

Enterprise Application Architecture

The field of enterprise architecture essentially started in 1987, with the publication in the IBM Systems Journal of an article titled "A Framework for Information Systems Architecture," by J.A. Zachman. In that paper, Zachman laid out both the challenge and the vision of enterprise architectures that would guide the field for the next 20 years. The challenge was to manage the complexity of increasingly distributed systems. As Zachman said: "*The cost involved and the success of the business depending increasingly on its information systems require a disciplined approach to the management of those systems*". Zachman's vision was that business value and agility could best be realized by a holistic approach to systems architecture that explicitly looked at every important issue from every important perspective. His multiperspective approach to architecting systems is what Zachman originally described as an information systems architectural framework and soon renamed to be an enterprise-architecture framework.

The evolution of enterprise architecture happened because at that time system development had to address two core problems:

- System complexity—Organizations were spending more and more money building IT systems
- Poor business alignment—Organizations were finding it more and more difficult to keep those increasingly expensive IT systems aligned with business need.

Enterprise: An organizational unit – from a department to a whole corporation.

Architecture: A formal description of a system, or a detailed plan of the system at component level to guide its implementation. The structure of components, their inter-relationships, and the principles and guidelines governing their design and evolution over time.

Enterprise Architecture: It is a formal description of an enterprise, a detailed map of the enterprise at component level to guide its changes. It provides details regarding the structure of an enterprise's components, their inter-relationships, and the principles and guidelines governing their design and evolution over time. Enterprise Architecture is about understanding all of the different components that go to make up the enterprise and how those components inter-relate.

Enterprise application (EA) (popularly known as enterprise software or enterprise application software (EAS)) is applications that a business uses to support the organization in order to solve enterprise problems. Normally, EA denotes a software platform that is quite complex and comparatively huge for small business use. However, they are designed to be user-friendly and easy to use. They are database-centric and demand high requirements in terms of security, user-access and maintenance.

It is a large software system platform intended to work in a corporate atmosphere like business or government. They are component-based, complex, scalable, distributed and mission critical. EA software comprises of a collection of programs with common business applications and organizational modeling. EAs are developed with enterprise architecture. The design of EA is accomplished in such a way that it can be successfully deployed across a

range of corporate networks (or intranets /Internet). It should assure to adhere with strict security requirements and administration management. Contrasting from the small business applications, which target particular business (e.g. billing software), an EA defines the business logic and supports the entire organization (including its departments), so as to lower costs and increase productivity as well as efficiency in the enterprise.

"At its heart, enterprise computing is all about combining separate applications, services and processes into a unified system that is greater than the sum of its parts."

- Jim Farley et al. in *Java Enterprise in a Nutshell*

An Enterprise Application is a combination of other applications, services and processes that, when unified, form a application that is greater than the sum of its parts."

- Paul Grenyer

Java is widely used for developing EA .

It is because it is named after(or illustrated by) coffee !!(Are you sure??)

Java has a vast array of excellent libraries for solving most of the common problems one needs to solve when developing enterprise applications. In many cases, there is more than one good choice for addressing a particular need, and oftentimes those libraries are free and open source under a business-friendly license.

Many companies choose Java for developing EA because:

Learning Curve

Java is a simple language with a huge support base. Enterprise projects tend to involve large numbers of developers and it is much easier to get a developer to a minimum level of competence in Java than other languages like C++. A lot of university graduates have been largely schooled in Java.

Choice & Reuse

Java has a vast array of libraries, frameworks, tools and IDEs, and server providers. To an enterprise it's good to have choice, even if that's just for use as a bargaining tool when negotiating price. The language lends itself to code quality tools that let implementation of corporate standards (and as mentioned there are a lot of those tools).

Platform Independence

Java is writing once, run everywhere. Sun has actively encouraged open standards that allow multiple vendors to implement their solutions. These standards give the customer the comfort that they can migrate from one vendor to another if a given vendor goes under or starts charging more. Of course the reality is that each vendor does their best to provide some "added value" features that tie the customer to them quite nicely.

Maturity

It has been around a long time, running a lot of servers. If your web application needs to be "6 sigma" or similar and you are the Mega Corp Chief Tech Officer, you are not going to look for developer wanting to do it in relatively immature platforms like RoR.

Timing/Marketing

Java was introduced to the world when programming was moving towards the web. It was positioned smartly and got a solid position early in web development. Because of the open standards, there are some very big companies producing these platforms and they market Java pretty hard to sell those platforms.

Inertia

Large corporations update/upgrade themselves very slow (many are still using Java 1.4), so once they have picked Java, it takes a huge investment to migrate to another platform. And convincing for change is to assure them to invest a lot of money for no immediate business benefit.

To recapitulate, Java ensures security, stability, robustness, scalability, platform independence, performance & resources usage, and Academic society wide acceptance.

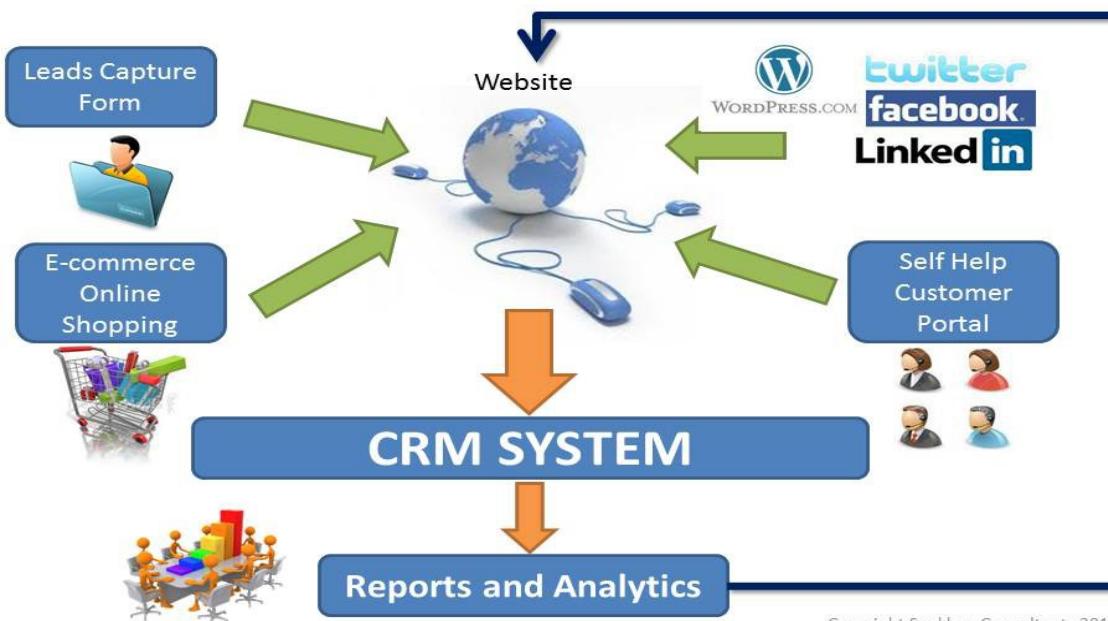
Common Types of Enterprise Applications

Some of the more common enterprise applications include the following:

- Customer relationship management (CRM)

Wikipedia says - CRM is a model for managing a company's interactions with current and future customers. It involves using technology to organize, automate, and synchronize sales, marketing, customer service and technical support. It involves all kinds of interaction that a business corporation has with its client, whether it is sales or service-related. CRM enables the business by: Understand the customer, Retain customers through better customer experience, Attract new customer, Win new clients and contracts, and Increase profitably, Decrease customer management costs.

How CRM fits in with your Online Strategy



Copyright Sankhya Consultants 2013

Fig: CRM<<http://crmconsulting.net.au>>

- Automated billing systems
- Enterprise Resource Planning
- HR Management

Enterprise Application Architecture (EAA)

Enterprise Architecture describes how organizational, information and technology structures support the strategy and operations of organizations. The EAA architecture of a business is a explanation of the structure of the application & software usage through the organization. The systems are decomposed into subsystems and connected/related to each other based on their interaction. The relationships with external environment, guidelines, users, terminology are also specified as a part of enterprise architecture of Software Systems. During the designing of EAA, an effective and thorough survey of existing systems should be performed by a team of high technical analysts in order to understand these systems and their interactions with other applications and databases.

Advantage/Need/Uses of enterprise application architecture

The purpose of enterprise architecture is to optimise across the enterprise the often fragmented legacy of processes (both manual and automated) into an integrated environment that is responsive to change and supportive of the delivery of the business strategy. Thus the primary reason for developing an EA is to get an overview (map) of the business' processes, systems, technology, structures and capabilities. We need an EA to provide a strategic context for the evolution of the IT system in response to the constantly changing needs of the business environment. We need an EA to achieve competitive advantage.

It acts for bridging the gap between Business and IT. It helps to enhance the relationships between IT and the business. It reinforces IT understanding of the business strategy. It creates a process for continuous IT/business alignment. Furthermore, it enhances IT agility to support business changes. And Hence, create business value from IT.

Describing and characterizing the architecture of an enterprise application is beneficial as it acts like a central document. This central document specifies an unambiguous understanding of the complete system and how it works as a whole. This aids in finding problems and solving them in order to speed up existing systems and enhance them in terms of performance. At time of introduction of a new application software or improvement of existing applications, the application architecture acts a key document for studying the effects of the changes in the entire enterprise. This document should be referred as a guide during the improvement/replacement of the current systems. Whenever an enterprise application is introduced into an existing system, the application architecture acts as the foundation for developing this software.

The value of EA:

- Business - IT and business – business alignment
- Change enabler
- Improved agility to enable a real time enterprise
- Standardisation, reuse and common principles, terms and work practices
- Integration and interoperability
- Structure, multiple perspectives and documentation

EA Bridges Strategy and Implementation



Business Strategy

- Business drivers
- Business goals
- Business policy
- Trend analysis



Implementation

- Business processes
- Application systems
- Tech infrastructure
- Organizational structure

The bridge between strategy & implementation

Lecture 15 – Enterprise Architecture, TOGAF, Gartner,
FEA
www.ntnu.edu

TDT4252, Spring 2012



NTNU – Trondheim
Norwegian University of
Science and Technology

Enterprise Application vs. Desktop Application vs. Web Application

Desktop application runs on a single tier architecture. All the presentation logic, business logic and data storage and access logic resides in a single machine. Moreover, it can be a part of network. But the application and its database stay in the same machine. It does not support multiple users concurrently but only one user can access it at a time.

For beginners, web application and enterprise application seem to be similar although they are not. Enterprise application normally includes more than a single tier i.e. a multi-tier application. Multiple concurrent users can access the application.

For example Enterprise Application has at different tiers like Client, Presentation, Business Logic and Data tier. Client Tier can be Java clients, browser-based clients and mobile clients. Presentation Tier can be servlets, JavaBeans components and JSP components. Business Logic Tier can include servers, web services (SOAP, RESTful) ,etc. Data Tier can be of DBMS and LDAP.

Web application can be implemented without such complex enterprise. For example we can use various technologies to implement a web application such as Servlets, JSP, Hibernate, Maven, Databases, JavaBeans, etc. However, web applications suffer shortcomings when special kind of infrastructure services (functional or non-functional) are required as an essential part of the application. There are various applications with such special service requirements which require a business layer running over an infrastructure offering special services like: timer services, distributed transactions, remote method invocation, messaging

processing, etc. This kind of infrastructure services is not available in web servers. Hence, need arises for enterprise application that runs on a heavyweight application server (e.g. IBM WebSphere, Jboss and Geronimo).

Furthermore, Oracle website has put some clarifications regarding the standard edition of Java and Java Enterprise Edition. The major difference of Java EE from the standard edition is that Java EE requires an application server, instead of a web server like tomcat. The reason behind the need of application server is that, EJBs where the business logic resides needs a container. EJBs can be managed by application servers like Geronimo, IBM WebSphere and Jboss. These special application/containers can also handle web applications. Furthermore, an enterprise application holds enterprise beans (*an enterprise bean is a server-side component that encapsulates the business logic of an application*) and executes in a J2EE Container which caters security and transaction services to the beans. The associated file is an .ear file. Whereas a web application runs in a web-container like tomcat, contains servlets/JSPs. The associated file is named with an extension of .war.

Enterprise Architecture Frameworks(EAF)

Frameworks help people organize and assess completeness of integrated models of their enterprises. An Architectural Framework gives a skeletal structure that defines suggested architectural artifacts, describes how those artifacts are related to each other, and provides generic definitions for what those artifacts might look like. EAF says:

- How to create and use an enterprise architecture.
- Principles and practices for creating and using architectural description of the system.

Purpose of Framework

- Organize integrated models of an enterprise
- Assess completeness of the descriptive representation of an enterprise
- Understand an organization or a system
- Assist in identification and categorization
- Provide a communication mechanism
- Help manage complexity
- Identify the flow of money in the enterprise

Zachman Enterprise Architecture Framework

The **Zachman Framework** is an EAF which provides a formal and highly structured way of viewing and defining an enterprise. It consists of a two dimensional classification matrix based on the intersection of six communication questions (What, Where, When, Why, Who and How) with five levels of reification , successively transforming the most abstract ideas (on the Scope level) into more concrete ideas (at the Operations level). The Zachman Framework is a schema for organizing architectural artifacts (in other words, design documents, specifications, and models) that takes into account both whom the artifact targets (for example, business owner and builder) and what particular issue (for example, data and functionality) is being addressed. The Zachman Framework is not a methodology in that it does not imply any specific method or process for collecting, managing, or using the

information that it describes

	1. What (data)	2. How (function)	3. Where (network)	4. Who (people)	5. When (time)	6. Why (motivation)
1. Scope (context)						
2. Business model (concept)						
3. System model (logical)						
4. Technology model (physical)						
5. Detailed representation (component)						
6. Real system, i. e. executing the game of baseball						

Fig: An Empty Zachman Framework

The rows present:

- different perspectives of the enterprise
- different views of the enterprise
- different roles in the enterprise

1. Scope describes the system's vision, mission, boundaries, architecture and constraints. The scope states what the system is to do. It is called a black box model, because we see the inputs and outputs, but not the inner workings.
2. Business model shows goals, strategies and processes that are used to support the mission of the organization.
3. System model contains system requirements, objects, activities and functions that implement the business model. The system model states how the system is to perform its functions. It is called a white box model, because we see its inner workings.
4. Technology model considers the constraints of humans, tools, technology and materials.
5. Detailed representation presents individual, independent components that can be allocated to contractors for implementation.
6. Real system depicts the operational system under consideration.

Columns present the various aspects of the enterprise.

1. What (data) describes the entities involved in each perspective of the enterprise. Examples include equipment, business objects and system data.
2. How (functions) shows the functions within each perspective.

3. Where (networks) shows locations and interconnections within the enterprise. This includes major business geographical locations, networks and the playing field.
4. Who (people) represents the people within the enterprise and metrics for assessing their capabilities and performance. The design of the enterprise organization has to do with the allocation of work and the structure of authority and responsibility.
5. When (time) represents time, or the event relationships that establish performance criteria. This is useful for designing schedules, the processing architecture, the control architecture and timing systems.
6. Why (motivation) describes the motivations of the enterprise. This reveals the enterprise goals, objectives, business plan, knowledge architecture, and reasons for thinking, doing things and making decisions.

Government Enterprise Architecture Frameworks (GEAFs)

In the context of government enterprises: a coordinated set of activity areas involving one or more public organizations and possibly third-party entities from private organizations or civil society, an EA provides technical descriptions of the organizational goals, business and administrative processes, information requirements, supporting applications and technology infrastructure of the enterprise. These descriptions are typically captured in the form of models, diagrams, narratives, etc. A Government Enterprise Architecture (GEA) may be associated with a single agency or span functional areas transcending several organizational boundaries, e.g. health care, financial management and social welfare.

Reasons for developing enterprise architectures in government include:

1. Understanding, clarifying and optimizing the inter-dependencies and relationships among business operations, the underlying IT infrastructure and applications that support these operations in government agencies and in the context of specific government enterprises
2. Establishing a basis for agencies to share information, knowledge and technology and other resources or jointly participate in the execution of business processes
3. Optimizing ICT investment and business cases across the whole of government by enabling the opportunities for collaboration and sharing of assets, thus reducing the tendency for duplicated and poorly integrated IT resources and capabilities

There is increasing awareness on the importance of EA as most of the leading countries in e-government have well established EA programs. Presently, there are EA maturity models with defined relations to well known e-Government Maturity stages. The increasing popularity of EA practices by governments in both developed and developing countries is indicated by the different global surveys on EA.

Despite the popularity of the EA practice in the private sector and increasingly in the government, the EA discipline is relatively new, lacking foundational theories and models and characterized by multiplicity of frameworks and reference models; even lacking an agreement on the definition and scope of the subject matter. From the earlier orientation of EA as a technological optimization or standardization concern, EA has gradually evolved to a management practice with stronger emphasis placed on the organizational-IT alignment.

However, there are mixed results in terms of outcomes from EA initiatives. In general, demonstrating concrete benefits from EA program has been challenging for many organizations. This difficulty is attributed to lack of metrics for EA initiatives. Notwithstanding,

a number of successful EA initiatives have been reported by some governments, particularly, in Canada and the US. Unfortunately, comparing and analyzing these EA initiatives and cases is difficult in the absence of assessment frameworks, techniques and tools.

1. Aim

The project aims to provide policy guidelines for the development of Government Enterprise Architecture Frameworks (GEAFs), establish concrete requirements for such a framework in Macao, and provide recommendations on how elements of a Macao GEAF (MGEAF) could be built from existing Government EA Frameworks, Reference Models, Methods, and Modeling Framework. The project will also provide an example of agency-specific EA based on the recommended Macao framework.

