

# Practical Approach to Computer Systems Design and Architecture

By [Nirosh Liyanawaduge](#)

The article gives an introduction to system designing and architecture by introducing a new concept that allow beginners to easily break and design complex software systems using a modular based development approach with a pattern, which we all practice in our day to day life.

## Table of Contents

1. [Introduction](#)
2. [Overview](#)
  - 2.1. [What Solution Providers Do Today?](#)
  - 2.2. [Why We Should Design Systems Properly? Do I have to answer this?](#)
3. [The Approach to Design a System](#)
  - 3.1. [Breaking the System with Easy Concepts](#)
4. [Start Designing our Document Management System](#)
5. [General Advices](#)
  - 5.1. [Object Oriented Programming \(OOP\) Concepts](#)
  - 5.2. [Design Patterns](#)
  - 5.3. [Modular Based Development and Reuse of Modules](#)
  - 5.4. [Quickly Develop the Data Access Layer \(DAL\)](#)
  - 5.5. [Group Operations into Classes](#)
  - 5.6. [Keep Front Layer Free](#)
  - 5.7. [Visualize the System](#)
  - 5.8. [Naming Convention](#)
6. [Think Fresh and Approach like a Newcomer](#)
7. [Additional Hints for System Designers](#)
8. [Final Note](#)
9. [Coincident](#)
10. [Summary](#)
11. [History](#)
12. [References](#)

## 1. Introduction

Our Business Analyst went abroad last week to meet a new customer who selected us to develop his Content Management System. The Analyst had met and extensively discussed details about the business needs with the customer. Our hard working Business Analyst had returned after two weeks with on hand detail Business Model document, where he together with a coworker prepared the formal requirement specification in a hurry, during this process they conferred with the customer to clarify some requirements. Formal requirement document was sent to the customer for initial approval, where it was returned with minor adjustments. The requested minor adjustments were made by the business analyst himself to get the final approval from the custom end. It was a good effort; we got the fixed set of requirements after three weeks, since we first met the customer. Business Analyst directed the business model to system architect. Soon, the Non-Functional Requirement Document was finished and was approved by the customer with no

changes. As the next immediate process, the user cases were defined. System Architect finalized the design of the system together with the System Analyst. The Database Admin was followed by identifying/ defining the entities and their relations together with DDL (Data Definition Language) of the system. The UI (User Interface) designer designed the system user interface to finalize the initial design effort. At last, the deployment model was defined and started implementing the system steadily, after getting all required approval from the customer end. What I just mentioned, was an imaginary system development process, which has limited real world applicability. In real world practice, there are many variables for one to feel that system design/ development methodologies needs to be adjusted, from customer to customer, as well as from project to project. This article is a challenging attempt to introduce a concept (model) that hold onto all extreme cases of modern day system designing requirements.

Designing a system needs thorough study of the problem domain. This implicitly means time, followed by money, which threaten the customer, who doesn't have an adequate visibility over the advantages of a properly designed system. It just makes things worse, when the current day diversity of requirements argue the value of one's experience. If one soon says that "Oh, This is just the same solution we delivered last year" he'll soon be ended up loosing the customer by delivering the wrong product.

There are good design methodologies out there, but often due to time limitations (where client push for quick releases, while developer struggle with extremely tight deadlines) solution providers aren't confident to apply them. Today, Solution Providers tend to implement systems without designing them properly. More often, the development starts after loosely evaluating the depth of the project. The System Requirement Specification is often changed at the middle of the development. This is due to the insufficient domain expertise and clarity of business requirements, had at the start of the project. The customer often waits till solution provider releases the system to understand it. The customer uses the first few releases of the system to understand the direction of the project and to apply corrections. Unfortunately, some of these changes lead to shake the whole foundation of the system, forcing designers to rethink about the initial design. But these impacts are hardly being seen by the customer's non-technical eye. A fight, between customer's money versus developer's 24 hours a day.

The customer estimates the correct time to enter the market at the initial stage of the project. The massive competition at the software market place makes it important to take decisions to pin-point accuracy. The most important is to hit the market on time.

## 2. Overview

### 2.1 What Solution Providers Do Today?

You don't need civil engineering expertise to understand the instability of a building foundation put up, not knowing the number of stories of the finished building. The same philosophy applies to software architecting as well, expanding a system, which is developed for a limited set of features is just adding more problems than features. Higher percentages of solution providers push for add-hoc development approaches,

with extreme programming to achieve tight deadlines. The customers are also encouraging them, having that daydream of re-factoring the system, once it is running at a profit earning stage. Another reason for customers to treat system designing a low priority is the time the designers/ analyst take to investigate/ design a system. The Customer, according to some terminology is the GOD, regularly expects tangible outputs from the solution provider, but a proper system designing process may make him wait for several months. In order to support this customer's need, solution provider often plans by weekly or monthly system releases from starting day of the project. This extreme need is burnout resources at the middle stage of the development.

The developers have used to start the system development process, with a scale down version of the system and gradually patch rest of the system around it, which result an unstable product with lots of repetitive, badly grouped codes. Additionally, it creates programmer dependant code modules, since each module is coded by individuals with a different skill set. The inconsistent code makes debugging of the system a nightmare. This costs time as well as money, in many direct/ indirect ways, while making it impossible to expand.

## 2.2. Why We Should Design Systems Properly? Do I have to answer this?

Properly architected systems always gain that strategic advantage over common hiccup of the software development life cycle. A well architected system is relatively cheaper when it comes to dealing with changes and upgrades. The architectural discipline regularizes the code, while supporting to manage the implementation with short predictable development times. The consistency of the design allows managers to pull/ push resources from/ to different sections of the project with minimal training. These are some of the few advantages; you gain with a properly designed system.

In order to drive the system development effort smoothly, the designer should introduce a steady design/ coding pattern at the early stage of the process, allowing every developer to gain mastery over it. This also opens the opportunity to remove the architect's dependency, at later stage of the project.

## 3. The Approach to Design a System

Let me highlight first, that I do not draw the famous user case diagram and sequence diagram, when designing systems. In-fact I have another approach that let you understand the system better and directly draw the class diagram. But before drawing the class diagram, I encourage you to draw an activity diagram and/ or a system overview diagram and/ or a module interaction diagram and/ or any other type of diagram, which help you to fully understand (feel) the system.

