and one with no output is a "black hole process". Both "miracle" and "black hole" processes are strong indicators that the DFD is missing a connection between processes and should be re-evaluated. Each process has the following:

- A Number
- A Name (verb phrase)
- A Description
- At least one input
- At least one output
- Data flow entity

The "Data Flow entity" identifies the flow of data between processes, data stores & external entities. A data flow cannot connect an external entity to a data source; at least one connection must be with a process. There are also "physical" flows, i.e. those that use a physical medium, like a membership card. Each data flow has the following:

- A Name (Noun)
- A Description
- One or more connections to a process.

Data Store Entity

The "Data Store entity" identifies stores of data, both manual and electronic. Electronic or "digital" stores are identified by the letter D, and manual filing systems by the letter M, e.g. D1 could be a MySQL database, and M4 could be a filing cabinet. Each data store has the following:

- A Number
- A Name

- A Description
- One or more input data flows.
- One or more output data flows.

External Entity

The "External Entity" identifies external entities which interacts with the system, usually clients but can be within the same organization. Examples of an external entity include customers, suppliers, management, certification agencies and competitors. Multiple existences of the same entity, e.g. the same doctor shown twice on the same diagram, can be identified by a horizontal line in the top left corner of the symbol. Each external entity has the following:

A Name (Noun)

A Description

The Feedback and Control data

The "Feedback and Control data" identifies a special purpose. Only the first four elements are needed to create a data flow diagram (DFD).

Top-Down Approach

The system designer makes "a context level DFD", which shows the "interaction" (data flows) between "the system" (represented by one process) and "the system environment" (represented by terminators).

The system is "decomposed in lower level DFD (Zero)" into a set of "processes, data stores, and the data flows between these processes and data stores".

Each process is then decomposed into an "even lower level diagram containing its sub-processes".

This approach "then continues on the subsequent sub-processes", until a necessary and sufficient level of detail is reached which is called the primitive process (chewable in one bite).

Event Partitioning Approach

This approach was described by Edward Yourdon in Just Enough Structured Analysis.

Construct detailed Data flow diagram.

The list of all events is made.

For each event a process is constructed.

Each process is linked (with incoming data flows) directly with other processes or via data-stores, so that it has enough information to respond to a given event.

The reaction of each process to a given event is modeled by an outgoing data flow.

Data Flow Diagram Levels

Context Level

This level shows the overall context of the system and it's operating environment and shows the whole system as just one process. It does not usually show data stores, unless they are "owned" by external

systems, e.g. are accessed by but not maintained by this system, however, these are often shown as external entities.

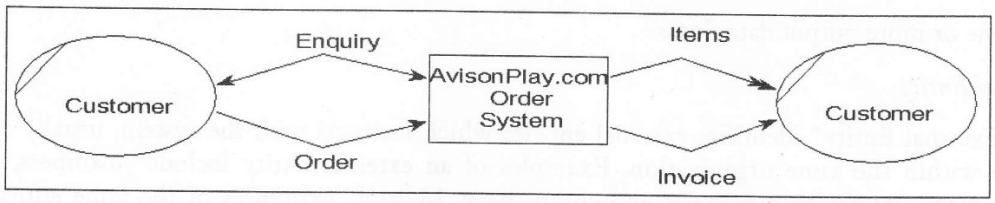


Figure 4.1: A context level Data flow diagram created using Select SSADM

Level 0

This level shows all processes at the first level of numbering, data stores, external entities and the data flows between them. The purpose of this level is to show the major high level processes of the system and their interrelation. A process model will have one, and only one, level 0 diagram. A level 0 diagram must be balanced with its parent context level diagram, i.e. there must be the same external entities and the same data flows, these can be broken down to more detail in the level 0, e.g. the "enquiry" data flow could be spilt into "enquiry request" and "enquiry results" and still be valid.