2. Objectives

1. Improving understanding and contributing to the body of knowledge of GEA through foundational research
2. Enhancing EA practice by providing policy guidelines and development of Government EA Frameworks based on results from (1) with the supporting toolkit
3. Understanding the factors that contribute to wide adoption of EA practice within a government
4. Building capacities of government agencies and their architects through development of courseware for educational and training purposes as well as the use of tools in (2)
5. Dissemination of project output (1 through 4) through various channels including publications (books, journals, conference papers and technical reports), schools and courses, seminars, workshops and projects.

References:

- <http://www.isummation.com/>
- <http://en.wikipedia.org>
- <http://stackoverflow.com>
- <http://www.oracle.com/>
- <http://msdn.microsoft.com/en-us/library/bb466232.aspx>
- <http://www.egov.iist.unu.edu/cegov/projects/Strategy-Government-Enterprise-Architecture Lecture Notes on Enterprise architecture , Norwegian University og Science and Technology>

Introduction

What is Enterprise?

- A business Organization
- In computer industry, term is often describe as the large organization that utilized computers
- E.g. Intranet example of an enterprise computing system
⇒Enterprise Application:
 - A business Application
 - Business application are complex, scalable ,distributed, component based and mission critical.

Trend(better tools, greater impact, powerful option)

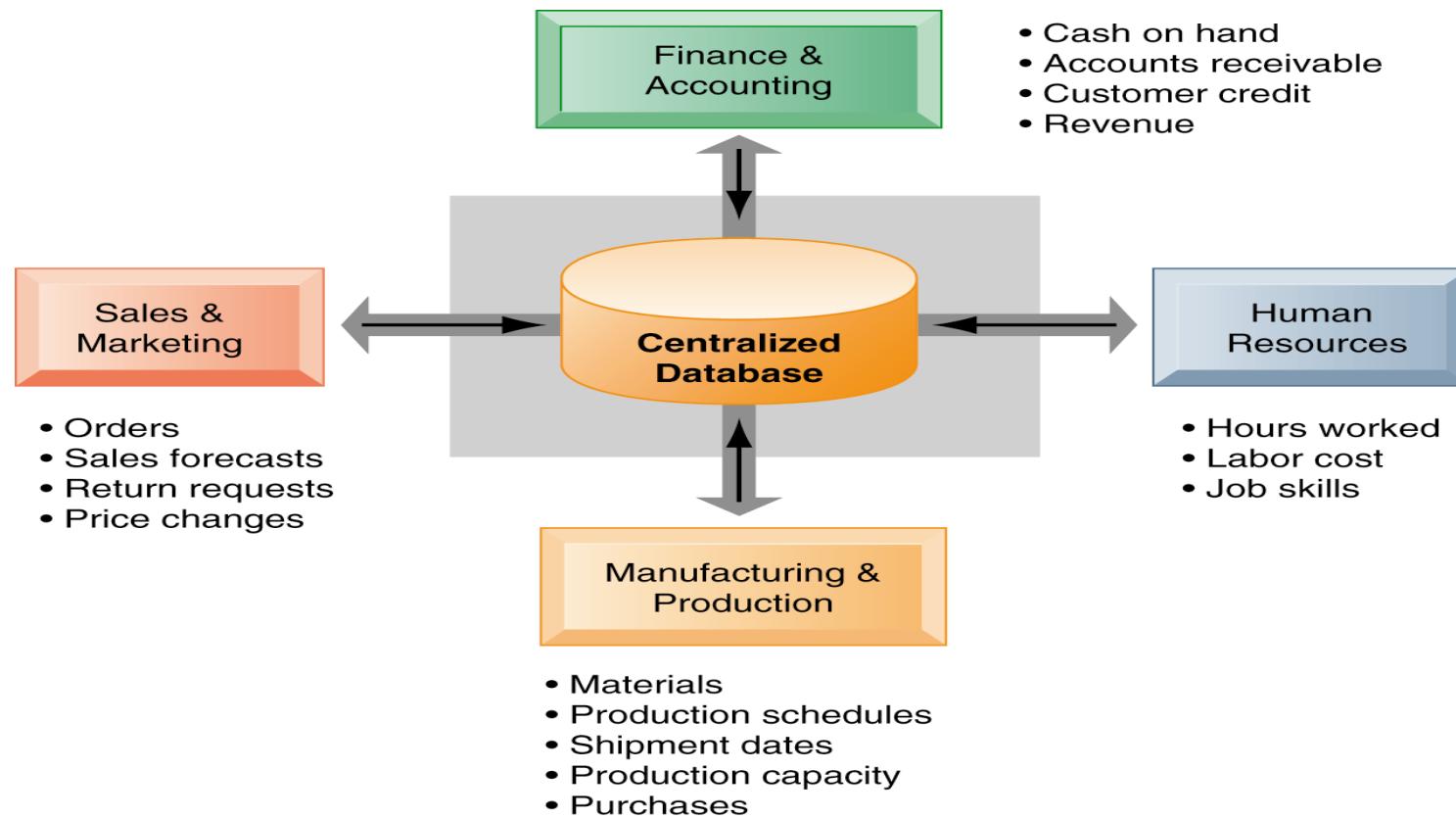
- Growth of User Experience (UX)
- Explosion of Development Tools
- Decline of Native Development
- Growing business focus on user experience
- Growing need for integration and web service
- Others.....????

Enterprise Application Integration

Enterprise application integration (EAI) is the use of technologies and services across an enterprise to enable the integration of software applications and hardware systems. Many proprietary and open projects provide EAI solution support. EAI is related to middleware technologies. Other developing EAI technologies involve Web service integration, service-oriented architecture, content integration and business processes. Intercommunication between enterprise applications (EA), such as customer relations management (**CRM**), supply chain management (**SCM**) and business intelligence is not automated. Thus, EAs do not share common data or business rules. EAI links EA applications to simplify and automate business processes without applying excessive application or data structure changes. *However, EAI is challenged by different operating systems, database architectures and/or computer languages, as well as other situations where legacy systems are no longer supported by the original manufacturers.*

Keyword Web services and business integration

How enterprise System Works?



Enterprise systems feature a set of integrated software modules and a central database that enables data to be shared by many different business processes and functional areas throughout the enterprise.

What is EAI?

- an application coupling mechanism, to integrate individual applications into enterprise
- provides a unified interface to legacy systems
- defined as the uses of software and computer systems architectural principles to integrate a set of enterprise computer applications
- The real beauty of Enterprise Application Integration is making various separate applications work together to produce a unified set of functionality

Why EAI?

- System development over the last 20 years has tended to emphasize core functionality as opposed to integration
- Many systems are difficult to integrate with other similar systems
- Ultimately, it comes down to a cost issue. Building a system with integration in mind reduces the amount of money spent on further system development
- 30% of the IT budget is spent on linking/merging of databases and sources

Why EAI?

- It would be great if everyone used the same servers with the same operating system with the same clients, etc
- **Reality is very diverse.** We can expect a mix of mainframes, Windows, UNIX, Linux, VMS, as well as many other systems
- Getting them to work/share data together is the issue!
- The market is stocked with legacy systems designed to support single users without thought to integrating them into a larger whole.
- Many of these systems still provide value to the organization.
- Packaged applications often create their own set of issues and problems

Applying Technology

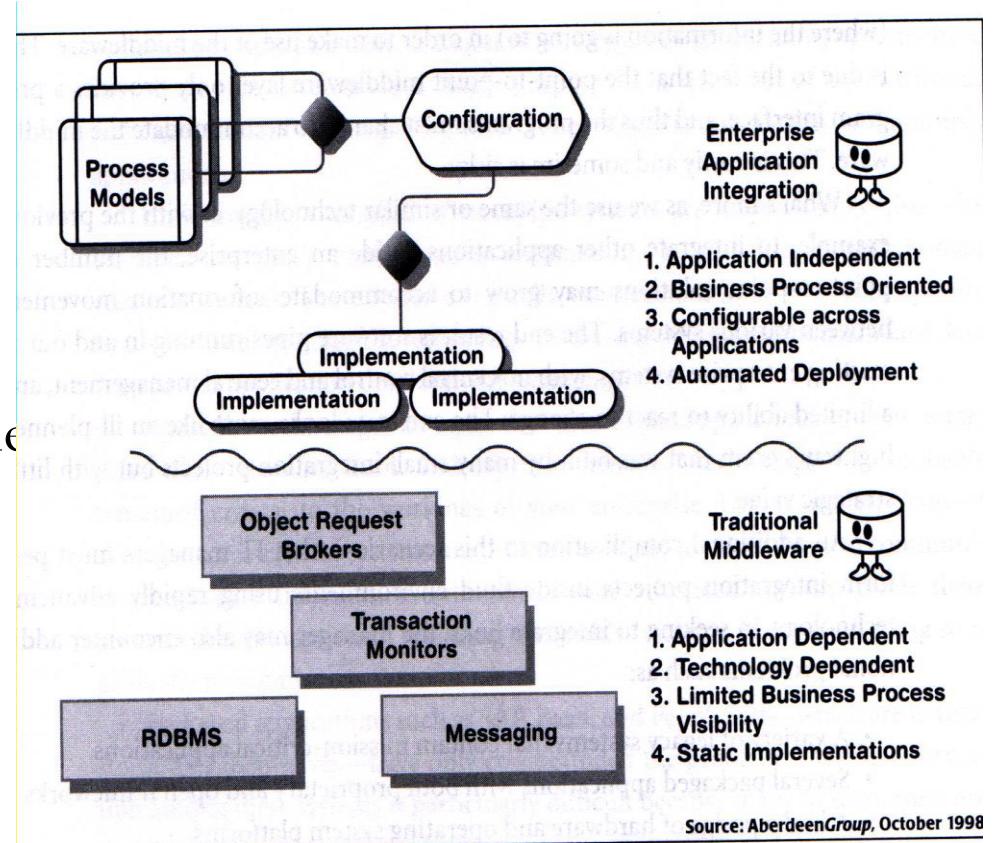
- The traditional solution to integration issues has been the introduction of ‘middle-ware’.
- Middle-ware acts as a transport mechanism to perform integration typically on client/server based systems.
- Many flavors/standards of middle-ware exist:
 - Remote Procedure Calls (RPC)
 - Sockets
 - Common Object Reuse Broker Architecture (CORBA)
 - Distributed Computing Environment (DCE)
 - Java’s Remote Method Invocation (RMI)

Applying Technology

- While middle-ware can be a solution, problems exist:
 - Changes are typically required to existing systems to incorporate middle-ware
 - No centralized management typically exists, so complex systems rapidly grow unmanageable
 - Technological advances tend to make middle-ware based systems look like an ill-planned highway system composed of small integration projects instead of a single over-reaching standard

Defining EAI

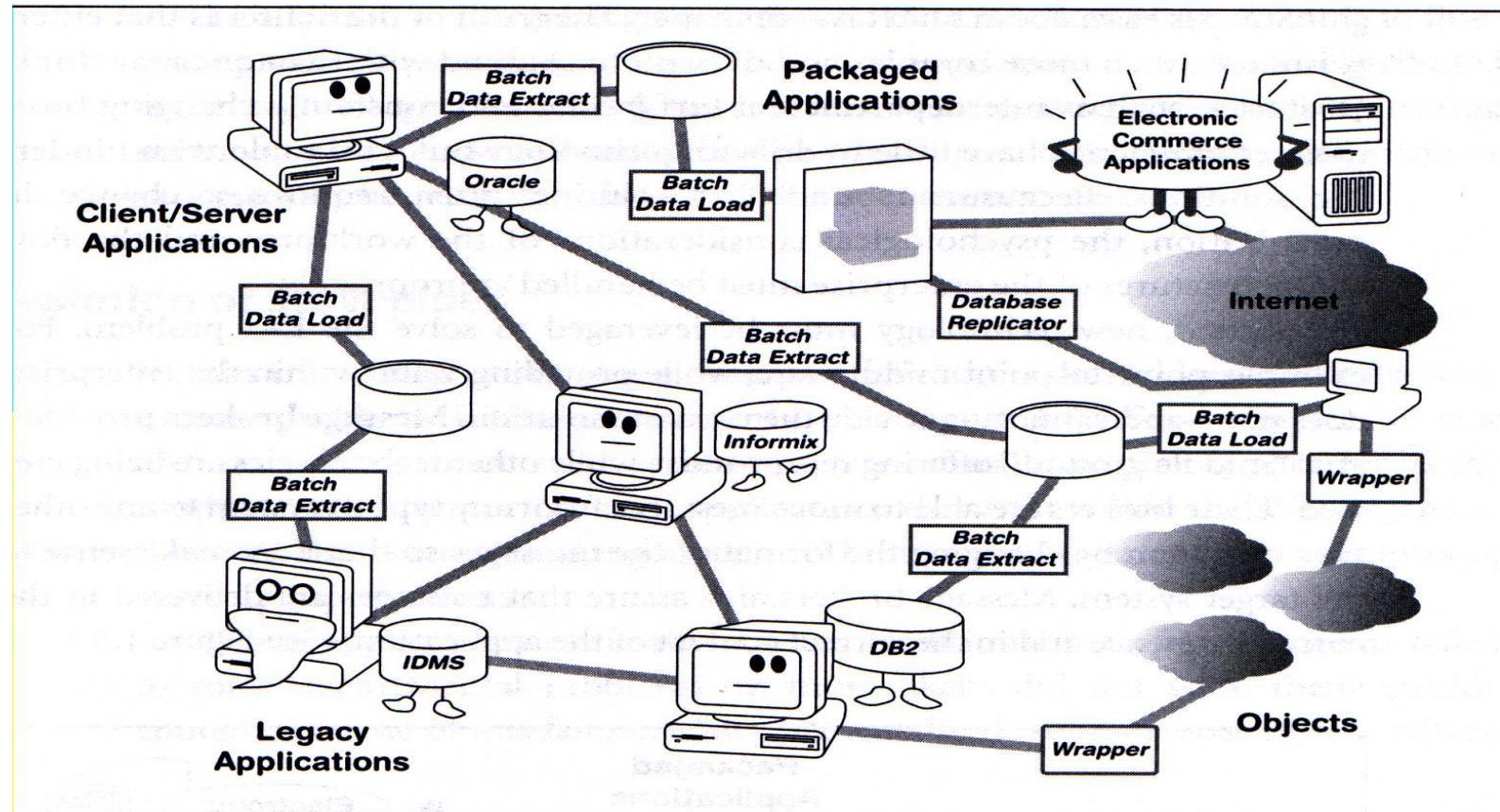
- Applying Technology
 - EAI focuses on the integration of both processes AND data while middle-ware is data oriented



Problem in EAI

- Most organizations lacked architectural foresight.
- As they upgraded from legacy systems, they moved to newer ‘open’ systems without consideration to how well these new systems would fit into their current structure and integrate with existing legacy systems.

Defining EAI

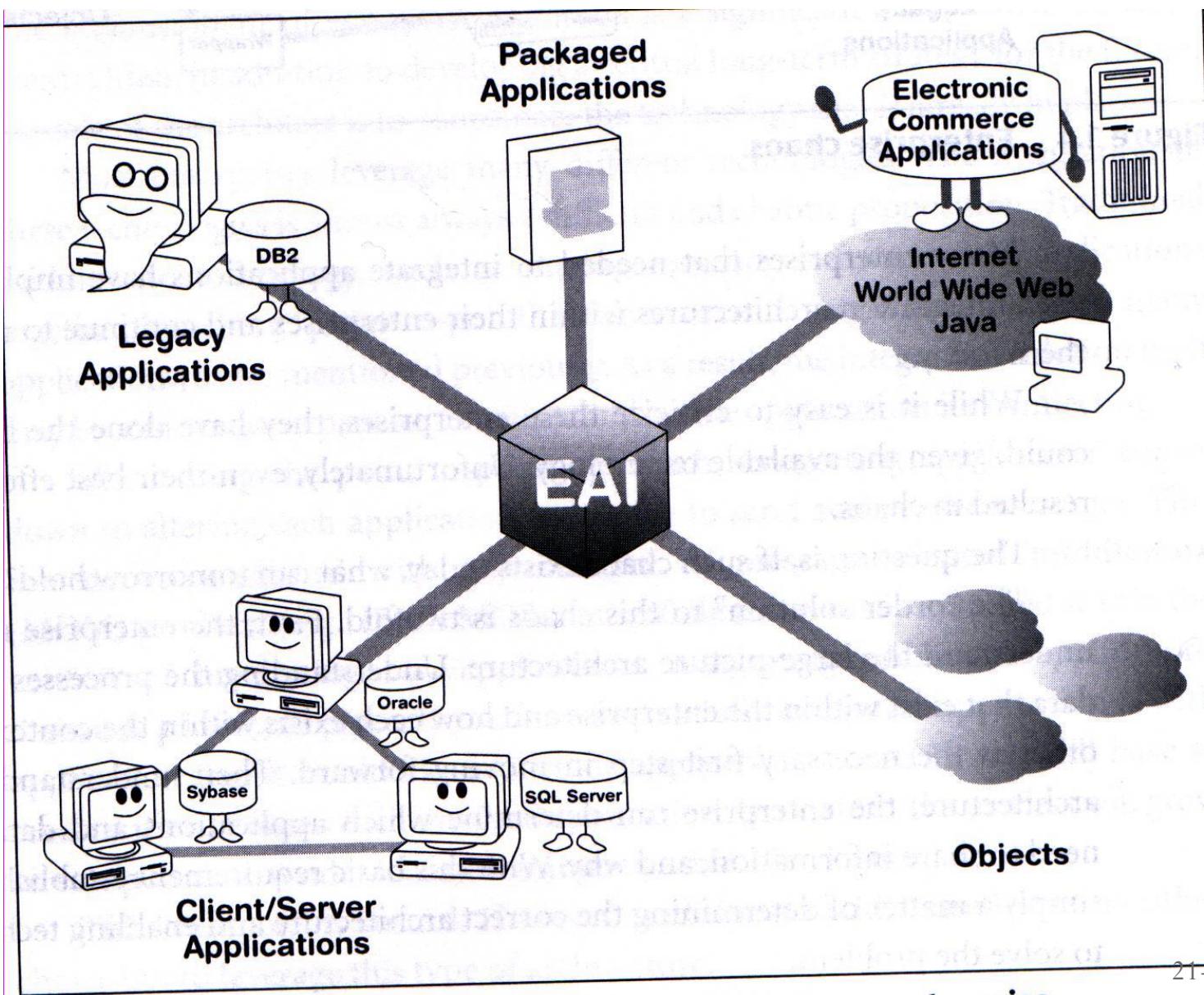


Enterprise Chaos!