As I reiterated many times, designing a system mainly depend on designer's understanding of the problem domain, so the better you understand, easier the system design would be. However in modern days, it is less probable to assume that designers get a sufficient time frame to fully investigate the system, before starting to design.

As the first step, the designer has to be a master of the problem domain. A well written System Requirement Specification (SRS) document can be used as the introductory to the system. The system designer should read it repeatedly and should carefully understand each and every important feature, hidden in odd corners of the document. But in most cases, the designer does not get a proper SRS prior to the system designing (At least, this is the case with service base companies). Instead he is forced to grab the requirement through the initial business meetings and telephone conferences. This unfortunate circumstance is the ideal case for designer to think of a throw away prototype or even two. The prototype is the best and the cheapest way to define the product specification, before committing yourself to an unknown territory. In case of a disagreement with prototyping the system, the designer should strategically drive the customer to write the specification for him. The traditional advice is to wait, till you get to the depth of the system, before starting the design. But in practice, you can speed up yourself by referring to online resources, such as articles, open source projects and other similar products without heavily depending on the SRS document (Note: New comers always need help, from seniors, to correctly identify the resource pools, since the complex business requirements are hardly seen by new technical eyes.). The domain expertise, you gain, will not only guarantee a good design, but also will help to better predict the future of the system and to keep adequate spaces for later expansions. In addition to this, new comers have to be attentive not to underestimate important business requirements by evaluating them from technical angles. This is another crucial area, where client could be extremely rigid and even would decide to reject the system, complaining that solution provider delivered a wrong product.

Discover the system by asking questions. I have a technique that I used, since the very first system, I designed. Here, I am presenting it for you, as the second step of approaching to design a new system. Open a Notepad or get a piece of paper, and then start asking question about the system. In this effort you are free to ask any question, but in order to get a better start, start the process with the three main questions (about input, process, and output of the system) listed bellow.

1. What are the Input(s) of the system?
2. What are the Processes of the system?
3. What are the Output(s) of the system?

In order to optimize the throughput, start to rethink the system with a fresh mind by forgetting all the information you gathered through various resources. This approach will help you to understand the definitions of the product, even when the product specification is not very clear. As you practice this technique you will get surprised to see the way it opens up new areas of the system. As you may already know, according to this method, interestingly, the questioner has to be the answerer and it has to be you. But some time you may not be able to answer a specific question. But at least you will end up having the list of questions, to find answers to clear the way. I have seen designers are struggling, when they are thrown at project with a widely (loosely) defined requirements. They do not know where to start and what to/ not to ask. If you are facing the same issue, then have a go with this technique and see.

A better question would be one that creates several derivative questions by the answer given to it. So an answer to a question may produce another question (one or more) or a leaf level operation or function of the system (use case of the system). This technique does not have rules or definitions. The questioning and answering

process is drifted freely in multiple directions by the questions created from answers. The questions that are not answerable may need to be directed to the client, and those, which are not answered by the client has to be creatively handled by the system designer in a way, so that it doesn't affect the solidity of the system. You can creatively group them into separate module(s) for later implementations.

As an example, let's try to apply this methodology to discover the feature list of a simple Document Management System (DMS). The questioning will start at a very abstract level with the three main questions.

1. What are the inputs for this?  
Files such as word, txt, media etc (I would say any type of files)
2. What are the processes of the system?  
It is a document management system and it will help my client to manage his documents properly.
3. What are the outputs?  
Allow user to view file(s) online as well as allow them to checkout/ remove file(s).

If you get fairly good answers to all three questions above, I would say that you have understood the system. The above three questions, virtually can be used to discover any system. The answers, you write to these three questions will create many more derivative questions to continue. This process will help us to uncover all use-cases or rather in simple term, leaf level functions of the system (A leaf level function is an independent, granular level operation that describes a specific function of the system) as well as all external/ internal (system) actors of the system. The process of asking questions and writing answers will continue, until we get to leaf level functions, where we cannot ask any more questions about the answers.

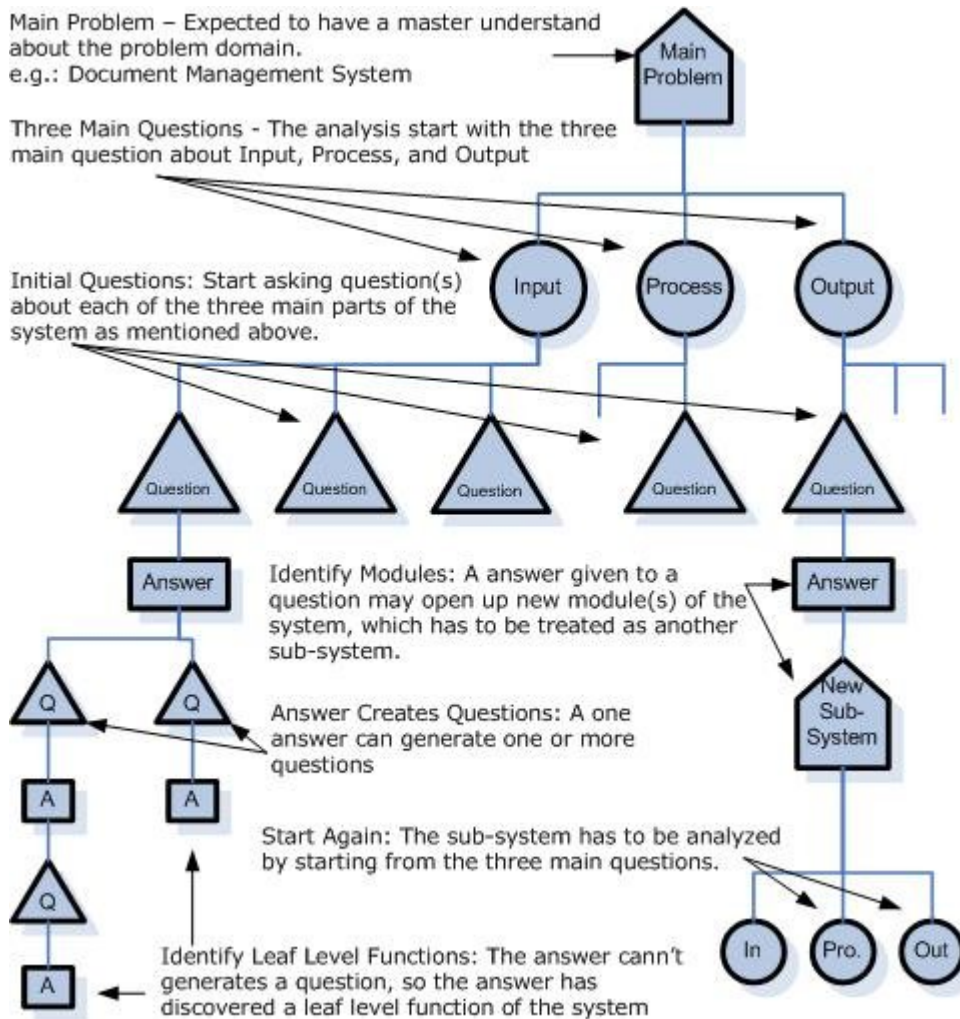


Figure-1: A graphical representation of a generic system is being analyzed by asking questions