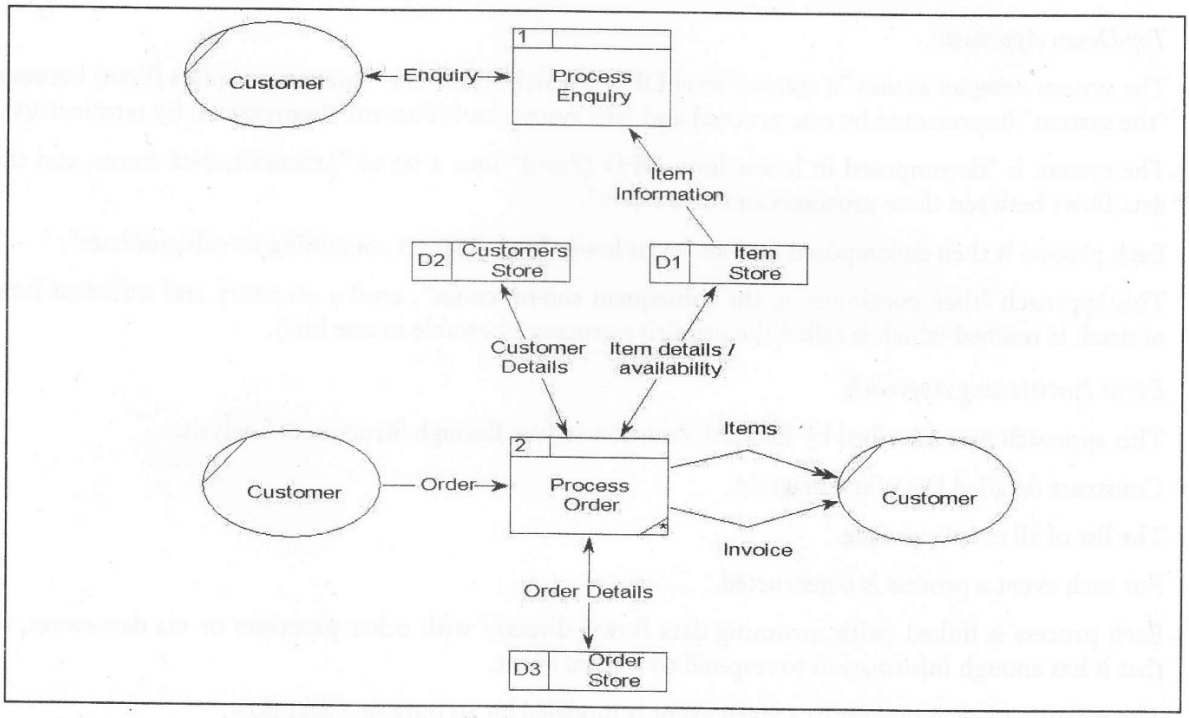


Figure 4.2: A Level 0 Data flow diagram for the same system

Level 1

This level is a decomposition of a process shown in a level 0 diagrams, as such there should be a level 1 diagram for each and every process shown in a level 0 diagram. In this example processes 1.1, 1.2 & 1.3 are all children of process 1, together they wholly and completely describe process 1, and combined

must perform the full capacity of this parent process. As before, a level 1 diagram must be balanced with its parent level 0 diagram.