How is EAI different?

- EAI focuses on the integration of both **business-level processes and data** whereas the traditional middleware approach is **data oriented**.
- EAI includes the notion of **reuse** as well as distribution of business processes and data.
- EAI allows users who understand very little about the details of the applications to integrate them

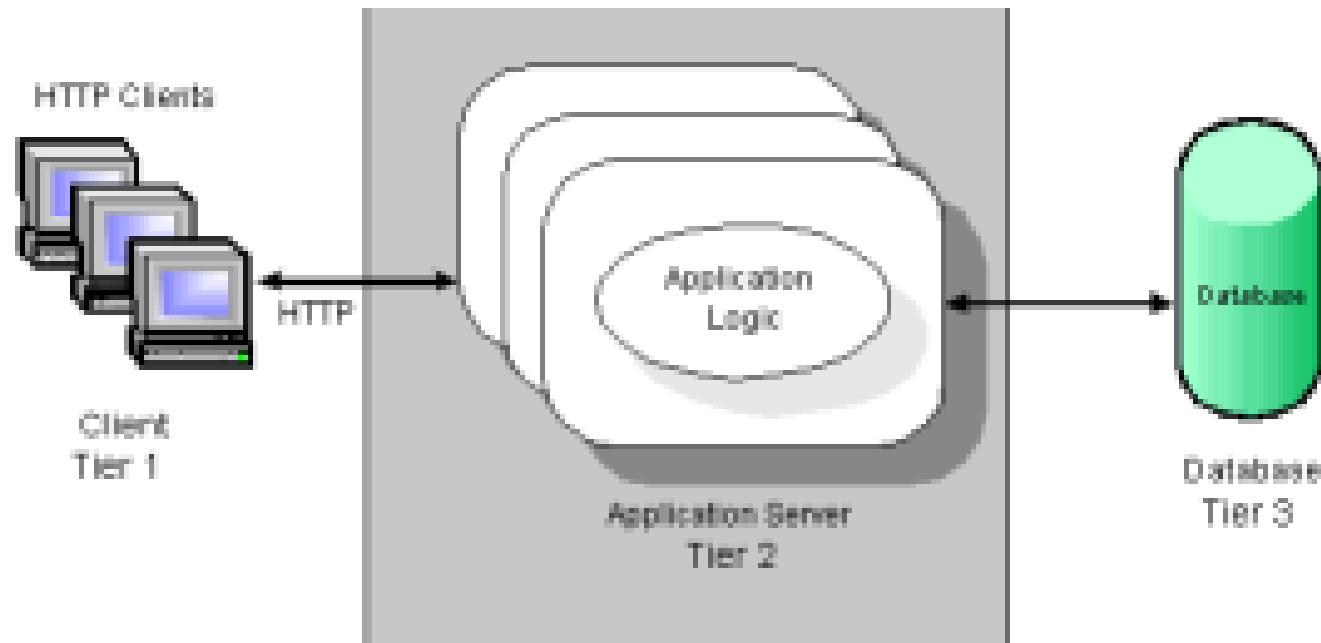
EAI Brings Order to the Enterprise



Purpose of EAI

- **Data (information) integration:** Ensuring that information in multiple systems is kept consistent. This is also known as EII (Enterprise Information Integration).
- **Process integration:** linking business processes across applications.
- **Vendor independence:** Extracting business policies or rules from applications and implementing them in the EAI system, so that even if one of the business applications is replaced with a different vendor's application, the business rules do not have to be re-implemented.
- **Common facade:** An EAI system could front-end a cluster of applications, providing a single consistent access interface to these applications and shielding users from having to learn to interact with different applications.

Multitier Architecture



Multitier architecture :

In software engineering, **multi-tier architecture** (often referred to as **n-tier architecture**) is a client–server architecture in which presentation, application processing, and data management functions are logically separated. For example, an application that uses middleware to service data requests between a user and a database employs multi-tier architecture. The most widespread use of multi-tier architecture is the **three-tier architecture**.

N-tier application architecture provides a model by which developers can create flexible and reusable applications. By segregating an application into tiers, developers acquire the option of modifying or adding a specific layer, instead of reworking the entire application. **Three-tier** architectures typically comprise a *presentation* tier, a *business or data access [logic]* tier, and a *data* tier.

(Wikipedia : Multitier Architecture)

3-tier architecture (application view)

Presentation tier

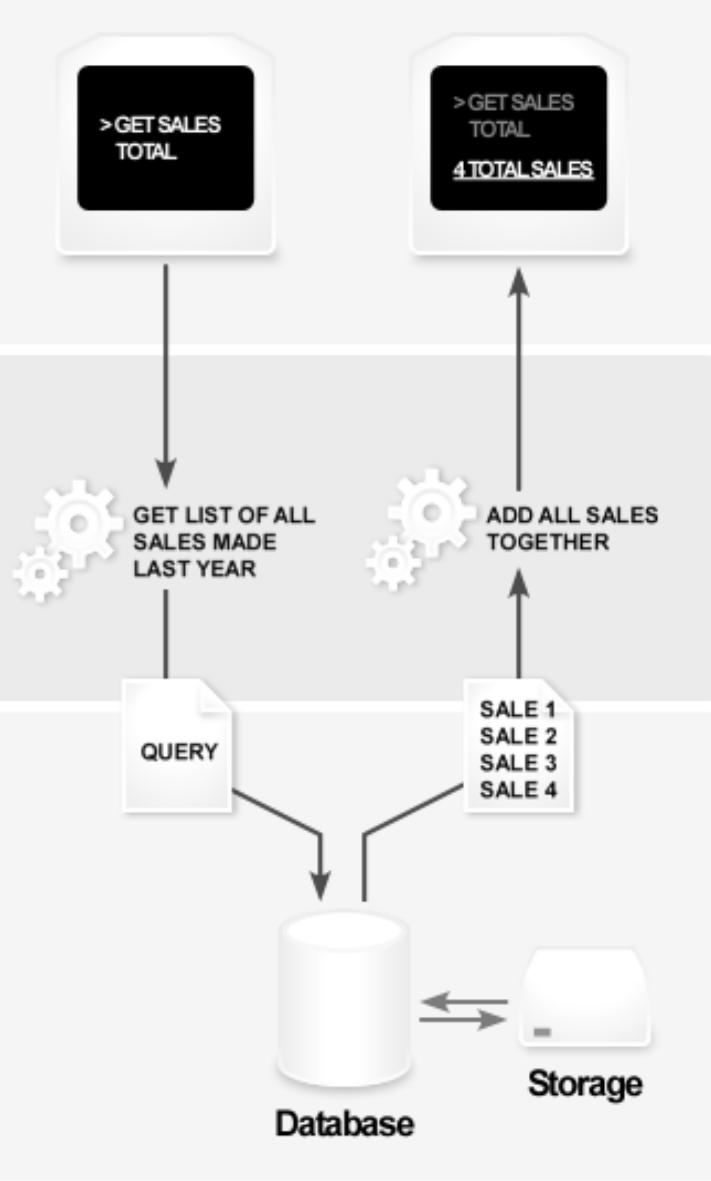
The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

Logic tier

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

Data tier

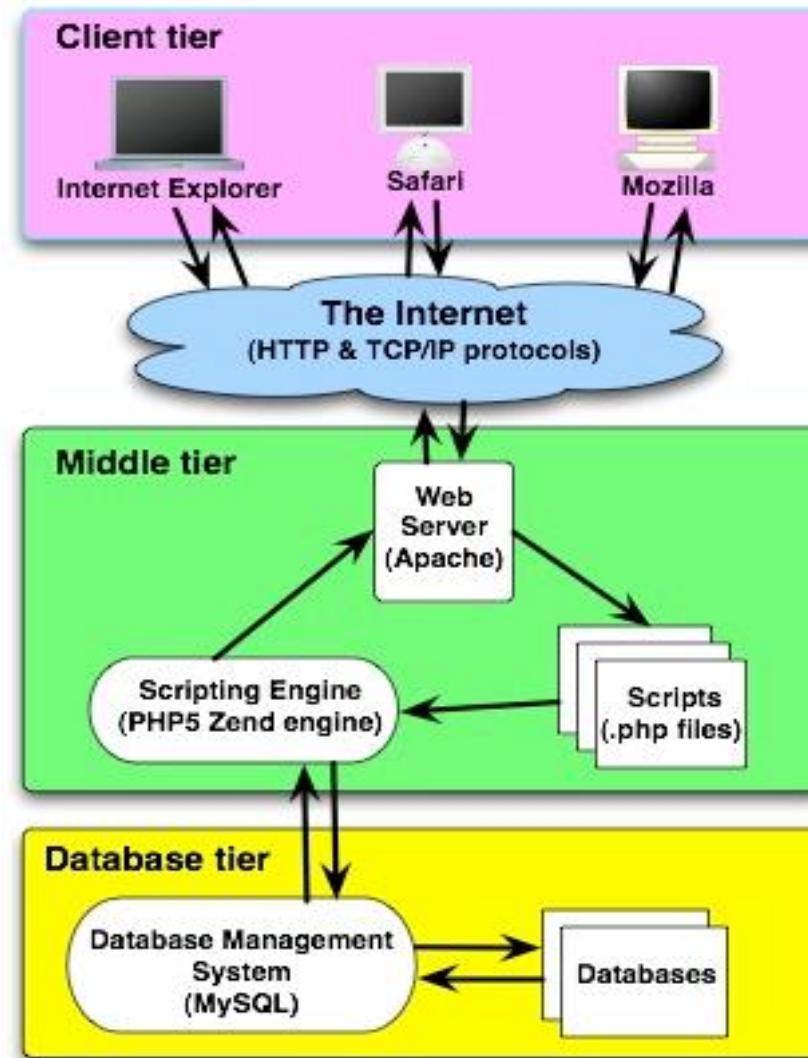
Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.



Advantages of the 3-tier architecture approach :

- the ability to **separate logical components** of an application ensures that applications are easy to manage and understand.
i.e. experts can be employed that specialise in one of the layers
e.g. user interface design
- because **communication can be controlled** between each logical tier of an application, changes in one tier, for example, the database access tier, do not have to affect the client component
i.e. a change from one DBMS to another would only require a change to the component in the data access layer with little or no effect on the business/logic (middle) or UI layer.
- **specific tools and technologies suited to each layer** can be deployed (and may evolve at a different pace) .

Typical web-oriented 3-tier architecture



Web-oriented 3-tier architecture: tools & technologies

- o **Presentation tier** – Browser / custom client, Client Side Scripting (JavaScript, ActionScript, VBScript etc.), Applets.
- o **Logical Tier** – Web Server (Apache, IIS, Websphere etc.); Scripting Languages (PHP, Perl etc.), Programming Languages (Java, C, C# etc), Application Frameworks (Ruby on Rails etc.)
- o **Data Tier** – Database Management System (DBMS) (Oracle, MySQL, SQL Server, DB2 etc.), XMLDB

N-Tier Architecture - Advantages

- Scalable:
 - this is due to its capability of multiple tier deployment and the tier decoupling it brought.
 - For example,
 - the data tier can be scaled up by database clustering without other tiers involving.
 - The web client side can be scaled up by load-balancer easily without affecting other tiers.
 - Windows server can be clustered easily for load balancing and failover.
 - In addition, business tier server can also be clustered to scale up the application, such as Weblogic cluster in J2EE.
- Better and finer security control to the whole system:
 - we can enforce the security differently for each tier if the security requirement is different for each tier.
 - For example,
 - business tier and data tier usually need higher security level than presentation tier does, then we can put these two high security tiers behind firewall for protection.
 - 1 or 2 tiers architecture cannot fully achieve this purpose because of a limited number of tiers.
 - Also, for N-Tier architecture, users cannot access business layer and data layer directly, all requests from users are routed by client presenter layer to business layer, then to data layer. Therefore, client presenter layer also serves as a proxy-like layer for business layer, and business layer serves as a proxy-like layer for data layer. These proxy-like layers provides further protection for their layers below.
- Better fault tolerance ability:
 - for example,
 - the databases in data layer can be clustered for failover or load balance purpose without affecting other layers.

- Independent tier upgrading and changing without affecting other tiers:
 - in object-oriented world, Interface-dependency implementation can decouples all layers very well so that each layer can change individually without affecting other layers too much.
 - Interface-dependency means a layer depends on another layer by interfaces only, not concrete classes.
 - Also, the dependency of a layer only on its directly-below layer also minimizes the side effect of a layer's change on the whole system.
 - For example,
 - if keep the interfaces unchanged, we can update or replace the implementation of any layer independently without affecting the whole system.
 - Due to the changing of business requirement and technology, changing the implementation of a layer to another totally different one does happen often.
- Friendly and efficient for development:
 - the decoupled layers
 - are logic software component groups mainly by functionality,
 - are very software development friendly and efficient.
 - Each layer
 - can be assigned individually to a team who specializes in the specific functional area; a specialized team can handle the relevant task better and more efficiently.

- Friendly for maintenance:
 - N-Tier architecture groups different things together mainly by functionality and then makes things clear, easily understandable and manageable.
- Friendly for new feature addition:
 - due to the logically grouped components and the decoupling brought by N-Tier architecture, new features can be added easily without affecting too much on the whole system.
- Better reusability:
 - due to the logically grouped components and the loose couplings among layers.
 - Loosely-coupled component groups are usually implemented in more general ways, so they can be reused by more other applications.

The Disadvantages of the N-Tier Deployment

- The performance of the whole application
 - may be slow if the hardware and network bandwidth aren't good enough because more networks, computers and processes are involved.
- More cost for hardware, network, maintenance and deployment
 - because more hardware and better network bandwidth are needed.

Tier and Layer

- Tier
 - the physical deployment computer.
 - Usually an individual running server is one tier.
 - Several servers may also be counted as multiple tier, such as server failover clustering.
- Layer
 - logic software component group mainly by functionality;
 - is used for software development purpose.
- Tier and Layer
 - Layer software implementation has many advantages and is a good way to achieve N-Tier architecture.
 - Layer and tier may or may not exactly match each other.
 - Each layer may run in an individual tier.
 - Multiple layers may also be able to run in one tier.
 - A layer may also be able to run in multiple tiers.
 - For example, in Diagram 2 below, the persistence layer in .NET can include two parts: persistence Lib and WCF data service, the persistence lib in the persistence layer always runs in the same process as business layer to adapt the business layer to the WCF data service. However, the WCF data service in persistence layer can run in a separate individual tier.
 - Here is another example: we may extract the data validation in business layer into a separate library (but still kept in business layer), which can be called by client presenter layer directly for a better client-side interactive performance. If this occurs, then data validation part of the business layer runs in the same process of the client presenter layer, the rest of business layer runs in a separate tier.



Model View Controller (MVC)

Smalltalk-80™

- In the MVC paradigm, the user input, the modeling of the external world, and the visual feedback to the user are explicitly separated and handled by three types of objects, each specialized for its task.
 - The **view** manages the graphical and/or textual output to the portion of the bitmapped display that is allocated to its application.
 - The **controller** interprets the mouse and keyboard inputs from the user, commanding the model and/or the view to change as appropriate.
 - The **model** manages the behavior and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller).

Smalltalk-80™ continued

- The formal separation of these three tasks is an important notion that is particularly suited to Smalltalk-80 where the basic behavior can be embodied in abstract objects: *View*, *Controller*, *Model* and *Object*.