Let's go and analyze the first question (i.e about inputs) little more to see, what it will discover for us. Just allow what, why, how etc to create questions for you. In the sample below, I have used '\*' to indicate the words that create questions. You may also follow the numbering sequence to better recognize the way it drifts.

1. What are the inputs for this?

Files such as word, txt, media etc (I would say any type of files).

2. How users upload a document?

The \*User\* gets a \*web interface\* to browse, select and upload a document

2.1 How users reach to this web interface?

The user has to be \*registered\* with the system to access this via the provided credentials

2.1.1. How users get registered with the system?

All \*employees\* will be registered by the DMS system administrator. This will happen as they deploy the system.

2.1.1.1. How admin knows the employee's profile, and how employees obtain their credentials?

Both of these will be manual processes, where Admin will obtain employee profile data from the HR department and Employee will get their access code by calling System Administrator manually.

Note: Here we found that System admin \*only\* can create users and obviously he can delete/ update them.

2.1.1.1.1. What about employee's privacy?

The system should allow employee to change their passwords.

2.1.2. Is the User Registration page a restricted one for general users?

Yes, System Admin will only see this page.

2.2. Who is a user anyway?

A user of this system \*can be from four different divisions\*, namely Administrative, Account, Marketing and Development. Each user will only belong to one of these four divisions. So this means that user of this system will have different privileges and access permissions.

2.2.1. Do users of different divisions need to have different privileges?

Yes. The Users of each division will have different privileges. A user from Marketing or Development division will have \*default privileges\* where they can upload and manage their documents but users from account division can \*over look\* marketing division and administrative division can over look any other divisions, apart from doing default functions of the system. (Administrative > Account > Marketing = Development).

2.2.1.1. What is a default privilege of a user?

This includes upload documents, view documents online, check out documents for editing and delete a document. Again any document uploaded will be shared among users of the same division automatically. A document belong to a one division can be shared with another division only by explicitly giving permission. A private document will not be supported by this system (confirmed by the client software requirement specification).

Note: I will not create question(s) out of this.

2.2.1.2. What is over looking a user?

This is the privilege of accessing documents uploaded by other users without needing any permission from the document owner.

Note: I will not create question(s) out of this.

Note: Above answers introduce us two modules called User Management, Security and Privileges. In order to analyze them further, you have to treat them separately by asking questions starting from input, process, and output of each module. Let's not dig in to that section right now.

3. How a documents flow through this system?

\*Select\* and \*upload\* the document by the user > System will validate the input file > \*Files added\* to "to be approved" section > Administrator \*Approve and publish\* the file > File is available for users.

### 3.1. Do we need a work flow management system here?

I don't think so, let's hardcode the work flow, since we don't expect them to be dynamically adjusted. This is not a very complex system to have a separate module to manage workflows, I am happy without it.

Now, we have found few granular level operations belonging to the document uploading section. Let's list them as below..

Client side:

- Select a document from the local directory base
- Read the byte stream to system memory
- Send it via the web to the server

Server Side:

- Read the byte stream from the memory
- Write it to a local temporary location for validations.
- Input Validation Module validates the input file.
- Move the validated file to the correct user's shared location.
- Add this new file to the repository system as a "File to be Approved".
- Load the files that are to be approved for the system administrator to view them.
- System administrator validates the content of the file and approves them to be viewed online.
- Change the file metadata status from "tobeApproved" to "Approved"

Note: Ideal place for a diagram..

### 3.2. Are we going to have versioning here?

Yes, we need to have versioning. So any updating to an existing file will run through a version handler that assigns the correct version number.

Note: This can be analyzed further deep..

### 3.3. Do we need to notify users, when a new document is successfully published?

The user, who own (up-loader) the document will get an email noting the reject/accept state. In addition to that we have to send another notification to users who have subscribed to be notified as this operation happens.

Note: This introduces a new module to the system named Notification Module. I will not dig in to this section either.

### 3.3. Can a user undo an update?

I need to talk to the client about this.

### 3.4. Can multiple user checkouts one document at the same time?

Let's stop this now...

Look at the way it expands, it just keeps on expanding as you ask questions. I omit lots of the questions, since the article is getting longer, but I believe that this sample question and answer set will provide enough guidance to understand the concept. This process needs to be repeated (go through the questioning and answering process again and again) several times to fully discover the system. Firstly, you have to recognize the main modules of the system, I have already identified a couple of modules named input validating module, user management module, security module, notification module and not the last but the least the main document

management module. The module reorganization is one of the important parts of this technique. If you can identify the modules of the system quicker, that will lead to an easy designing. The ones, who do not have experience, will always find it harder to recognize the modules of the system at once. But as you practice the technique several times, you will gain that control over it (This comes with the experience, and to get it quicker, you can apply the technique over and over on the same system, until you feel comfortable with the module break down).

In my early days, I always started by recognizing the full list of leaf level functions of the system (granular level use cases of the system). Then it followed the way by grouping the functions to form classes (classes will group same types of functions/ operations). These classes are the ones that form the modules of the system. As I reach this point, the deep understanding automatically guides me to identify the modules of the system. I recommend new comers to take that path until you gain mastery over the concept. The recognized modules have to be further analyzed by asking questions (as explained above) starting from the three main questions about the input, process and output. The module has to be truly object-oriented and independent. They have to have clear definition and should fully cover the specific section of the system. They also have to have a well defined functionality rich communication channel(s) for external party communications.