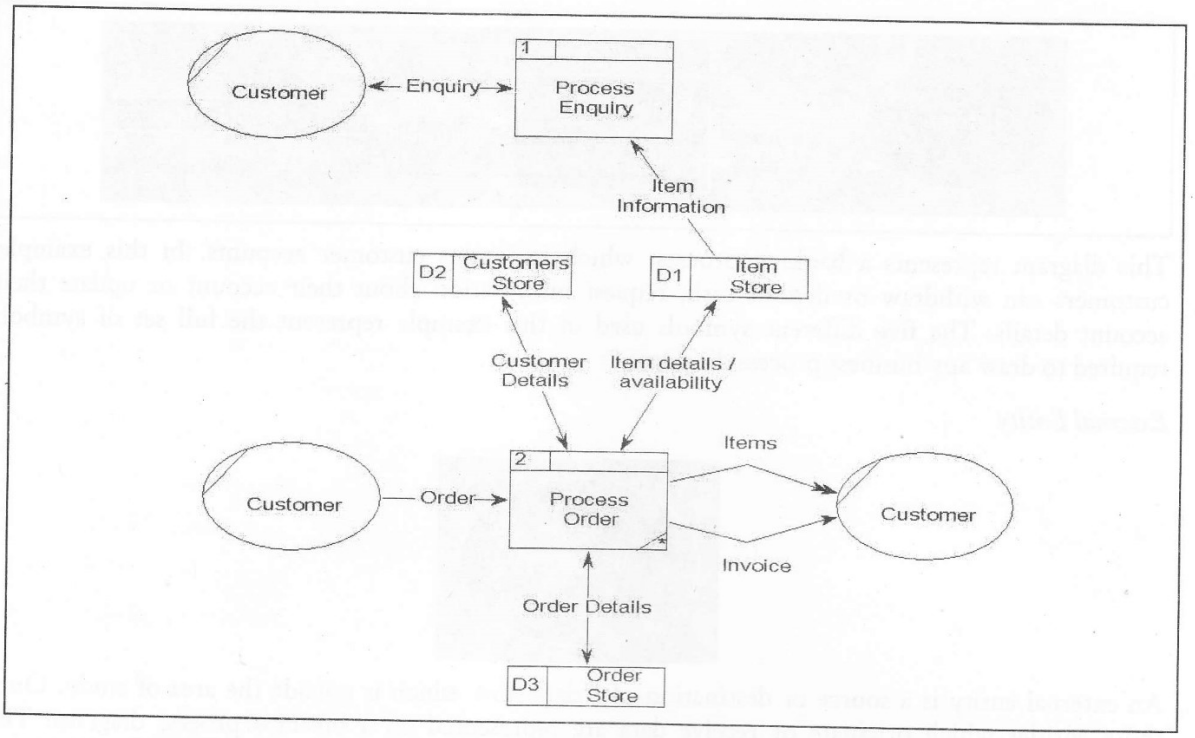
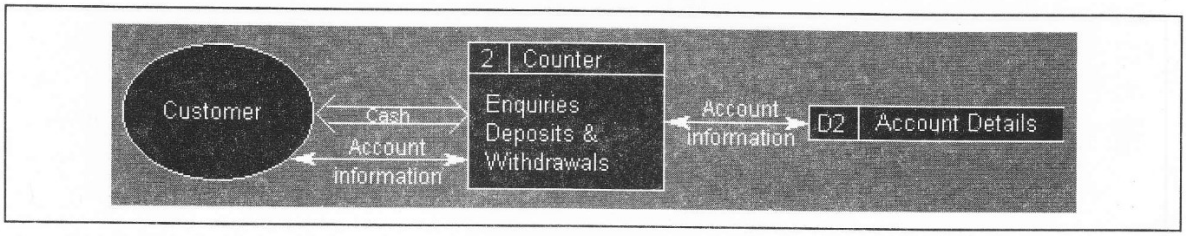


Figure 4.3: A Level 1 Data flow diagram showing the "Process Enquiry" process for the same system

Data Flow Diagram Tools

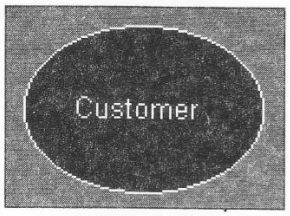
- CA ERwin Data Modeler, a data modeling tool
- ConceptDraw, a Windows and Mac OS X data flow diagramming tool
- Dia, a free source diagramming tool with flowchart support
- Kivio, a free source diagramming tool for KDE
- Microsoft Visio, a Windows diagramming tool which includes very basic DFD support (Images only, does not record data flows)
- SmartDraw, a Windows diagramming tool with Yourdon and Coad process notations and Gane and Sarson process notation
- System Architect, an enterprise architecture tool, supporting Coad/Yourdon, Gane & Sarson, Ward/Mellor, and SSADM notations and techniques
- DFDdeveloper, an open source software application that allows Microsoft Office users to create interactive leveled data flow diagrams and data dictionaries
- Flow-based programming diagramming tool (DrawFBP), developed by J. Paul Morrison.

There are only five symbols that are used in the drawing of business process diagrams (data flow diagrams). These are now explained, together with the rules that apply to them.