Sun says

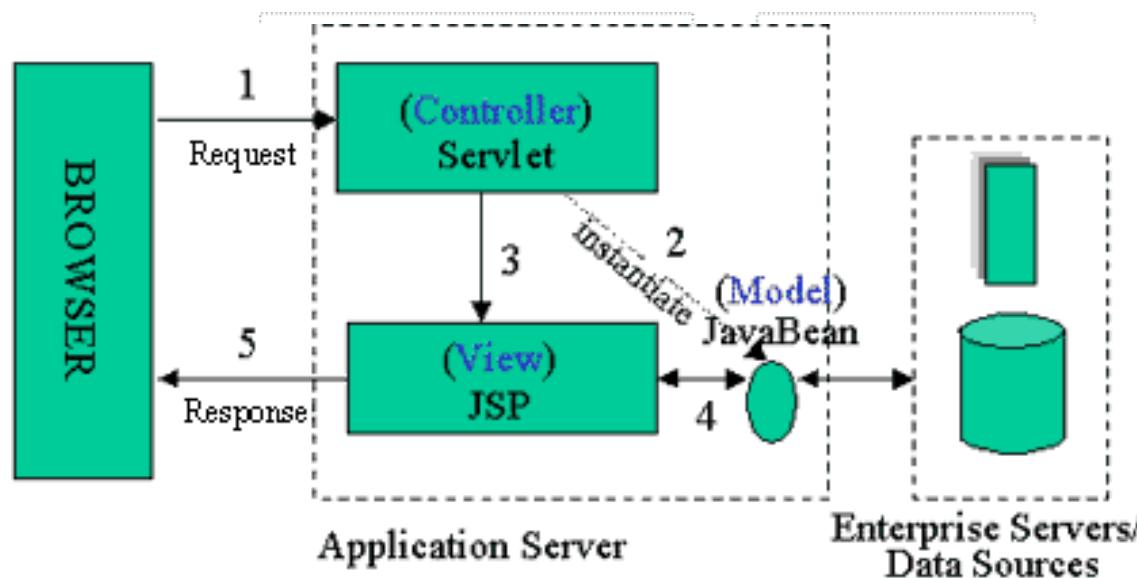
- Model-View-Controller ("MVC") is the recommended architectural design pattern for interactive applications
- MVC organizes an interactive application into three separate modules:
 - one for the application model with its data representation and business logic,
 - the second for views that provide data presentation and user input, and
 - the third for a controller to dispatch requests and control flow.

Sun continued

- Most Web-tier application frameworks use some variation of the MVC design pattern
- The MVC (architectural) design pattern provides a host of design benefits

Java Server Pages

- Model 2 Architecture to serve dynamic content
 - Model: Enterprise Beans with data in the DBMS
 - JavaBean: a class that encapsulates objects and can be displayed graphically
 - Controller: Servlets create beans, decide which JSP to return, do the bulk of the processing
 - View: The JSPs generated in the presentation layer (the browser)



OO-tips Says

- The MVC paradigm is a way of breaking an application, or even just a piece of an application's interface, into three parts: the model, the view, and the controller.
- MVC was originally developed to map the traditional input, processing, output roles into the GUI realm:
 - Input --> Processing --> Output
 - Controller --> Model --> View

Wikipedia says

- **Model-View-Controller (MVC)** is a software architecture that separates an application's data model, user interface, and control logic into three distinct components so that modifications to one component can be made with minimal impact to the others.
- MVC is often thought of as a software design pattern. However, MVC encompasses more of the architecture of an application than is typical for a design pattern. Hence the term architectural pattern may be useful (Buschmann, et al 1996), or perhaps an aggregate design pattern.

MVC Benefits

- Clarity of design
 - easier to implement and maintain
- Modularity
 - changes to one don't affect the others
 - can develop in parallel once you have the interfaces
- Multiple views
 - games, spreadsheets, powerpoint, Eclipse, UML reverse engineering,

Summary (MVC)

- The intent of MVC is to keep neatly separate objects into one of three categories
 - Model
 - The data, the business logic, rules, strategies, and so on
 - View
 - Displays the model and usually has components that allow user to edit or change the model
 - Controller
 - Allows data to flow between the view and the model
 - The controller mediates between the view and model

Model

- The Model's responsibilities
 - Provide access to the state of the system
 - Provide access to the system's functionality
 - Can notify the view(s) that its state has changed

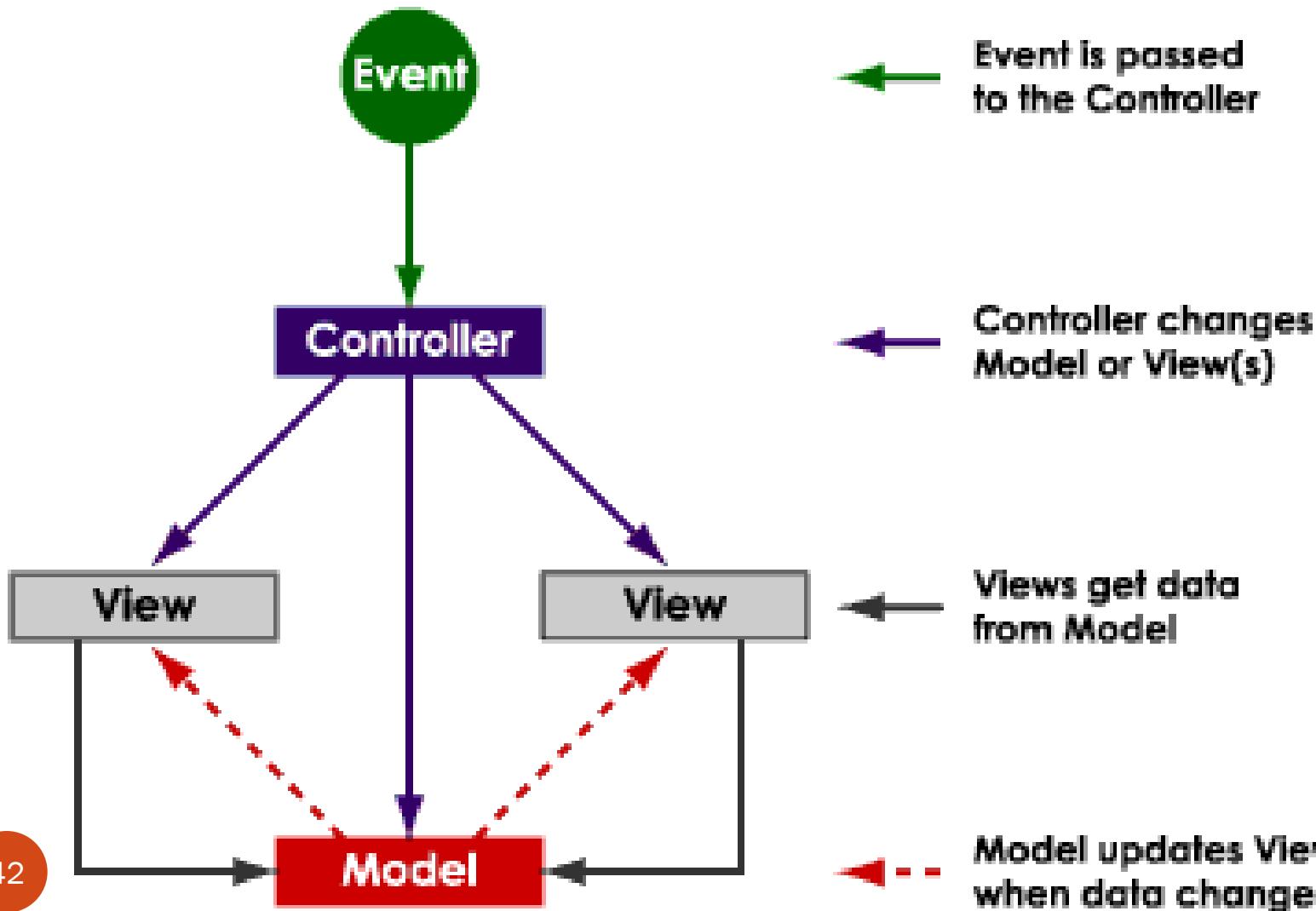
View

- The view's responsibilities
 - Display the state of the model to the user
- At some point, the model (a.k.a. the observable) must registers the views (a.k.a. observers) so the model can notify the observers that its state has changed

Controller

- The controller's responsibilities
 - Accept user input
 - Button clicks, key presses, mouse movements, slider bar changes
 - Send messages to the model, which may in turn notify its observers
 - Send appropriate messages to the view
- In Java, listeners are controllers

from <http://www.enode.com/x/markup/tutorial/mvc.html>

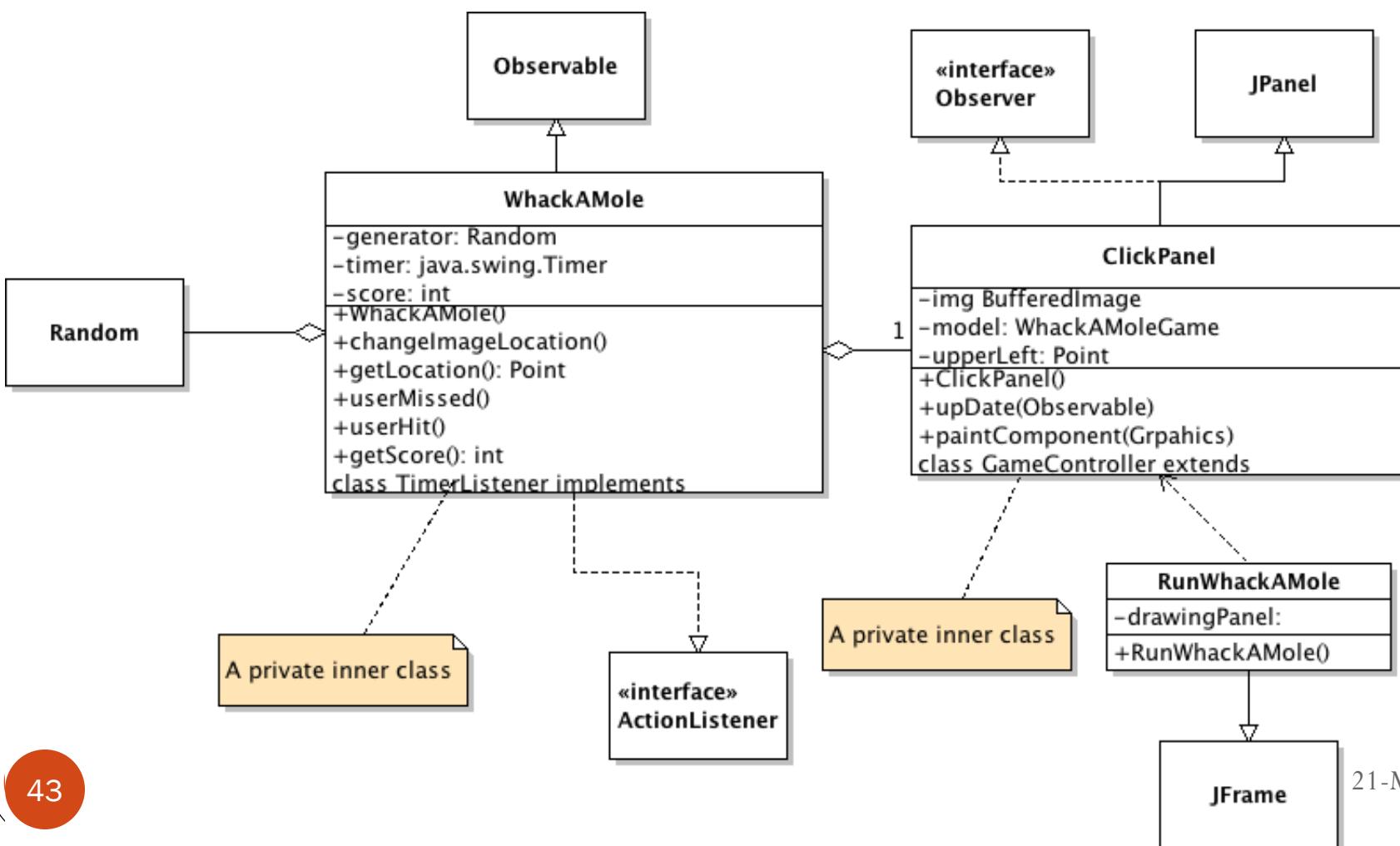


Quiz?

Model? _____

View? _____

Controller? _____



Web Technology Concept

- ▶ Internet is a global system of interconnected computer networks that connect various Domain Name System (DNS) servers, web hosting servers, home and business users, etc. Internet is a network of networks.
- ▶ Computers are connected together via a network or transmission line
- ▶ The objective of the ARPAnet project was to investigate the development of a *decentralized computer network*, The network then became known as the *Internet*
- ▶ It has since adopted a suite of protocols called the **Internet Protocol Suite** or as more commonly known as **TCP/IP**
- ▶ Now, the Internet has grown to encompass a huge number of autonomous networks

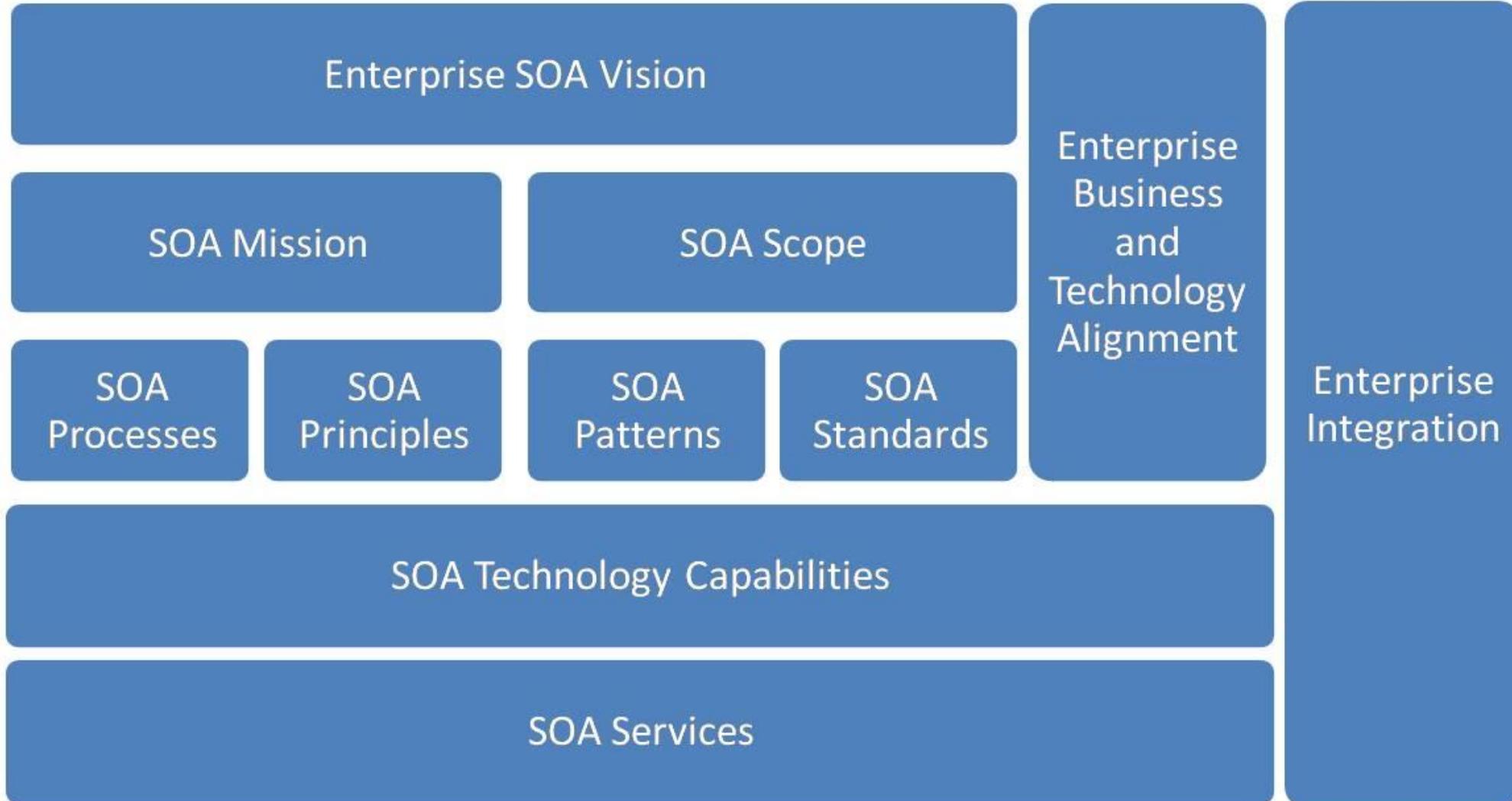
Protocols

- ▶ Protocols exist at several levels in a telecommunication connection. For example, there are protocols for the data interchange at the hardware device level and protocols for data interchange at the application program level.
- ▶ Some of the important protocols of internet are:
- ▶ TCP (Transmission Control Protocol),
- ▶ IP (Internet Protocol),
- ▶ UDP (User Datagram Protocol),
- ▶ HTTP (Hypertext Transfer Protocol),
- ▶ FTP (File Transfer Protocol) and
- ▶ SMTP (Simple Mail Transfer Protocol).

- ▶ **Transmission Control Protocol (TCP)** is a connection-oriented protocol that provides reliable, ordered, error-checked delivery of packets.
- ▶ **Internet Protocol (IP)** has the task of delivering packets from the source host to the destination host solely based on the IP addresses in the packet headers.
- ▶ **UDP** is a connection-less protocol that does not provide reliability, order or error-checking. UDP messages are referred to as datagrams and a datagram is defined as a basic transfer unit associated with a packet-switched network in which the delivery, arrival time, and order of arrival are not guaranteed by the network.
- ▶ **HTTP** is the protocol to exchange or transfer hypertext. Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text.
- ▶ **The World Wide Web** (abbreviated as WWW or W3) or the web is a system of interlinked hypertext documents accessed via the Internet. With a web browser, one can view web pages and navigate between them via hyperlinks.
- ▶ **FTP** is a standard network protocol used to transfer files from one host to another host over a TCP-based network.
- ▶ **SMTP** is an Internet standard for electronic mail (e-mail) transmission across Internet Protocol (IP) networks.

Service Oriented Architecture

- ▶ SOA is an architectural pattern in software design.
- ▶ SOA application components provide services to other components via a communications protocol, typically over a network.
- ▶ The principles of service-orientation are independent of any vendor, product or technology.



SOA Concepts and Principles

- ▶ Design Concept
 - ▶ SOA is based on the concept of a service.
 - ▶ Depending on the service design approach taken,
 - ▶ Each SOA service is designed to perform one or more activities by implementing one or more service operations.
 - ▶ SOA defines how to integrate widely disparate applications for a Web-based environment and uses multiple implementation platforms. Rather than defining an API, SOA defines the interface in terms of protocols and functionality.