Once you design a couple of systems, the module breakdown will become easier and the concept will become friendlier. This experience will also reveal you that some of those modules are repeatedly coming in every other system you design, so that you can easily recognize/ reuse them.

You may follow this process to identify all the leaf level functions and main/ sub modules of the DMS system.

### 3.1. Breaking the System with Easy Concepts

Think of a governing body of a country or a private organization, and how they are being managed. Both of these complex organizations have hierarchical structures with many smaller departments (divisions) link to the top. The more important finding is that they are proved to be rock solid. In this application designing technique, we treat the software system as a real world organization. The careful study found that all software systems can be easily mapped to a real world organization. This mapping helps to easily identify/ define modules and also to control the system designing effort throughout the project life cycle. When you have that luxury of visualizing a software system via a pure manual real world organization (more friendly) you can quickly and accurately respond to requirement changes, without hurting the stability of the system. When a change to the software system is proposed, virtually apply it to the manual system first, and it will help to identify the best way/ place to apply it in the software system. This approach will better suits for complex systems, but will obviously suite for simple systems too. In order to successfully map a software system to a real world organization, you need to correctly analyze the software system in a way, so that it explains, how it would process in a pure manual environment. The next step is to automate that manual process with a software model. It is highly recommended to keep similar modules, communication pattern, rules and naming convention in between manual system and software system for better visualization.

Let's have a look at a typical organization as bellow and try to see, how it would respond to a typical request.

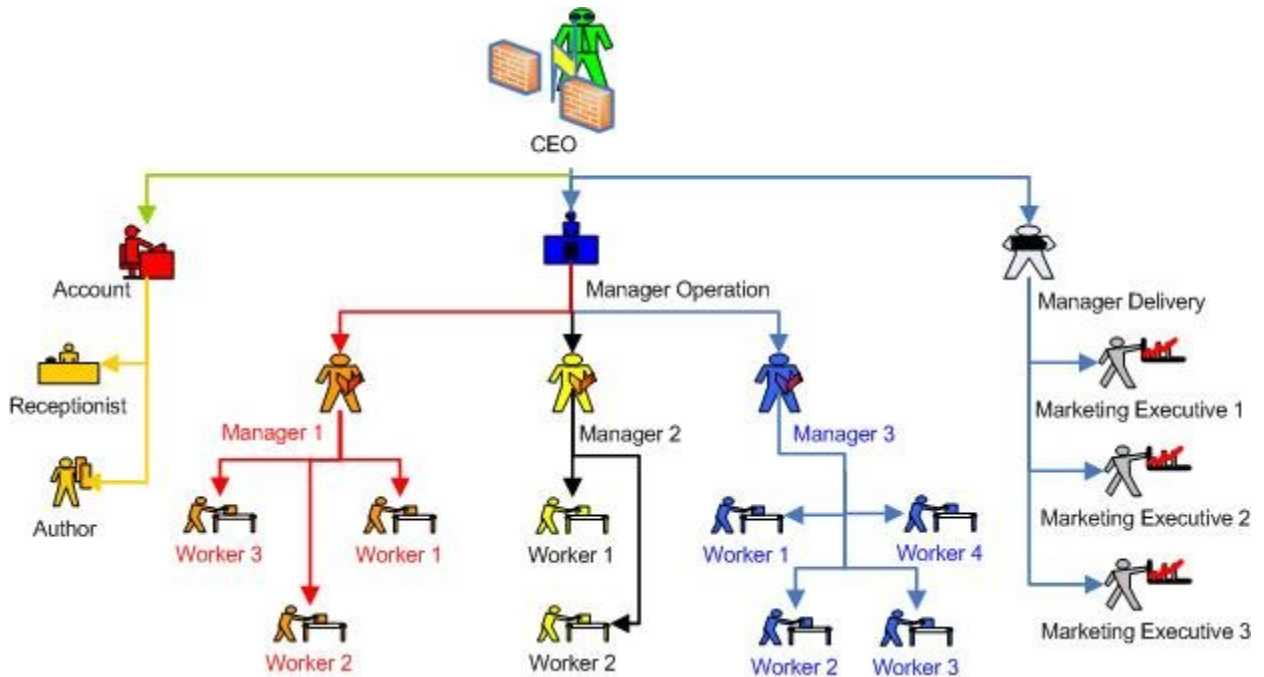


Figure-2 Showing a typical organization hierarchy

CEO – Chief Operation Officer of this system is acting as the front end (User Interface of the software system) of the organization. The main responsibility of the CEO is to interact with out side and identify/ guide managers in the correct sequence to complete a task. The CEO will directly map to the front interface of our sample Document Management System (DMS).

Super senior, senior and junior Managers – The manager's duty is to utilize resources (workers or sub managers) in the correct sequence to complete a particular task. A complex organization may have several levels of managers (super senior managers, senior managers and junior managers). Higher level managers will use one or many lower (immediately) level managers to complete a task. According to our diagram, we have three senior level managers named Accountant, Manager Operation, and Manager Delivery, where each manger is given a set of junior managers to perform their duties. As you can see, all junior managers are equipped with group of workers.

Workers – Workers do all granularly level operations and more often, workers are given pure independent tasks, which they can perform without making any dependency over any other task or worker.

Rules of the Organization –

- Higher level managers have comprised with the knowledge of the capacity of junior level worker groups. Hence, they know what worker to pick, in order to complete a task.

- Same level entities are not allowed to communicate with each other, but if a task is needed two or more same level managers to complete, then that task will be handled by the super manager, who is senior to the managers, who needed to be involved to complete that task.
- Any entity is only responsive to the immediately higher entity, and only utilizes the immediately lower entities.

In order to understand the organization well, let's see how this organization will respond to a typical request, made by the CEO.