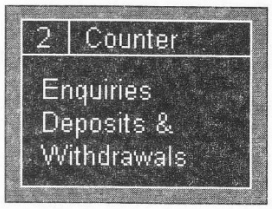
This diagram represents a banking process, which maintains customer accounts. In this example, customers can withdraw or deposit cash, request information about their account or update their account details. The five different symbols used in this example represent the full set of symbols required to draw any business process diagram.

External Entity



An external entity is a source or destination of a data flow which is outside the area of study. Only those entities which originate or receive data are represented on a business process diagram. The symbol used is an oval containing a meaningful and unique identifier.

Process



A process shows a transformation or manipulation of data flows within the system. The symbol used is a rectangular box which contains 3 descriptive elements:

Firstly an identification number appears in the upper left hand corner. This is allocated arbitrarily at the top level and serves as a unique reference.

Secondly, a location appears to the right of the identifier and describes where in the system the process takes place. This may, for example, be a department or a piece of hardware. Finally, a descriptive title is placed in the centre of the box. This should be a simple imperative sentence with a specific verb, for example 'maintain customer records' or 'find driver'.

Data Flow

A data flow shows the flow of information from its source to its destination. A data flow is represented by a line, with arrowheads showing the direction of flow. Information always flows to or from a process and may be written, verbal or electronic. Each data flow may be referenced by the processes or data stores at its head and tail, or by a description of its contents.

Data Store

A data store is a holding place for information within the system:

It is represented by an open ended narrow rectangle.

Data stores may be long-term files such as sales ledgers, or may be short-term accumulations: for example batches of documents that are waiting to be processed. Each data store should be given a reference followed by an arbitrary number.

Resource Flow

A resource flow shows the flow of any physical material from its source to its destination. For this reason they are sometimes referred to as physical flows.

The physical material in question should be given a meaningful name. Resource flows are usually restricted to early, high-level diagrams and are used when a description of the physical flow of materials is considered to be important to help the analysis.

Data Flow Diagrams - The Rules

1. ***External Entities:*** It is normal for all the information represented within a system to have been obtained from, and/or to be passed onto, an external source or recipient. These external entities may be duplicated on a diagram, to avoid crossing data flow lines. Where they are duplicated a stripe is drawn across the left hand corner, like this.

   The addition of a lowercase letter to each entity on the diagram is a good way to uniquely identify them.

2. ***Processes:*** When naming processes, avoid glossing over them, without really understanding their role. Indications that this has been done are the use of vague terms in the descriptive title area - like 'process' or 'update'.

   The most important thing to remember is that the description must be meaningful to whoever will be using the diagram.

3. *Data Flows:* Double headed arrows can be used (to show two-way flows) on all but bottom level diagrams. Furthermore, in common with most of the other symbols used, a data flow at a particular level of a diagram may be decomposed to multiple data flows at lower levels.
4. *Data Stores:* Each store should be given a reference letter, followed by an arbitrary number. These reference letters are allocated as follows:

   'D' - indicates a permanent computer file

   'M' - indicates a manual file

   'T' - indicates a transient store, one that is deleted after processing.

   In order to avoid complex flows, the same data store may be drawn several times on a diagram. Multiple instances of the same data store are indicated by a double vertical bar on their left hand edge.

Data Flow Diagrams - Relationship Grid

	External Entity	Process	Data Store
External Entity	✓	✓	✗
Process		✓	✓
Data Store			✓

There are rules governing various aspects of the diagram components and how they can relate to one another.

1. *Data Flows:* For data flows the rules are as follows:

   Data flows and resource flows are allowed between external entities and processes. Data flows are also allowed between different external entities. However, data flows and resource flows are not allowed between external entities and data stores.
2. *Processes:* For processes the data flow rules are as follows:

   Data flows and resource flows are allowed between processes and external entities and between processes and data stores. They are also allowed between different processes. In other words processes can communicate with all other areas of the business process diagram.
3. *Data Stores:* For data stores the data flow rules are as follows:

   Data flows and resource flows are allowed between data stores and processes. However, these flows are not allowed between data stores and external entities or between one data store and another. In practice this means that data stores cannot initiate a communication of information, they require a process to do this.

Data Flow Diagrams - Context Diagrams

The context diagram represents the entire system under investigation. This diagram should be drawn first, and used to clarify and agree the scope of the investigation.