Types of SOA services

- ▶ There are several types of services used in SOA systems.
 - Business services
 - Entity services
 - Functional services
 - Utility services

SOA benefits

- ▶ Ability to build business applications faster and more easily
- ▶ Easier maintenance / update
- ▶ Business agility and extensibility
- ▶ Lower total cost of ownership

- ▶ Architectures can operate independently of specific technologies and can therefore be implemented using a wide range of technologies, including:
- ▶ Web services based on WSDL and SOAP
- ▶ Messaging, e.g., with ActiveMQ, JMS, RabbitMQ
- ▶ RESTful HTTP, with Representational state transfer (REST) constituting its own constraints-based architectural style
- ▶ OPC-UA
- ▶ WCF (Microsoft's implementation of Web services, forming a part of WCF)
- ▶ Apache Thrift
- ▶ SORCER

Data Exchange Format

XML

- ▶ The eXtensible Markup Language (XML),
- ▶ Derived from the Standard Generalized Markup Language (SGML),
- ▶ Was originally envisioned as a language for defining new document formats for the World Wide Web

XML Document

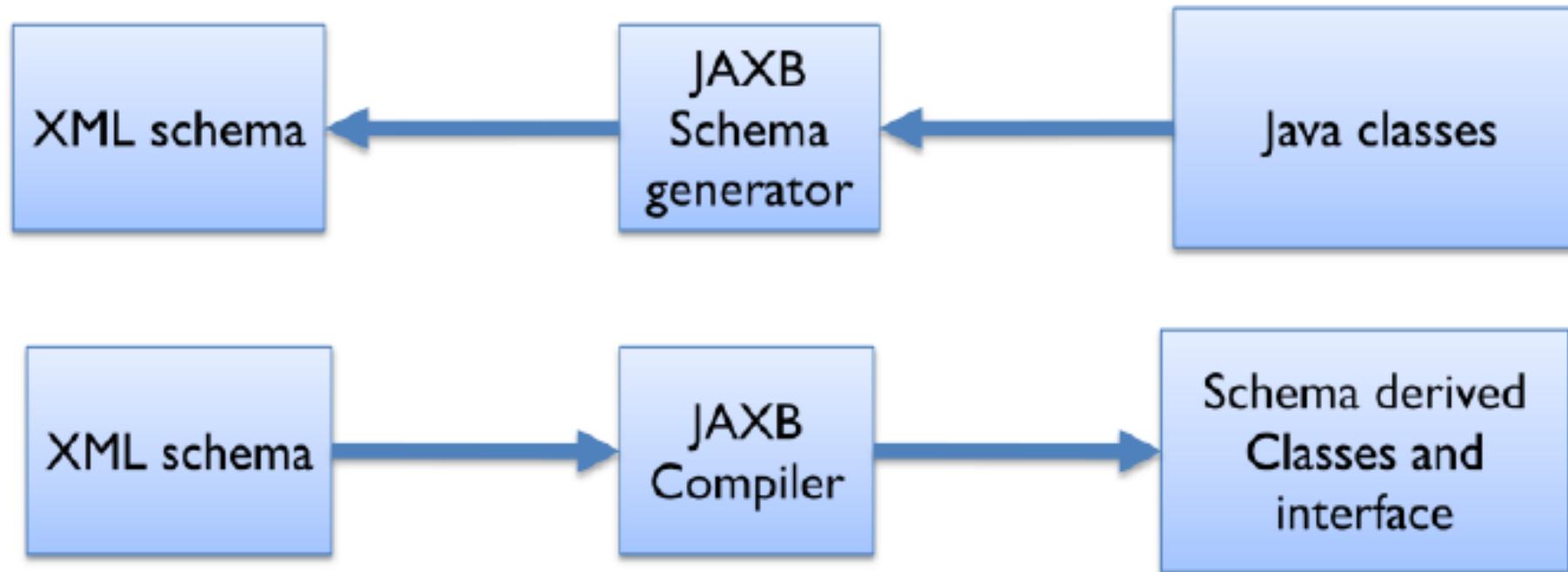
```
<?xml version="1.0" encoding="UTF-8" ?>
<order id="1234" date="05/06/2013">
    <customer first_name="James" last_name="Rorrison">
        <email>j.rorri@me.com</email>
        <phoneNumber>+44 1234 1234</phoneNumber>
    </customer>
    <content>
        <order_line item="H2G2" quantity="1">
            <unit_price>23.5</unit_price>
        </order_line>
        <order_line item="Harry Potter" quantity="2">
            <unit_price>34.99</unit_price>
        </order_line>
    </content>
    <credit_card number="1357" expiry_date="10/13" control_number="234" type="Visa"/>
</order>
```

XML Terminology

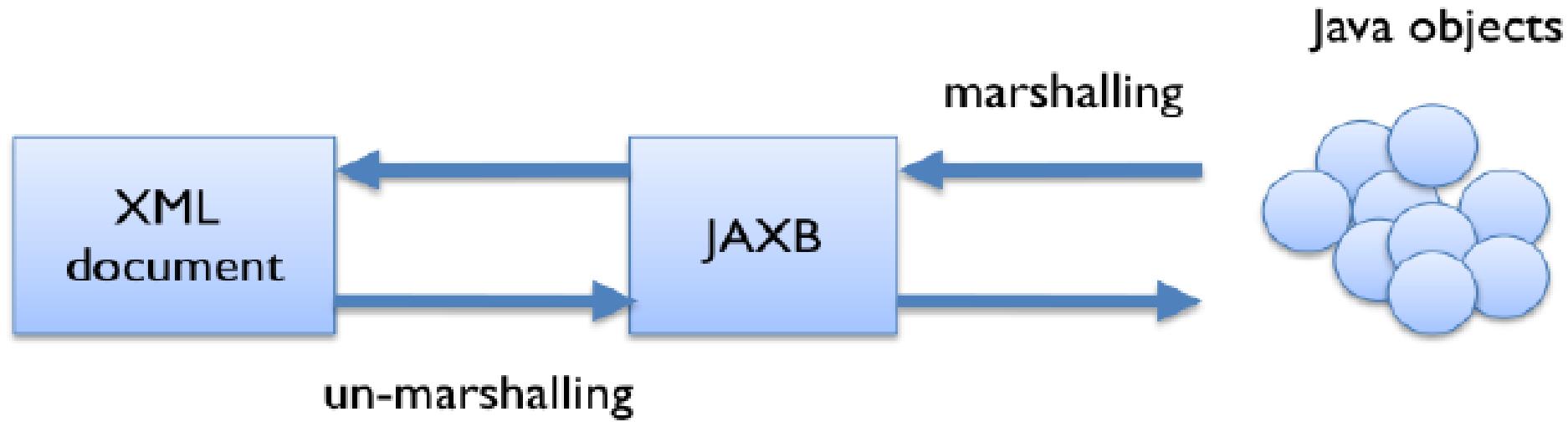
Terminology	Definition
Unicode character	An XML document is a string of characters represented by almost every legal Unicode character
Markup and content	The Unicode characters are divided into markup and content. Markups begin with the character < and end with a > (<email>) and what is not markup is considered to be content (such as j.rorri@me.com)
Tag	Tags come in three flavors of markups: start-tags (<email>), end-tags (</email>) and empty-element tags (<email/>)
Element	An element begins with a start-tag and ends with a matching end-tag (or consists only of an empty-element tag). It can also include other elements, which are called child elements. An example of an element is <email>j.rorri@me.com</email>
Attribute	An attribute consists of a name/value pair that exists within a start-tag or empty-element tag. In the following example item is the attribute of the order_line tag: <order_line item="H2G2">
XML Declaration	XML documents may begin by declaring some information about themselves, as in the following example: <?xml version="1.0" encoding="UTF-8" ?>

XML Binding in Java

JAXB: Java Architecture for XML Binding



Marshalling and un-marshalling



JSON

- ▶ JSON: JavaScript Object Notation.
- ▶ JSON is a syntax for storing and exchanging data.
- ▶ JSON is text, written with JavaScript object notation.

Exchanging Data

- ▶ When exchanging data between a browser and a server, the data can only be text.
- ▶ JSON is text, and we can convert any JavaScript object into JSON, and send JSON to the server.
- ▶ We can also convert any JSON received from the server into JavaScript objects.
- ▶ This way we can work with the data as JavaScript objects, with no complicated parsing and translations.

Why Json?

- ▶ Since the JSON format is text only, it can easily be sent to and from a server, and used as a data format by any programming language.
- ▶ JavaScript has a built in function to convert a string, written in JSON format, into native JavaScript objects:
- ▶ `JSON.parse()`
- ▶ So, if you receive data from a server, in JSON format, you can use it like any other JavaScript object.

Sending Data

```
var myObj = { "name":"John", "age":31, "city":"New York" };
var myJSON = JSON.stringify(myObj);
window.location = "demo_json.php?x=" + myJSON;
```

Receiving Data

- ▶ var myJSON = '{ "name":"John", "age":31, "city":"New York" }';
 var myObj = JSON.parse(myJSON);
 document.getElementById("demo").innerHTML = myObj.name;

RDF(Resource Description Framework)

- ▶ RDF is a framework for describing resources on the web
- ▶ RDF is designed to be read and understood by computers
- ▶ RDF is not designed for being displayed to people
- ▶ RDF is written in XML

Example of use

- ▶ Describing properties for shopping items, such as price and availability
- ▶ Describing time schedules for web events
- ▶ Describing information about web pages (content, author, created and modified date)
- ▶ Describing content and rating for web pictures
- ▶ Describing content for search engines
- ▶ Describing electronic libraries

RDF Resource, Property, and Property Value

- ▶ A **Resource** is anything that can have a URI, such as "https://www.abc.com/rdf"
- ▶ A **Property** is a Resource that has a name, such as "author" or "homepage"
- ▶ A **Property value** is the value of a Property, such as "Jan Egil Refsnes" or <https://www.abc.com>

```
<?xml version="1.0"?>

<RDF>
  <Description about="https://www.abc.com/rdf">
    <author>Jan Egil Refsnes</author>
    <homepage>https://www.abc.com</homepage>
  </Description>
</RDF>
```

How Do Web Services Work?

- ▶ A Web service enables this communication by using a combination of open protocols and standards, chiefly XML, SOAP and WSDL.
- ▶ A Web service uses XML to tag data, SOAP to transfer a message and finally WSDL to describe the availability of services.

Web Services Terminologies

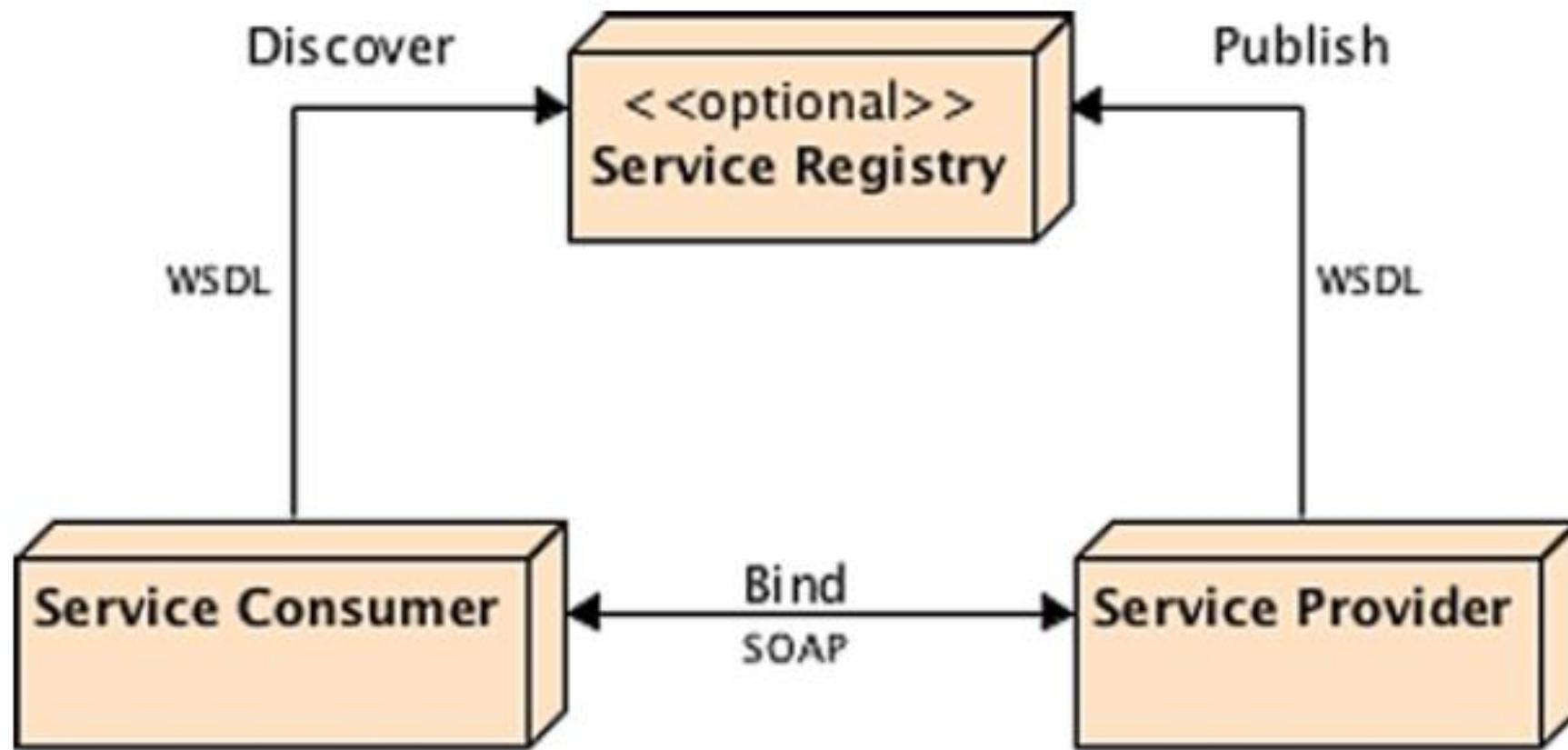
- ▶ Hypertext transfer protocol [HTTP]
- ▶ Extensible Markup Language [XML]
- ▶ Web Services Description Language [WSDL]
- ▶ SOAP

Web Services

- ▶ Web service is a realization of SOA.
- ▶ It is important to note that the SOA is an architectural model that is independent of any technology platform and Web Services the most popular SOA implementation.
- ▶ As the name implies, web services offers services over the web. This is not surprising as the choice of the Internet it already connects many different systems from all over the world.

SOAP (Simple Object Access protocol)

- ▶ It is a protocol specification for exchanging structured information in the implementation of web services in computer networks.
- ▶ It uses XML Information Set for its message format, and relies on other application layer protocols, most notably Hypertext Transfer Protocol (HTTP),
- ▶ or Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission.



SOAP Elements and Attributes

Element	Description
Envelope	Defines the message and the namespace used in the document. This is a required root element
Header	Contains any optional attributes of the message or application-specific infrastructure such as security information or network routing
Body	Contains the message being exchanged between applications
Fault	Provides information about errors that occur while the message is processed. This element is optional

SOAP Request

POST /InStock HTTP/1.1

Host: www.bookshop.org

Content-Type: application/soap+xml; charset=utf-8

Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"

soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.bookshop.org/prices">

 <m:GetBookPrice>

 <m:BookName>The Fleamarket</m:BookName>

 </m:GetBookPrice>

</soap:Body>

</soap:Envelope>

SOAP Response

POST /InStock HTTP/1.1

Host: www.bookshop.org

Content-Type: application/soap+xml; charset=utf-8

Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.bookshop.org/prices">

 <m:GetBookPriceResponse>

 <m: Price>10.95</m: Price>

 </m:GetBookPriceResponse>

</soap:Body>

</soap:Envelope>

WSDL

- ▶ WSDL stands for Web Services Description Language
- ▶ WSDL is used to describe web services
- ▶ WSDL is written in XML

Element	Description
<types>	Defines the (XML Schema) data types used by the web service
<message>	Defines the data elements for each operation
<portType>	Describes the operations that can be performed and the messages involved.
<binding>	Defines the protocol and data format for each port type

► <definitions>

<types>

 data type definitions.....

</types>

<message>

 definition of the data being communicated....

</message>

<portType>

 set of operations.....

</portType>

<binding>

 protocol and data format specification....

</binding>

</definitions>

UDDI(Universal Description, Discovery, and Integration)

- ▶ UDDI is an XML-based standard for describing, publishing, and finding web **services**.
- ▶ UDDI uses Web Service Definition Language(WSDL) to describe interfaces to web **services**
- ▶ UDDI is often compared to a telephone book's white(basic info), yellow(details about the company), and green pages(technical information). The project allows businesses to list themselves by name, product, location, or the Web services they offer.
- ▶ UDDI can communicate via SOAP, CORBA, Java RMI Protocol.
- ▶ UDDI uses Web Service Definition Language(WSDL) to describe interfaces to web services.
- ▶ UDDI is seen with SOAP and WSDL as one of the three foundation standards of web services.
- ▶ UDDI is an open industry initiative, enabling businesses to discover each other and define how they interact over the Internet.

ebXML(Electronic Business XML)

- ▶ It is a modular suite of specifications that gives businesses of any size the ability to conduct business over the Internet.