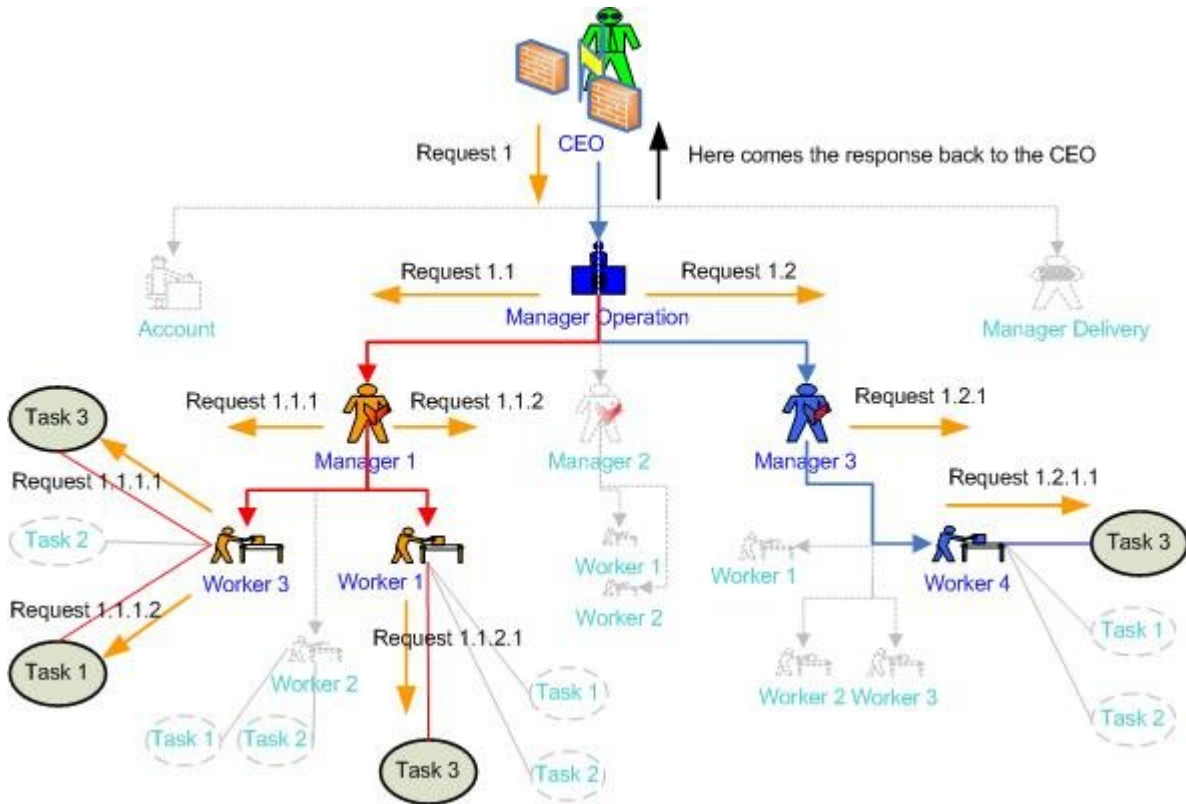


Figure-3 A request is been processed utilizing different types of workers of the organization

In this sample, the CEO of the organization has issued a request named "request 1" to "Manager Operation" (This operation assumes to be an independent one from Accountant and Manager Delivery). The CEO knows the manager, who he needs to issue the command to process this particular request successfully. In this case "Manager Operation" has received the request from the CEO and he has proceeded by making the request named "request 1.1" to "Manager 1". The "Manager Operation" and CEO are at waiting mode now while "Manager 1" is processing the first part of the request. The "Manager 1" has triggered a request named "request 1.1.1" to "Worker 3" to complete the "Task 3". As the "Manager 1" received the result of the "Task 3" he has issued the next request to complete the "Task 1" to the same worker. After getting the result, the "Manager 1" has used another worker named "Worker 1" to complete the "Task 3", which completes the "request 1.1.2" followed by "request 1.1". The "Manager 1" has responded to "Manager Operation" with result of "request 1.1". The "Manager Operation" has picked the correct junior manager to handle the second part of the main request i.e. "request 1.2". As a

result, "Manager 3" has received the "request 1.2" from the "Manager Operation", where he has responded to it by making the "request 1.2.1" to "Worker 4" to complete the "Task 3". Finally, "Manager 3" has responded to "Manager Operation" with the result of "request 1.2" which complete the whole processing of the "request 1".

In our system design approach, we will also use the same concept to break and design the software systems. You will first identify the granular/ leaf level functions of the system (Tasks of the system) and will group them in to classes, where each class is responsible for similar types of operations. These classes are pretty much similar to workers of the above diagram. As the number of workers of the organization increases, you add managers to manage the worker group and same will be done in software system too. Again as the number of managers grows, that group will be pushed down and few senior managers will be introduced to the top to control them. However during this process, you need to carefully group classes of the system to form modules as well (The modules can be formed by grouping similar types of classes). The number of managers and the depth of the manager pool inside a module can be decided by the complexity of the module (or the amount of worker class loaded or grouped in to the module). These modules will be treated as separate divisions of an organization, where each module will be controlled by one or more managers, positioned considering the complexity of the system. These module controlling managers will help modules to interact with each other.

#### 4. Start Designing our Document Management System

Initial analysis of the Document Management System has discovered a set of modules with their leaf level functions. As explained before, it is important to break the system into smaller modules before designing them; better you modularize the system, easier the system maintenance would be. So just following that concept, let's modularize our DMS system horizontally as well as vertically. This will allow us to design each module separately by treating each as a separate division of the main organization. Even though we didn't fully analyze the DMS system, I guess, what we have discovered is detailed enough to explain the concept with an example. As the third step of designing a system, we will draw a system architecture diagram as below. This diagram will help us visually abstract the system and understand the key modules with their interaction in our DMS system.

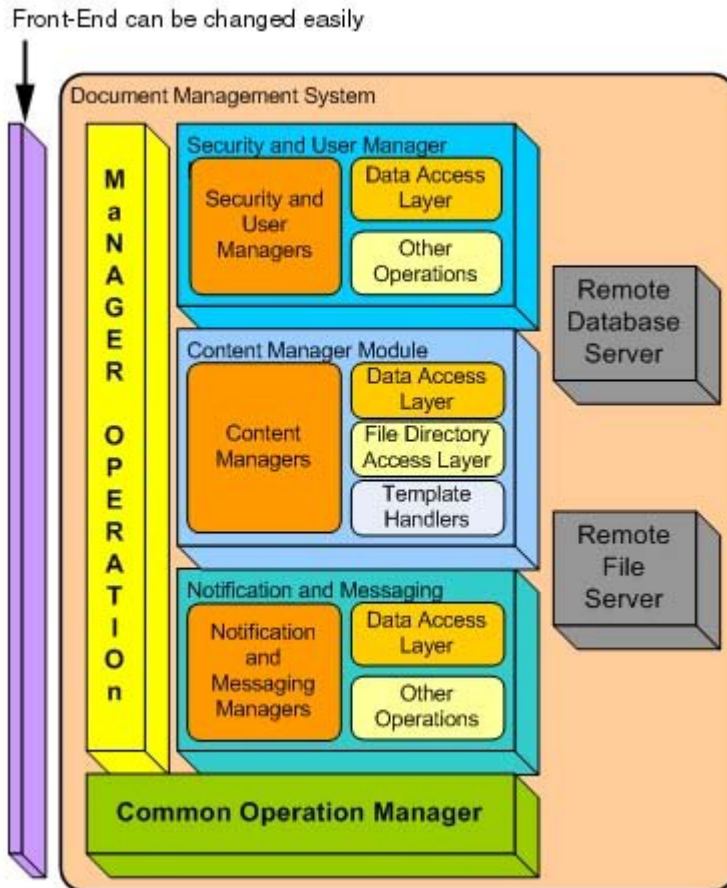


Figure-4: System Architecture Diagram of the Document Management System

Manager Operation – Manager Operation is acting as the head for all divisions, where divisions are the Document Management Division, Email/ Notification Division, and User Manager/ Security Division. The Manager Operation will coordinate all functions of the system while helping each division to interact with one another as and when it is appropriate.

E.g.:- Once a document is published, several emails needed to be sent, so then manager operation will first request the manger of the DMS division to publish the document and depending on the state of the publication the manager operation will request the email/ Notification division to send the correct email to the document owner and other subscribers. In this attempt the system has used two divisions to complete the task and these two divisions have being controlled by the Manager Operation.

DMS, Notification, User Manager/ Security – These are similar to three divisions of a generic organization. I have separated the leaf level workers in to three groups named Data Access Layer, Template Handlers and Other Operations. The DAL (Data Access Layer) is dealing with all database related operations such as get, add, update, and delete data, where as the section named "Other Operations" is responsible to any other leaf level operations as required.

Common Operation – This module does the common operations. A correctly designed common operation module can be easily reused in any system. This division is shared among all component or modules of the system. In real world organizations also you find common divisions, such as company library, company canteen, reception etc. In most cases, you may use this module to group classes that log exception/ transactions, store shared objects, handle errors etc.

Let's further analyze the User Management module of our Document Management System to see, how it support functioning three basic features of the user management module named add, edit and remove user(s) (Please refer to the class diagram below). According to the diagram, firstly, you have the system user interacting with the DMS interface. There, we only have one senior manager named "Manager Operation" to control the full system and a one junior level manager (to head the user management module). The manager named "User Manager" directly communicates with the junior manager of the sub module named DAL (Data Access Layer for the User Manager module). The data access layer is responding to the corresponding immediate manager only via the abstract class named "HandlerData". Inside the DAL you can see there are three classes to handle three basic types of data related operations named Edit, Add and Remove data (In this sample all handlers are same as workers of the above described organization). The drawing of the class diagrams completes the fourth and final step, of our approach to design the system.

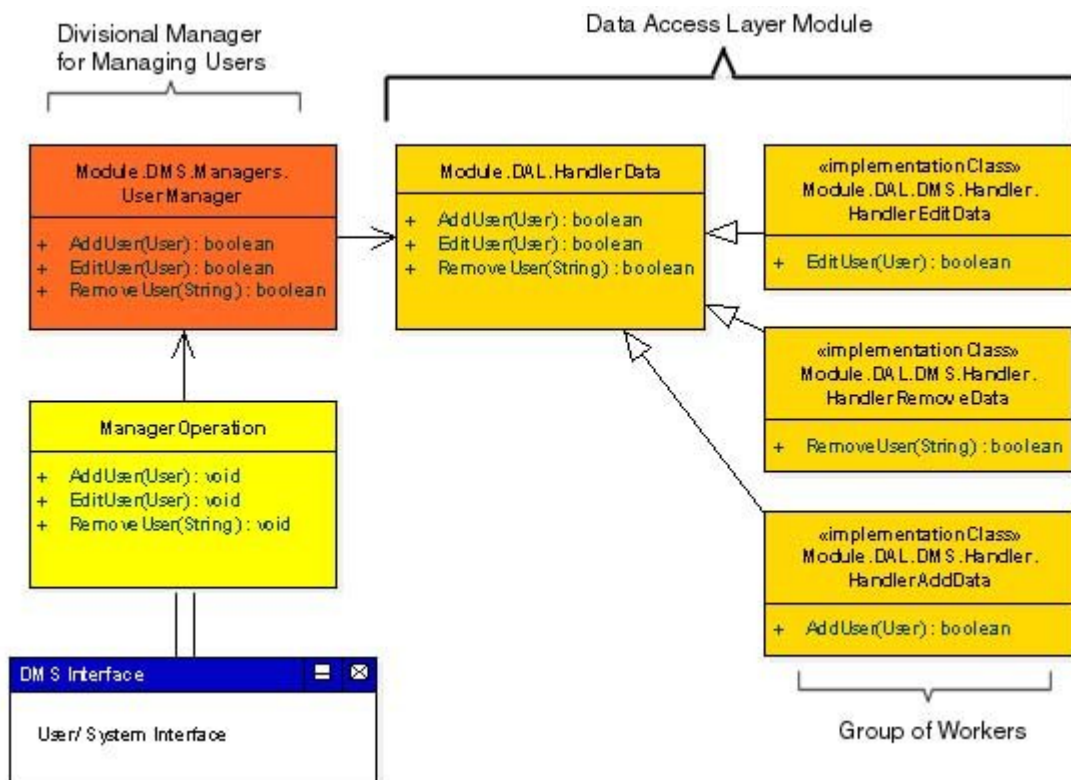


Figure-5: Class Diagram of Scale down Version of a UserManager Module

E.g.:- Adding a User: The user is requested to fill the registration form via the user interface. As the user click on the submit button, the UI side is expected to do all

required client side validation to verify the input. Then the user profile is stored inside the model class named "User" (where it can store user profile with correct entity relations) pass it to the main manager of the DMS system named "ManagerOperation" by invoking the method named "AddUser". The main operation manager correctly identifies the module to talk to complete the request. So it invokes the method named "AddUser" of the manager named "UserManager" (of the module named "UserManager") where that talk to the DAL (Data Access Layer) via junior manager class named "HandlerData" to store data in the database. Once this process is completed the "ManagerOperation" evaluate the status of the operation. Then, depending on the status, it will invoke the main manager of the notification module to send a welcome email to the newly registered user.