The features of ebXML are as follows:

- ▶ ebXML is an end-to-end B2B XML framework.
- ▶ ebXML is a set of specifications that enable a modular framework.
- ▶ ebXML relies on the Internet's existing standards such as HTTP, TCP/IP, MIME, SMTP, FTP, UML, and XML.
- ▶ ebXML can be implemented and deployed on virtually any computing platform.
- ▶ ebXML provides concrete specifications to enable dynamic B2B collaborations.

Development Process Management

- A software **development process** or life cycle is a structure imposed on the **development** of a software product.
- There are several models for such **processes**, each describing approaches to a variety of tasks or activities that take place during the **process**.

- More and more software development organizations implement process methodologies.
- The Capability Maturity Model (CMM) is one of the leading models. Independent assessments can be used to grade organizations on how well they create software according to how they define and execute their processes.
- There are dozens of others, with other popular ones being ISO 9000, ISO 15504, and Six Sigma.

Process Activities/Steps

Software Engineering processes are composed of many activities, notably the following:

- **Requirements Analysis**
 - Extracting the requirements of a desired software product is the first task in creating it. While customers probably believe they know what the software is to do, it may require skill and experience in software engineering to recognize incomplete, ambiguous or contradictory requirements.
- **Specification**
 - Specification is the task of precisely describing the software to be written, in a mathematically rigorous way. In practice, most successful specifications are written to understand and fine-tune applications that were already well-developed, although safety-critical software systems are often carefully specified prior to application development. Specifications are most important for external interfaces that must remain stable.
- **Software architecture**
 - The architecture of a software system refers to an abstract representation of that system. Architecture is concerned with making sure the software system will meet the requirements of the product, as well as ensuring that future requirements can be addressed.
- **Implementation**
 - Reducing a design to code may be the most obvious part of the software engineering job, but it is not necessarily the largest portion.

- **Testing**
 - Testing of parts of software, especially where code by two different engineers must work together, falls to the software engineer.
- **Documentation**
 - An important task is documenting the internal design of software for the purpose of future maintenance and enhancement.
- **Training and Support**
 - A large percentage of software projects fail because the developers fail to realize that it doesn't matter how much time and planning a development team puts into creating software if nobody in an organization ends up using it. People are occasionally resistant to change and avoid venturing into an unfamiliar area, so as a part of the deployment phase, its very important to have training classes for the most enthusiastic software users (build excitement and confidence), shifting the training towards the neutral users intermixed with the avid supporters, and finally incorporate the rest of the organization into adopting the new software. Users will have lots of questions and software problems which leads to the next phase of software.
- **Maintenance**
 - Maintaining and enhancing software to cope with newly discovered problems or new requirements can take far more time than the initial development of the software. Not only may it be necessary to add code that does not fit the original design but just determining how software works at some point after it is completed may require significant effort by a software engineer. About 60% of all software engineering work is maintenance, but this statistic can be misleading. A small part of that is fixing bugs. Most maintenance is extending systems to do new things, which in many ways can be considered new work.
-

Process Models

- A decades-long goal has been to find repeatable, predictable processes or methodologies that improve productivity and quality.

Waterfall processes

- The best-known and oldest process is the [waterfall model](#), where developers follow these steps in order. They state requirements, analyze them, design a solution approach, architect a software framework for that solution, develop code, test, deploy, and maintain. After each step is finished, the process proceeds to the next step.

Iterative processes

- [Iterative development](#) prescribes the construction of initially small but ever larger portions of a software project to help all those involved to uncover important issues early before problems or faulty assumptions can lead to disaster. Iterative processes are preferred by commercial developers because it allows a potential of reaching the design goals of a customer who does not know how to define what he wants.

Source code Management

- **Source Code Control System (SCCS)** is a version **control system** designed to track changes in **source code** and other text files during the development of a piece of software. This allows the user to retrieve any of the previous versions of the original **source code** and the changes which are stored.
- In the beginning, there was the spreadsheet, the white board and the release engineer. The release engineer ran from one cubicle to the next, trying to keep track of which developer was working on what module and when, as well as which bugs had been fixed, discovered and introduced. Needless to say, that process was fraught with problems and errors. And so, source control management systems were created.

- Given a **version number** MAJOR.MINOR.PATCH, increment the: **MAJOR version** when you make incompatible API changes, **MINOR version** when you add functionality in a backwards-compatible manner, and. **PATCH version** when you make backwards-compatible bug fixes.
- **Version control** is a system that records changes to a file or set of files over time so that you can recall specific versions later. For the examples in this book, you will use software source code as the files being **version controlled**, though in reality you can do this with nearly any type of file on a computer.

- A component of software configuration **management**, version control, also known as revision control or **source control**, is the **management** of changes to documents, computer programs, large web sites, and other collections of information.
- Tools to be used ,SVN, Git etc

- Source Control Management (SCM) is all about the way software changes are made. It has a number of goals which are fundamentally geared toward ensuring that development teams can deliver higher quality code changes at faster speeds. By improving tracking, visibility, collaboration and control throughout the release lifecycle, SCM tools provide more creativity, freedom and possibilities for developers when undertaking complex and challenging work. Moreover, SCM can protect the original source files from any kind of mishap and enables all team members to look at who has made what changes at what point.

Continuous Integration

- **Continuous Integration (CI)** involves producing a clean build of the system several times per day, usually with a tool like Cruise Control, which uses Ant and various source-control systems. **Agile** teams typically configure **CI** to include automated compilation, unit test execution, and source control **integration**.
- **Continuous integration (CI)** is a software engineering practice in which isolated changes are immediately tested and reported on when they are added to a larger code base. ... **Continuous integration** software **tools** can be used to automate the testing and build a document trail.

- Jenkins is an open-source **CI** tool written in **Java**. It originated as the fork of Hudson when the Oracle bought the Sun Microsystems. Jenkins is a cross-platform **CI** tool and it offers configuration both through GUI interface and console commands.
- **Continuous integration** is a **DevOps** software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run. ... With **continuous integration**, developers frequently commit to a shared repository using a version control system such as Git.

Software Testing

- *Software Testing* is the process of identifying the *correctness and quality* of software program.
- The purpose is to check whether the software satisfies the specific requirements, needs and expectations of the customer. In other words, testing is executing a system or application in order to find software *bugs, defects or errors*.

Ways of Software Testing

- ***Manual Testing:*** *Test Cases executed manually.*
- ***Automation Testing:*** *Testing performed with the help of automation tools.*

Who does Testing?

- Software Tester
- Software Developer
- Project Lead/Manager
- End User

When to Start Testing?

- During the requirement gathering phase, the analysis and verification of requirements are also considered as testing.
- Reviewing the design in the design phase with the intent to improve the design is also considered as testing.
- Testing performed by a developer on completion of the code is also categorized as testing.

ISO/IEC 9126 standards

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

Software Testing Tools

- HP Quick Test Professional
- Selenium
- IBM Rational Functional Tester
- SilkTest
- TestComplete
- Testing Anywhere
- WinRunner
- LoadRunner
- Visual Studio Test Professional
- WATIR

Testing Method

Black-Box Testing

- The technique of testing without having any knowledge of the interior workings of the application is called black-box testing. The tester is oblivious to the system architecture and does not have access to the source code.

White-Box Testing

- White-box testing is the detailed investigation of internal logic and structure of the code. White-box testing is also called **glass testing** or **open-box testing**. In order to perform **white-box** testing on an application, a tester needs to know the internal workings of the code.

Grey-Box Testing

- Grey-box testing is a technique to test the application with having a limited knowledge of the internal workings of an application. In software testing, the phrase the more you know, the better carries a lot of weight while testing an application.

Level of Testing

Functional Testing

- This is a type of black-box testing that is based on the specifications of the software that is to be tested. The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for.
- **Non-functional testing** involves testing a software from the requirements which are nonfunctional in nature but important such as performance, security, user interface, Load balancing between servers etc.

Software Documentation (UML Diagram and Tools)

- UML is a way of visualizing a software program using a collection of diagrams.
- The notation has evolved from the work of Grady Booch, James Rumbaugh, Ivar Jacobson, and the Rational Software Corporation to be used for object-oriented design, but it has since been extended to cover a wider variety of software engineering projects.
- Today, UML is accepted by the Object Management Group (OMG) as the standard for modeling software development

What is UML?

- UML stands for Unified Modeling Language. UML 2.0 helped extend the original UML specification to cover a wider portion of software development efforts including agile practices.
- Improved integration between structural models like **class diagrams** and behavior models like activity diagrams.
- Added the ability to define a hierarchy and **decompose a software system into components and sub-components**.

Structural UML diagrams

- Class diagram
- Package diagram
- Object diagram
- Component diagram
- Composite structure diagram
- Deployment diagram

Behavioral UML diagrams

- Activity diagram
- Sequence diagram
- Use case diagram
- State diagram
- Communication diagram
- Interaction overview diagram
- Timing diagram

- **Class Diagram**

[Class diagrams](#) are the backbone of almost every object-oriented method, including UML. They describe the static structure of a system.

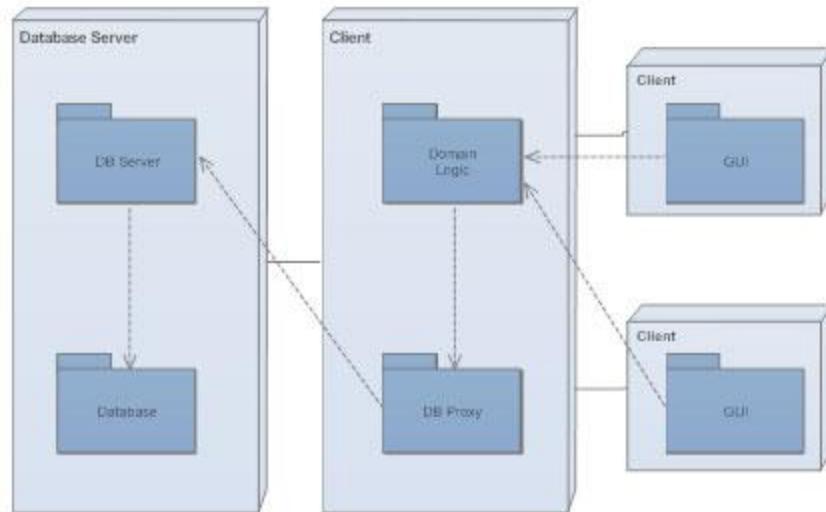
- **Package Diagram**

Package diagrams are a subset of class diagrams, but developers sometimes treat them as a separate technique.

Package diagrams organize elements of a system into related groups to minimize dependencies between packages.

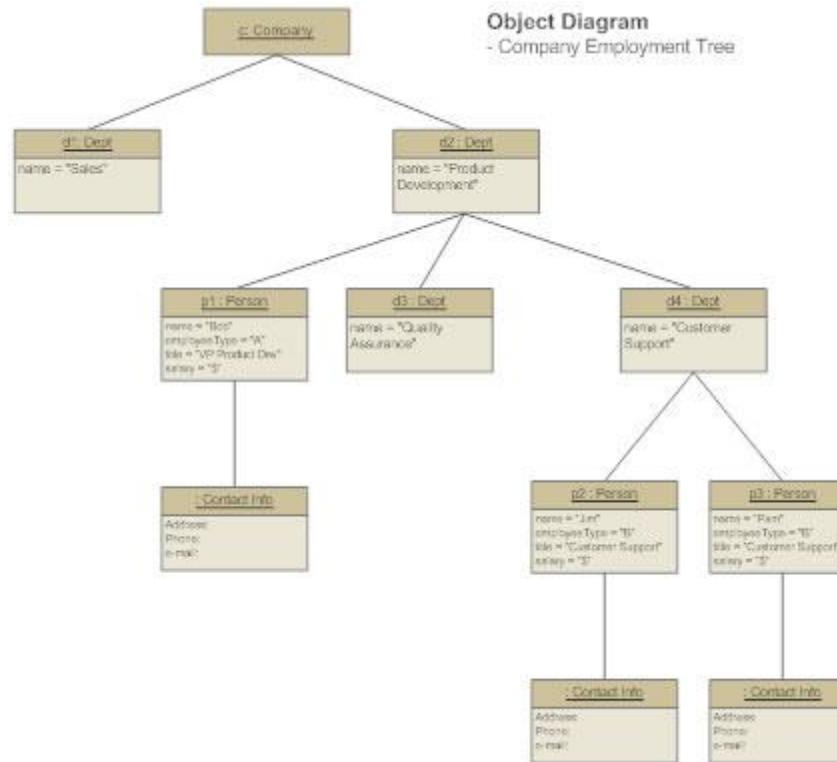
UML Package Diagram - Encapsulation

-



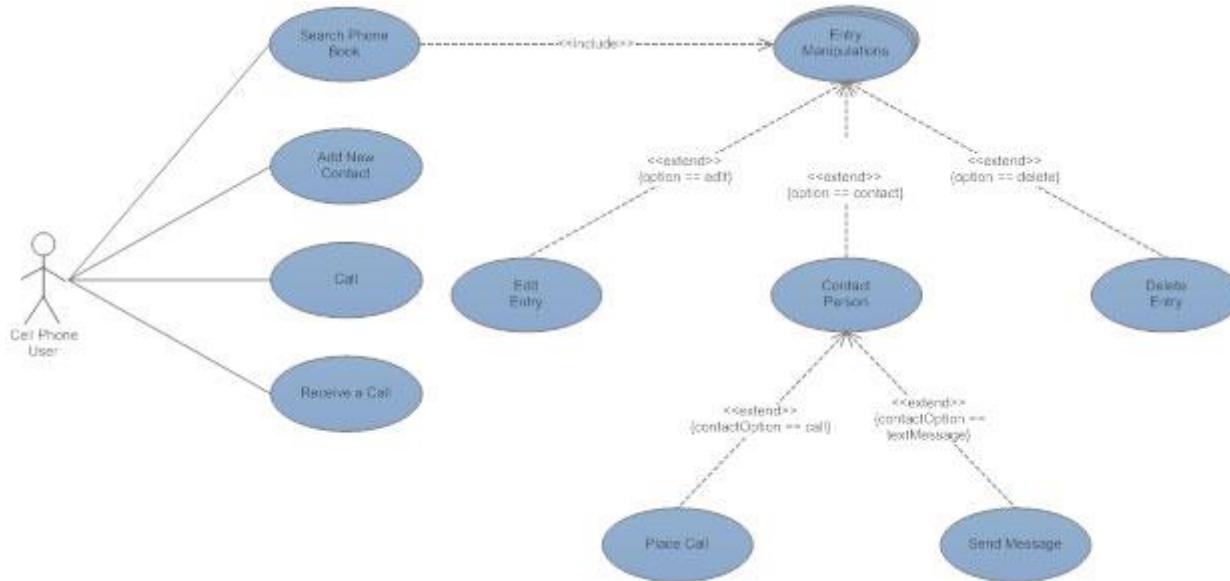
• Object Diagram

Object diagrams describe the static structure of a system at a particular time. They can be used to test class diagrams for accuracy.



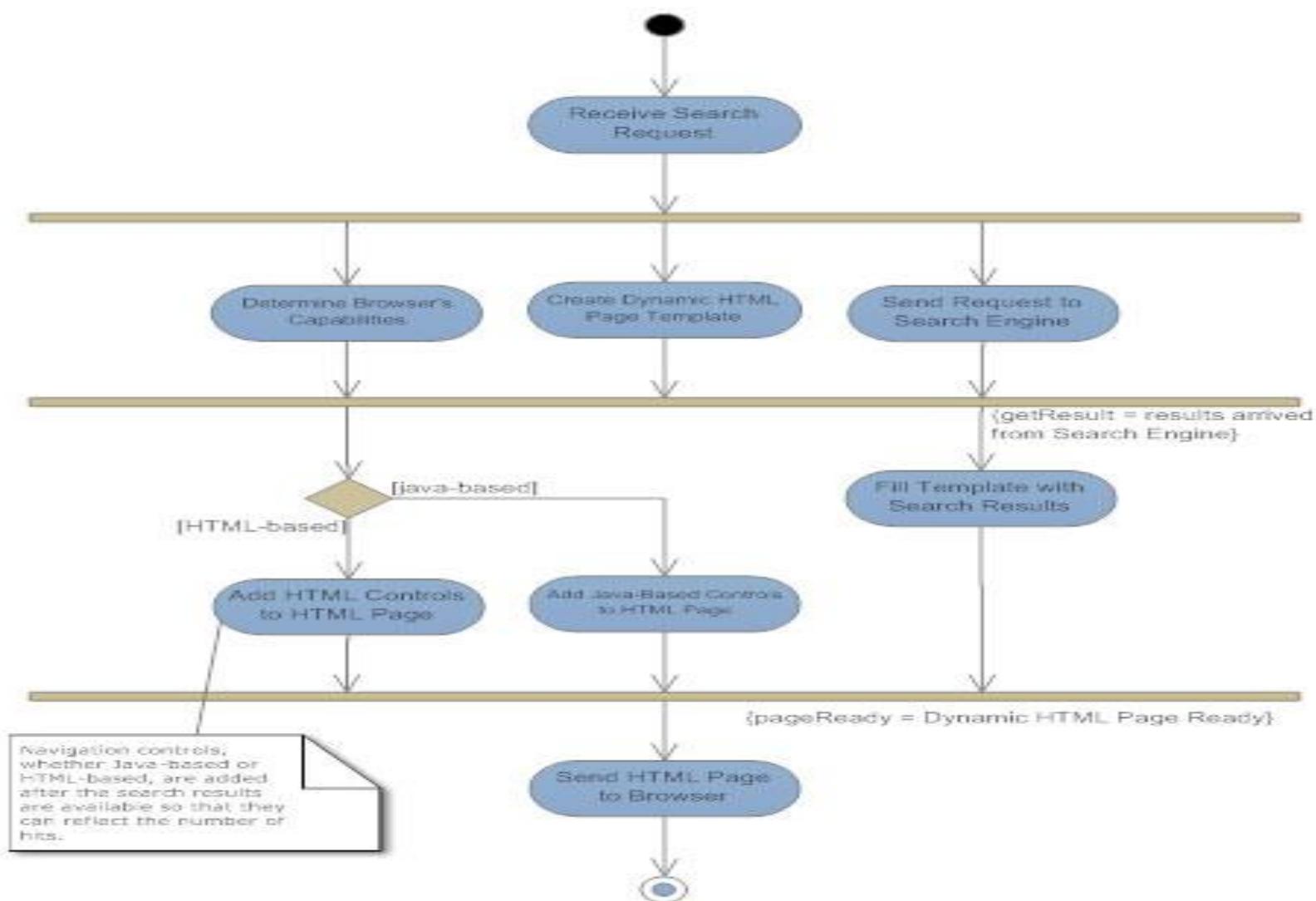
- **Use Case Diagram**

Use case diagrams model the functionality of a system using actors and use cases.