The analysis of this system shows that generic DMS system can be easily mapped to a real world organization. You can continue the process to build the whole DMS to form a organization that manage documents. This approach will produce a consistent system that has distributed its functionality across several modules, allowing easy maintenance. This method can be used to design any system, including a web site, web service, other types of services (windows service etc), form based application (windows form etc) or library, where as if you follow any other architecture/ design/ coding pattern, you may advice to use one method for web site and another to form based application and may be some thing totally new when it comes to any other type.

## 5. General Advices

### 5.1. Object-Oriented Programming (OOP) Concepts

Object-oriented programming (well known as OOP) is the concept of defining and combining independent objects to form a software system. It has four base techniques namely inheritance, encapsulation, polymorphism and abstraction. Today, almost all popular programming languages (such as C++, C#, Java, PHP, Ruby, Python etc) support OOP.

In my book, I recognize OOP as another unnoticed theory of the nature, which was waited till the right time of the information age. It was noticed in the 1960s and now is successfully using in the virtual form of the nature (so called software field). Just to understand the relation between the two, let's think of the functionality of a part of the human body (let's say a hand) and a well architected software module of a system. If you carefully study them, you will realize that both of these are provided with similar kind of communication interfaces where a set of standard instructions uses to drive the objects in the required manner. There is nothing new in OOP it strongly emphasize modularity in software (just like the nature does). It is something already known and experienced.

Having these things in mind, designers are welcome to use any object oriented concepts as and when they are appropriate in software systems. It will achieve the flexibility and maintainability of a complex software system. Please refer to online resources for more details on this topic.

### 5.2. Design Patterns

As the OOP concept become widely popular in the software world, the designers started encounter similar type of challenges in every other object-oriented design they do. Right on time, there comes a set of widely accepted solution to these challenging problems with the name "Design Patterns". The Design Patterns describes a set of recurring solutions to common problems in software design. This was originally described by a book written by four authors known as the "Gang of Four" or simply "GoF". Hence their pattern set was named as GoF patterns. Since then and still the design patterns are evolving.

The design patterns will brighten the design, so you should use them. Please refer to online resources to find more details about design patterns.

### 5.3. Modular Based Development and Reuse of Modules

The modules of a software system have to be treated as divisions of an organization. This means that each division has to have one or many managers, considering the complexity of the modules. The "module managers" help inter module communications and also help to bind them together to form the entire system. A bigger module can be broken into several sub modules, where each module has to be treated just like sub divisions of a division. It is recommended to have separate sub modules inside each module to handle leaf level operations such as accessing a database, accessing a file server etc. The modules can be designed using the expertise you have on various design patterns and object oriented programming concepts.

Properly design modules can be reused in other systems too. Some of the famous modules that can be reused are, Logging, Notifications, Exception, File Directory IO etc. As you start with this approach you will find it harder to reuse the modules, you will find that you have to enhance the modules to reuse it in every other system. This will happen until you correctly define the specification of the module or until you learn to design truly object oriented modules, but it is recommended to expand the functions of the module until they are rich enough. This should be a continuous process and that will gradually create powerful, more complete, and functionality rich modules, where you can reuse them in future projects.

Here, other than delivering a smart product to your customer, you can open up a new market, if you can develop extensible framework for all commonly use modules.

### 5.4. Quickly Develop the Data Access Layer (DAL)

It is important to separate the data access layer and quickly nailing it down. The Data Access Layer (DAL) consists of classes that directly operate with database, so that it is like the engine of the system. This separation will better modularize the system and also helps developers to edit the data access layer (which cause lots of changes at the early stage of the project) without hurting other part of the system. For more information about this section, you may refer to following three articles.

- Reference 1: <http://www.codeproject.com/cs/database/ModelCreator.asp>
- Reference 2: [http://www.codeproject.com/cs/database/CSharp\\_Wrapper.asp](http://www.codeproject.com/cs/database/CSharp_Wrapper.asp)

### 5.5. Group Operations into Classes

When you identify similar types of functions or operations group them together. In above example I have created three separate classes for Add, Edit, Delete operations (refers to Figure 5). This way you can keep consistence coding across the methods of the class, since all the methods of the class are doing similar types of operations. This way you can handle exception/ logging etc the same way for all the functions of the class.

## 5.6. Keep Front Layer Free

User Interface or the Front Interface of your application has to be kept free from application logic related coding. The whole idea behind this approach is to replace the front layer (User Interface) without hurting other part of the system.

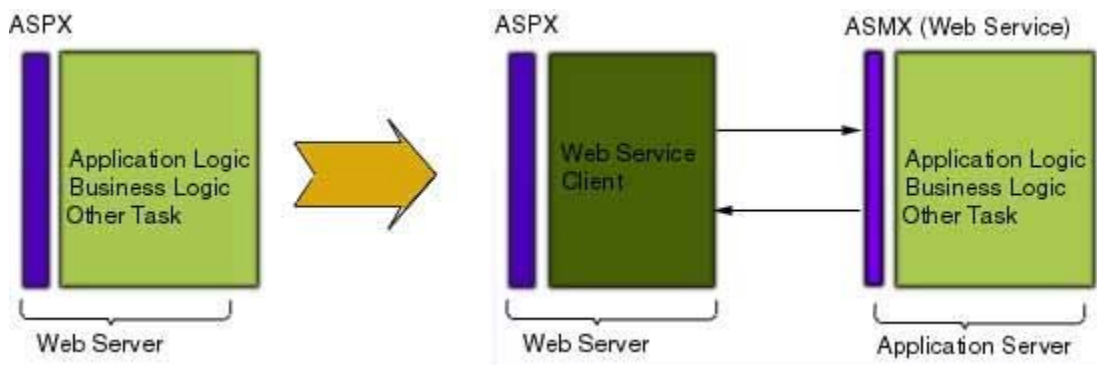


Figure-6: Explain how the front end can be easily replaced by different types of interfaces

E.g.:- As an example think of a web site you developed using .NET/ ASPX pages. In that case front interface of the application is the set of ASPX pages and their code behind files. If you have followed the concept above as you were implementing the system probably you should have set of ASPX pages that are free from application logic and also another set of modules which contains the core application logic. This simple breaking allows you to introduce a web service interface (ASMX file) to distribute the system across two machines to expand the system. This can be done just by replacing the ASPX files with a web service interface. This will separate the heart of the application to a separate application server and ASPX files to a web server as the above diagram explains.

## 5.7. Visualize the System

This allows designers to visually model the system to capture the structure and behavior of architectures and components. Visual abstractions help you understand the bigger view, while opening hidden areas of the system. This includes identifying component of the system correctly; understanding how each component of the system fits together; understanding how each component communicates with each other; and finally make each component design consistently. In my practice I draw an activity diagram and a System Architecture Diagram (as drawn in Figure -4) if the system is very complex but only the second, when it is not so. I encourage you to visually display the system before starting the design.

## 5.8. Naming Convention

1. Use lengthy meaningful/ readable names when naming variables, methods, classes, modules and any other. This will also excuse you for not commenting your code.
2. Follow the naming convention of the technology owner, if your application is developed using Microsoft .NET then follows the Microsoft standard and if it is any thing else then follow their naming convention, by doing so you will have the luxury of directly using their sample code in your application with out going through that naming convention adjustment process.

## 6. Think Fresh and Approach like a Newcomer

Today the technology changes at a rapid rate, allowing new things to evolve every day. The designers are expected to keep their knowledge up to date with the most recent technologies, so that they can utilize them early in their designs. But the superiority of the new technology may also lead the overwhelmed designers to overuse the technology. I have heard some designers say that "we designed our system according to the X model, it is the latest, and that does not recommend doing it", my simple advice is not to make bottlenecks in your system, just because you have to follow the latest technology or because everyone else was doing it that way. There are no magical formula that suite for everything, so if the technique (or the model) does not suite, be brave to change. Think what is needed? Identify, what suits you most? Then take your decisions, while letting everything else stand aside. Concepts are there to help you and better guide you, but not to control you. I invite you to be creative, but also note you, not to reinvent the wheel.

## 7. Additional Hints for System Designers

- Keep Interfaces simple. An interface should capture the minimum essentials of an abstraction. Don't generalize; generalizations are generally wrong. Again, the interface must not promise more than the implementer knows how to deliver.
- Make it fast, rather than general or powerful. It is much better to have basic operations executed quickly than more powerful ones that are slower (of course, a fast, powerful operation is best, if you know how to get it).
- Don't hide power. When a low level of abstraction allows something to be done quickly, higher levels should not bury this power inside something more general.
- Do a prototype. If there is anything new about the function of a system, the first implementation will have to be redone completely to achieve a satisfactory (that is, acceptably small, fast, and maintainable) result. It costs a lot less if you plan to have a prototype. Unfortunately, sometimes two prototypes are needed, especially if there is a lot of innovation, but go for it.
- Divide and conquer. This is a well known method for solving a hard problem: reduce it to several easier ones.
- Handle separately. Handle normal and worst cases separately as a rule, because the requirements for the two are quite different. The normal case must be fast. The worst case must consider all cases.
- Memory is cheap. Therefore cached the answers to expensive computations, rather than doing them over.
- Compute in background when possible. In an interactive or real-time system, it is good to do as little work as possible before responding to a request. The reason is twofold: firstly, a rapid response is better for the

- users, and secondly, the load usually varies a great deal, so there is likely to be idle processor time later in which to do background work.
- Make actions atomic or restart-able. An atomic action (often called a transaction) is one that either completes or has no effect.
  - Allow Customer to Lead. If the requirements are not finalized, keep the design as open as possible. You may strategically drive the customer to lead the requirement gathering process.

## 8. Final Note

There probably isn't a 'best' way to build a computer system; much more important is to avoid choosing a terrible way. The software designing methodologies are still evolving and can be considered as fairly new. The software system is an automation of a known manual process, indeed software cannot be defined for things that are not seen/ heard in the physical world. Any process gets refined as it is being reused. The processes of the physical world (manual processes) have evolved/ reused for many generations and have tuned to perfection. The system designers can take advantage from available manual systems, when designing a software system to automate such process. They can first study the manual system and automate the manual system with a software system.

So the safest path, the system designer can take is to design/ implement the software system as closely as its parallel manual system of the physical world.

## 9. Coincident

HIPO - Hierarchy plus Input-Process-Output, is a technique for use in the top-down design of systems, and was originally found by IBM in 1970s. The second step of the above proposed design technique is some what equal to HIPO technique. But HIPO had serious flaws that caused it to fall out of favor. But now there is another merging technique named HIPO-II, which competes with the most advanced design methods while maintaining its original simplicity.

The technique I have presented here is one of my own and is not some thing I've learnt or heard before, but found that it is extremely practical and helpful to deal with current day system designing requirements. In summary it is a consistent, logical and also a teachable technique. Amazingly some part of this technique is mapped with the IBM HIPO technique. I think it is yet another time, where it proves, that every thing is going on a cyclical path.

## 10. Summary

The approach presented here for system designing, can be broken in to four main steps as listed below.

- Identify, study and be a master of the problem domain
- Analyze the system by asking questions
  - Identify the list of granular level use cases or the functions of the system and internal/ external actors.
  - Form classes by grouping similar functions together.
  - Form modules by grouping similar classes together.
  - Recognizes module's communication paths

- Visualize the system with a diagram, showing the module interactions
- Identify the relations of the classes and draw the class diagrams of the system separately for each module
  - Place "module managers" correctly in between modules to bind them together to form the complete class diagram of the system.

I have ideas and contents to fill a book on this same concept. But I'd rather make it the minimum having concerns about the article download time. Here I am, concluding the article, hoping that it was written enough for you to understand the concept.

## 11. History

- 19-11-2006:

- Added Summary section.
- Updated/ improved overall wording of the article.
- Added figure 1.
- Updated figure 2 and 3.
- Updated the sample Questions and Answers list.

- 07-01-2007

- Table of Contents Added.
- Updated paragraphs under OOP (section 5.1) and design patterns (section 5.2)

## 12. References

1. [Hints for Computer System Design - Butler W. Lampson](#)
2. [A State of the Art Report: Software Design Methods - Robert L. Vienneau and Roy Senn](#)
3. [HIPO and Integrated Program Design - J. F. Stay](#)