- Activity diagrams illustrate the dynamic nature of a system by modeling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operation.

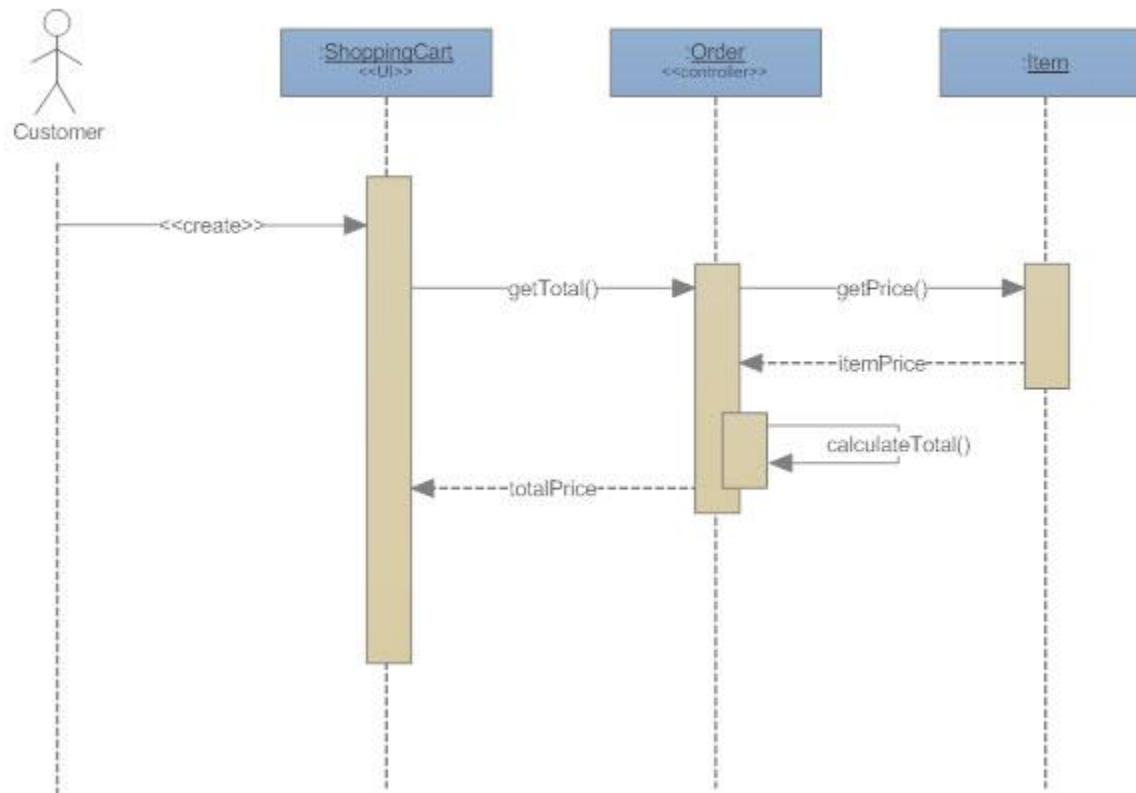
UML Activity Diagram: Web Site



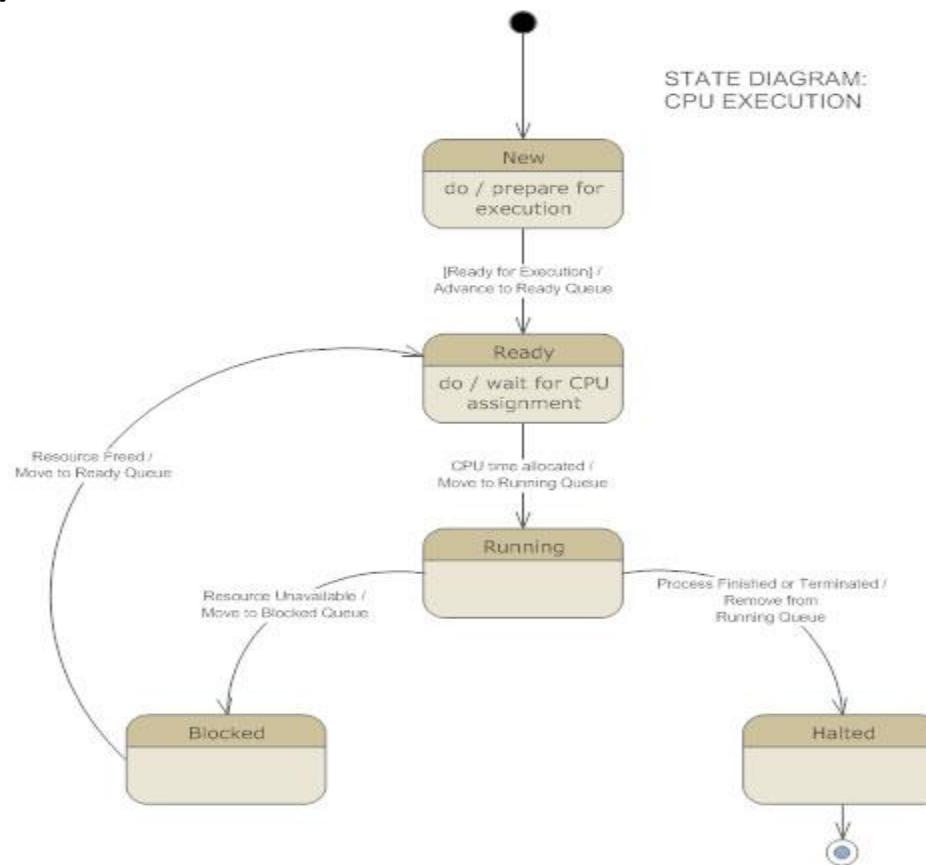
- # Sequence Diagram

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

Sequence Diagram: Shopping Cart



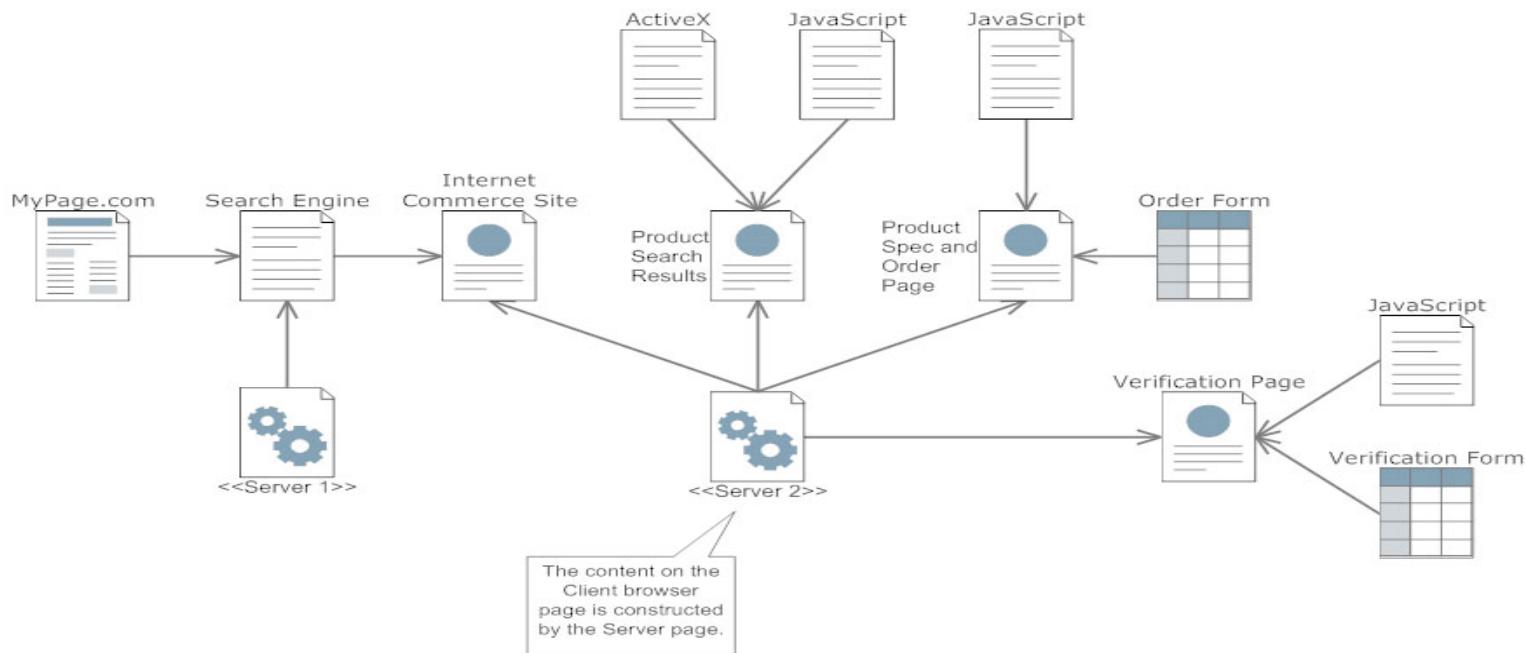
- Statechart diagrams, now known as state machine diagrams and state diagrams describe the dynamic behavior of a system in response to external stimuli. State diagrams are especially useful in modeling reactive objects whose states are triggered by specific events



- Component diagrams describe the organization of physical software components, including source code, run-time (binary) code, and executables.

Web Applications

See Also: UML Class Diagram - Web Transactions



Why Do We Use UML?

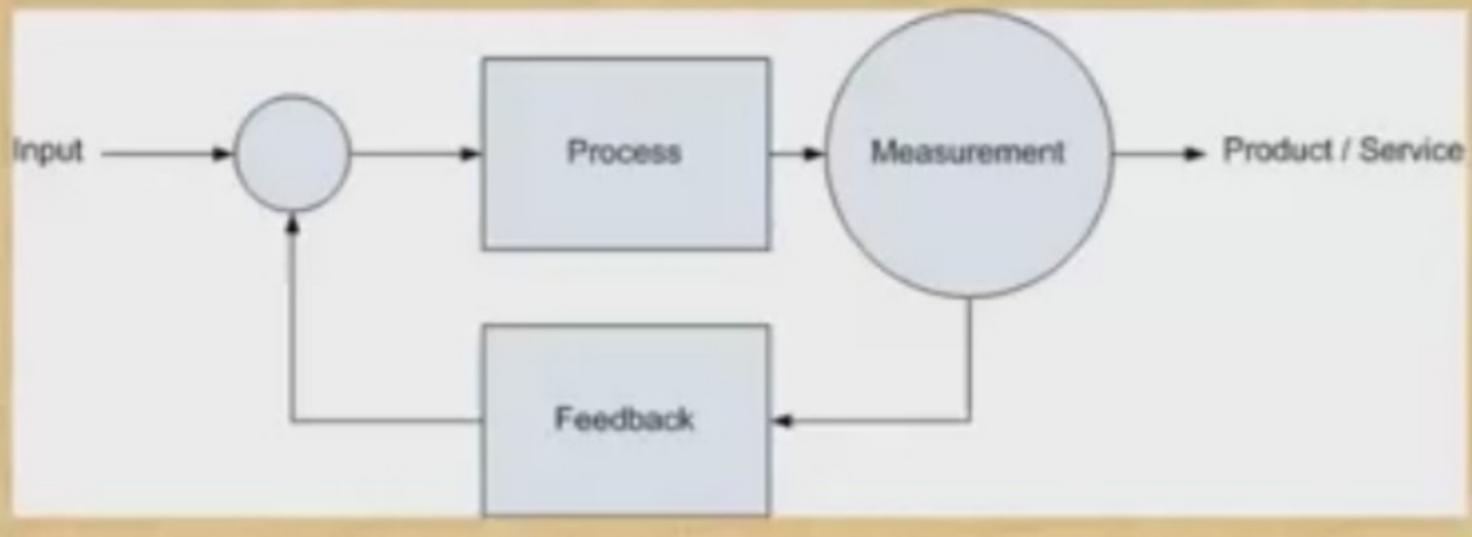
- A complex enterprise application with many collaborators will require a solid foundation of planning and clear, concise communication among team members as the project progresses.
- Tools can be used – UMLet, smartdraw **draw.io**, [Lucidchart](#) etc.

Business Processes

An ongoing set of related activities, the output of which results in value to:

1. An organization
2. Its business partners
3. Its customers

Business Process



Types of Business Processes

- Functional processes e.g., hiring employees, managing bills of materials, applying Internet use policy
- Cross functional processes e.g., procurement process

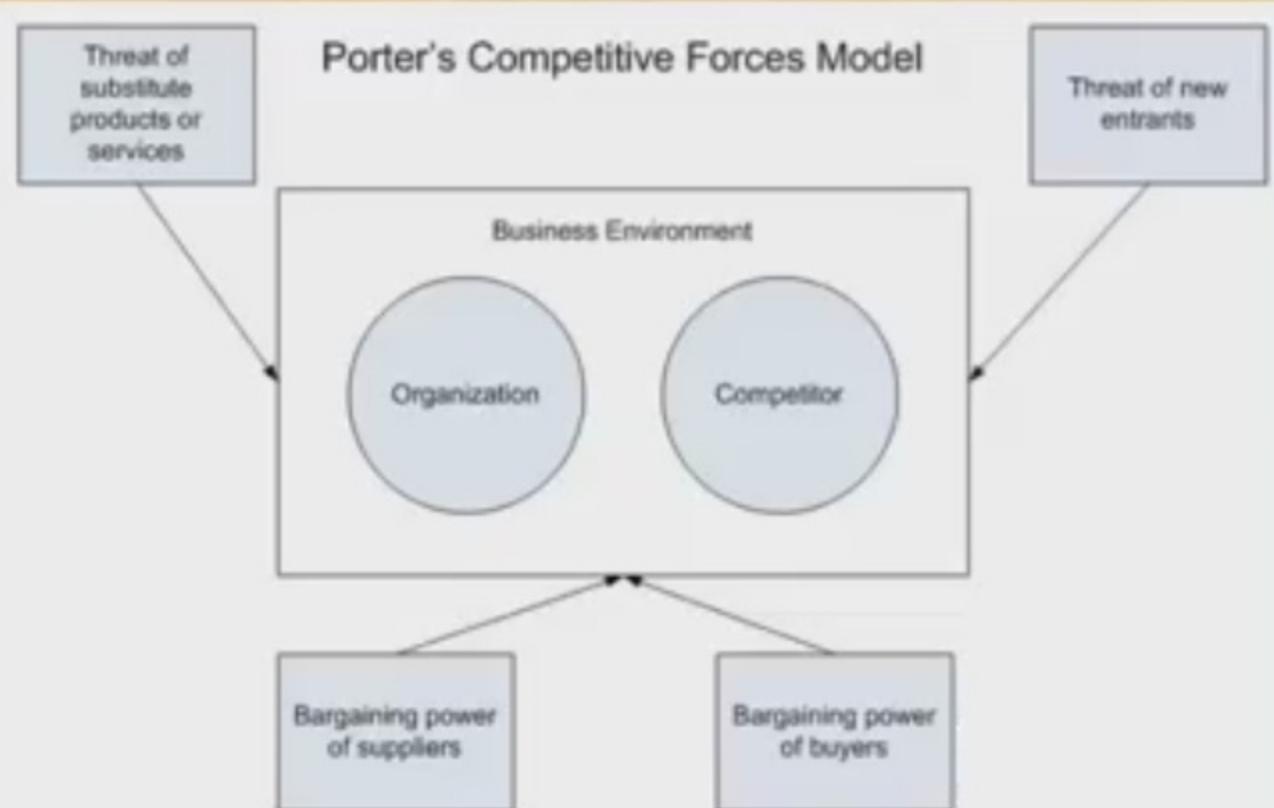
Business Process Reengineering

Redesigning of business processes to improve efficiency and effectiveness

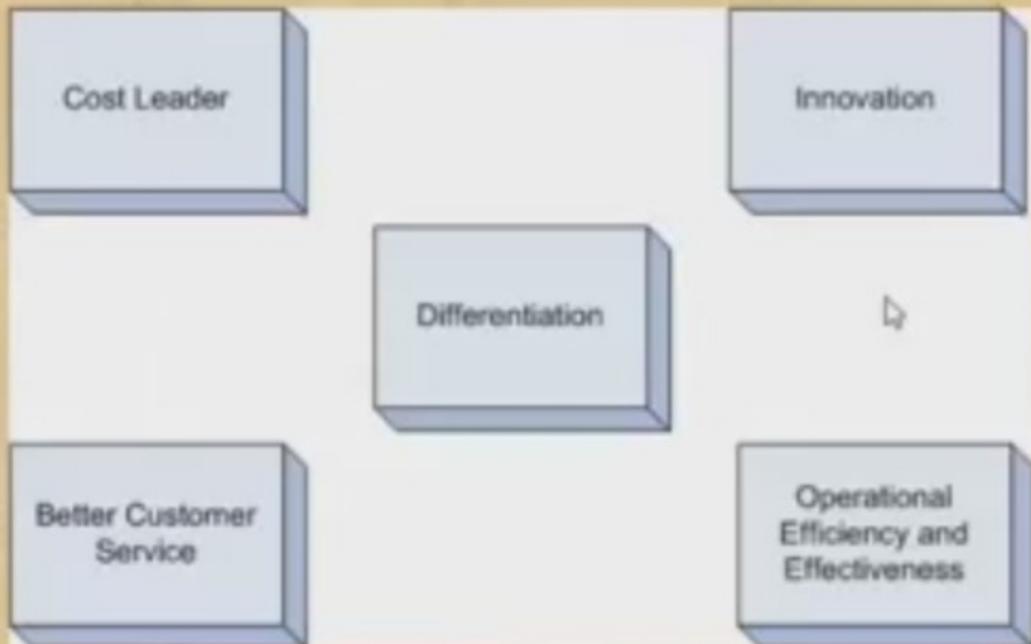
Business Process Management

Management technique to support the design, analysis, implementation, management , and optimization of business processes

Porter's Competitive Forces Model



Competitive Advantage Strategies



Characteristics of Business – IT Alignment

- IT is an engine of innovation
- Customer service receives paramount importance
- Business and IT professionals are rotated across departments and job functions
- Organizational goals are crystal clear to business and IT personnel
- IT personnel understand the functioning of the organization's revenue stream
- Organizational culture is vibrant and inclusive

Definition

"Business Process Management (BPM) is a disciplined approach to identify, design, execute, document, monitor, control, and measure both automated and non-automated business processes to achieve consistent, targeted results consistent with an organization's strategic goals. BPM involves the deliberate, collaborative and increasingly technology-aided definition, improvement, innovation, and management of end-to-end business processes that drive business results, create value, and enable an organization to meet its business objectives with more agility."

- **2. MEANING** “A functional information system is a system that provides detailed information for a specific type of activity or related group of activities, as well as summarized information for management control of such activities”.
 - **3. CHARACTERISTICS** Many small changes in a large database
Systematic records (mostly numerical)
Routine actions & updating Data preparation is a large & important effort
 - **4. EQUIPMENT REQUIREMENTS OF FUNCTIONAL INFORMATION SYSTEMS** Large auxiliary storage
Dual use files
Moderate input/output requirements
Flexible printing capacity
Offline data entry
Often difficult to define the problem
Needs fast random access to large storage capacity
Organization of computer storage is difficult
Versatile inquiry stations desirable
 - **5. FUNCTIONAL INFORMATION SYSTEM** **Information System for Marketing (MIS)**
Information System for HR Management (HRIS)
Information System for Accounts (AIS)
Information System for Production & Manufacturing
Information System for Finance Management
-

Functional Information System is based on the various business functions such as Production, Marketing, Finance and Personnel etc. These departments or functions are known as functional areas of business. Each functional area requires applications to perform all information processing related to the function. The popular functional areas of the business organization are:

-
- (i) Financial Information System
- (ii) Marketing Information System
- (iii) Production/Marketing Information System
- (iv) Human Resource Information System

Financial Information System:

-
- Financial information system is a sub-system of organizational management information system. This sub-system supports the decision-making process of financial functions at the level of an organization.

Marketing Information System

This sub-system of management information system provides information about various functions of the marketing system of an organization. Marketing is another functional area of the business organization, which is engaged in marketing (selling) of its products to its customers.

Important functions of the marketing process include the following.

- o The marketing identification function
- o The purchase motivation function.
- o The product adjustment function
- o The physical distribution function
- o The communication function
- o The transaction function
- o The post-transaction function

Production /manufacturing Information System

Manufacturing or production information system provides information on production /operation activities of an organization and thus facilitates the decision-making process of production managers of an organization. The main decisions to be taken in manufacturing system are:

- o Product Design
- o _____

Human Resources Information System

This functional information system supports the functions of human resource management of an organization. The human resource management function, in its narrow sense, it also known as personnel management .The function involves:

- o Manpower planning.
- o Staffing

- Training and development
- Performance evaluation, and
- Separation activities

The Organizational Process

Organizing, like planning, must be a carefully worked out and applied process. This process involves determining what work is needed to accomplish the goal, assigning those tasks to individuals, and arranging those individuals in a decision-making framework (organizational structure). The end result of the organizing process is an **organization** — a whole consisting of unified parts acting in harmony to execute tasks to achieve goals, both effectively and efficiently.

A properly implemented organizing process should result in a work environment where all team members are aware of their responsibilities. If the organizing process is not conducted well, the results may yield confusion, frustration, loss of efficiency, and limited effectiveness.

In general, the organizational process consists of five steps (a flowchart of these steps is shown in Figure 1):

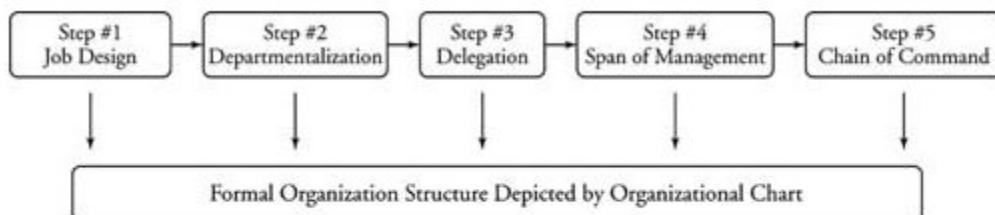


Figure 1
The organizational process.

1. Review plans and objectives.

Objectives are the specific activities that must be completed to achieve goals. Plans shape the activities needed to reach those goals. Managers must examine plans initially and continue to do so as plans change and new goals are developed.

2. Determine the work activities necessary to accomplish objectives.

Although this task may seem overwhelming to some managers, it doesn't need to be. Managers simply list and analyze all the tasks that need to be accomplished in order to reach organizational goals.

3. Classify and group the necessary work activities into manageable units.

A manager can group activities based on four models of departmentalization: functional, geographical, product, and customer.

4. Assign activities and delegate authority.

Managers assign the defined work activities to specific individuals. Also, they give each individual the authority (right) to carry out the assigned tasks.

5. Design a hierarchy of relationships.

A manager should determine the vertical (decision-making) and horizontal (coordinating) relationships of the organization as a whole. Next, using the organizational chart, a manager should diagram the relationships.

NOC Infrastructure

Applications

Middleware

OS

Virtualization

Servers

Storage

Database

Network

Compliance

Security

Operations

Management

Monitoring



People



Process



Technology

CUSTOMER
PORTAL

Comparison chart

	Data	Information
Meaning	Data is raw, unorganized facts that need to be processed. Data can be something simple and seemingly random and useless until it is organized.	When data is processed, organized, structured or presented in a given context so as to make it useful, it is called information.
Example	Each student's test score is one piece of data. "Data" comes from a singular Latin word, datum, which originally meant "something given." Its early usage dates back to the 1600s. Over time "data" has become the plural of datum.	The average score of a class or of the entire school is information that can be derived from the given data. "Information" is an older word that dates back to the 1300s and has Old French and Middle English origins. It has always referred to "the act of informing, " usually in regard to education, instruction, or other knowledge communication.

Six reasons why information systems are so important for business today include:

1. Operational excellence
2. New products, services, and business models
3. Customer and supplier intimacy
4. Improved decision making
5. Competitive advantage
6. Survival

Here is one other answer to this question. The emergence of a global economy, transformation of industrial economies, transformation of the business enterprise, and the emergence of digital firm make information systems essential in business today. Information system is a foundation for conducting business today. In many businesses, survival and the ability to achieve strategic business goals is difficult without extensive use of information technology. There are six reasons or objectives why businesses use information system:

1. Operational excellence. Business improve the efficiency of their operations in order to achieve higher profitability. Information systems are important tools available to managers for achieving higher levels of efficiency and productivity in business operations. A good example is Wal-Mart that uses a RetailLink system , which digitally links its suppliers to every one of Wal-Mart's stores. as soon as a customer purchase an item , the supplier is monitoring the item , knows to ship a replacement to the shelf.

2. New products, services, and business models. Information system is a major tool for firms to create new products and services, and also an entirely new business models. A business model describe how a company produces, delivers, and sells a product or service to create wealth.

Example: Apple inc transformed an old business model based on its iPod technology platform that included iPod, the iTunes music service, and the iPhone.

3. Customer/supplier intimacy. When a business serves its customers well, the customers generally respond by returning and purchasing more. this raises revenue and profits. The more a business engage its suppliers, the better the suppliers can provide vital inputs. This lower costs. Example: The Mandarin Oriental in manhattan and other high-end hotels exemplify the use of information systems and technology to achieve customer intimacy. they use computers to keep track of guests' preferences, such as their preferred room temperature, check-in time, television programs.

4. Improved decision making. Many managers operate in an information bank, never having the right information at the right

time to make an informed decision. These poor outcomes raise costs and lose customers. Information system made it possible for the managers to use real time data from the marketplace when making decision. Example: Verizon Corporation uses a Web-based digital dashboard to provide managers with precise real -time information on customer complains, network performance.. Using this information managers can immediately allocate repair resources to affected areas, inform customers of repair efforts and restore service fast.

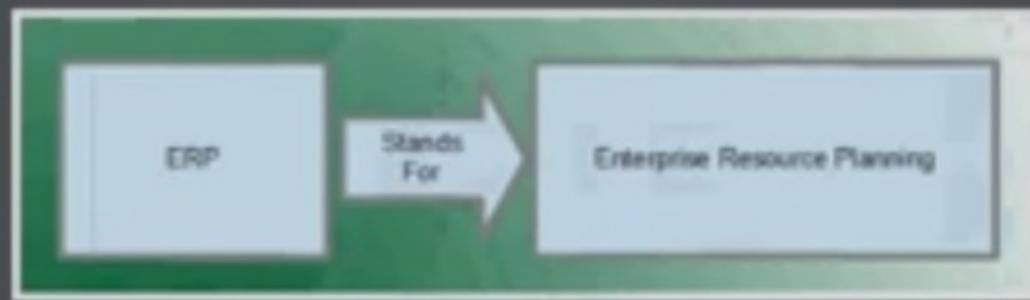
5. Competitive advantage. When firms achieve one or more of these business objectives(operational excellence, new products, services, and business models, customer/supplier intimacy, and improved decision making) chances are they have already achieved a competitive advantage. Doing things better than your competitors, charging less for superior products, and responding to customers and suppliers in real time all add up to higher sales, and higher profits. Example: Toyota Production System focuses on organizing work to eliminate waste, making continues improvements, TPS is based on what customers have actually ordered.

6. Day to day survival. Business firms invest in information system and technology because they are necessities of doing business. This necessities are driven by industry level changes. Example: Citibank introduced the first automatic teller machine to attract customers through higher service levels, and its competitors rushed to provide ATM's to their customers to keep up with Citibank. providing ATMs services to retail banking customers is simply a requirement of being in and surviving in the retail banking business. Firm turn to information system and technology to provide the capability to respond to these.

Information systems are the foundation for conducting business today. In many industries, survival and even existence without extensive use of IT is inconceivable, and IT plays a critical role in increasing productivity. Although information technology has become more of a commodity, when coupled with complementary changes in organization and management, it can provide the foundation for new products, services, and ways of conducting business that provide firms with a strategic advantage.

THE TERM ERP

The term "ERP" stands for "Enterprise Resource Planning".



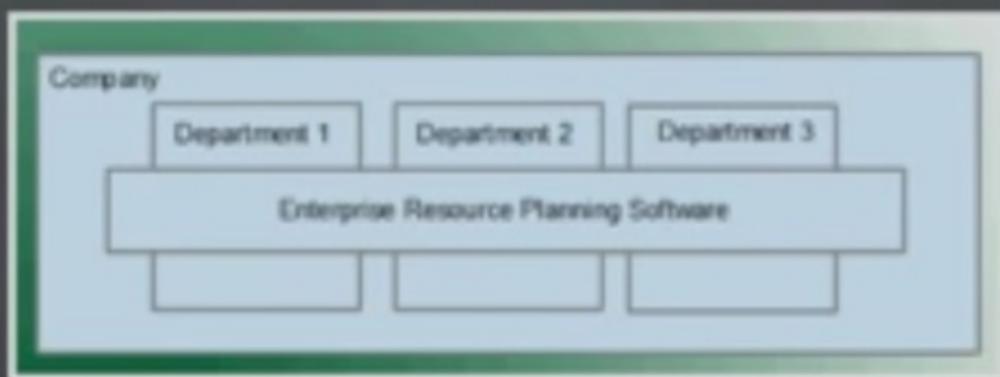
TYPE OF SOFTWARE

ERP is not a name of any software, instead it is a class (or type) of software.



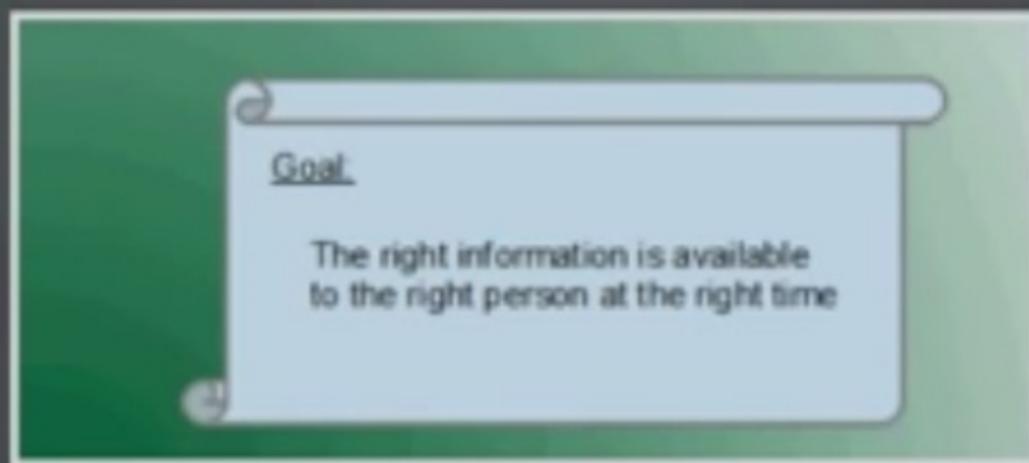
FUNCTIONALITY

An ERP software provides an end to end information management solution for a company. The software could be used by all departments of the company to manage the information.



OBJECTIVE

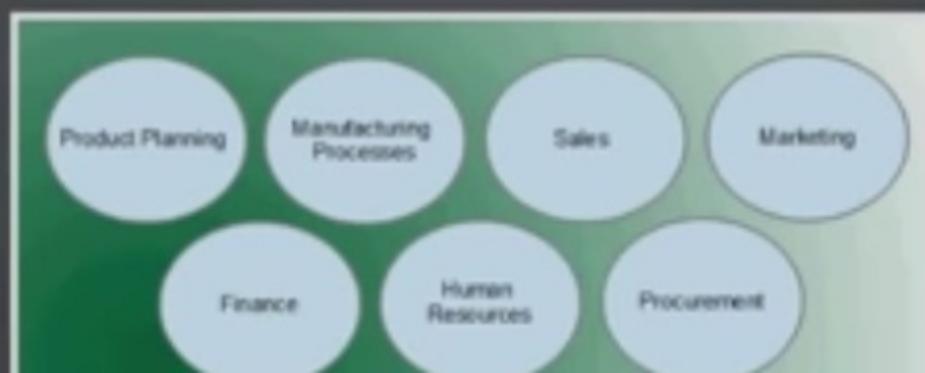
The goal is that right information is available to the right person at the right time.



BUSINESS AREAS

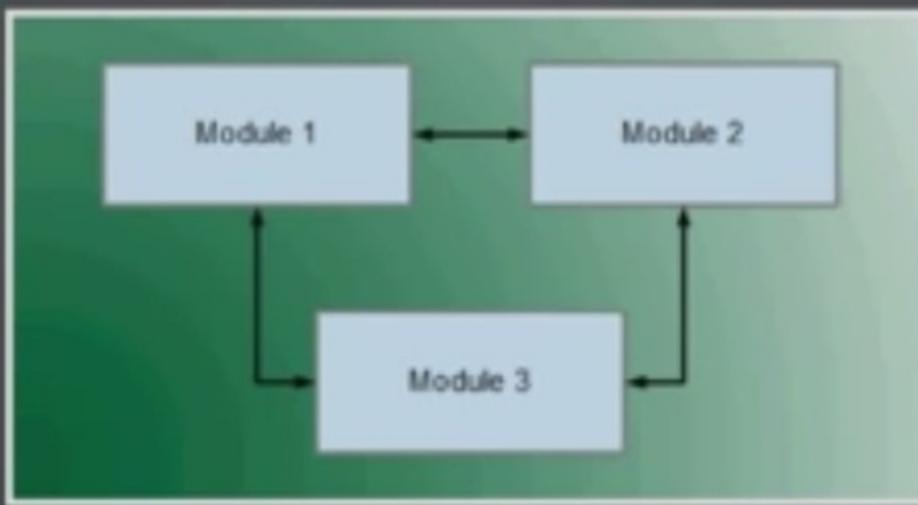
ERP software integrates all areas of operations including:

- Product Planning
- Manufacturing Processes
- Sales
- Marketing
- Finance
- Human Resources
- Procurement



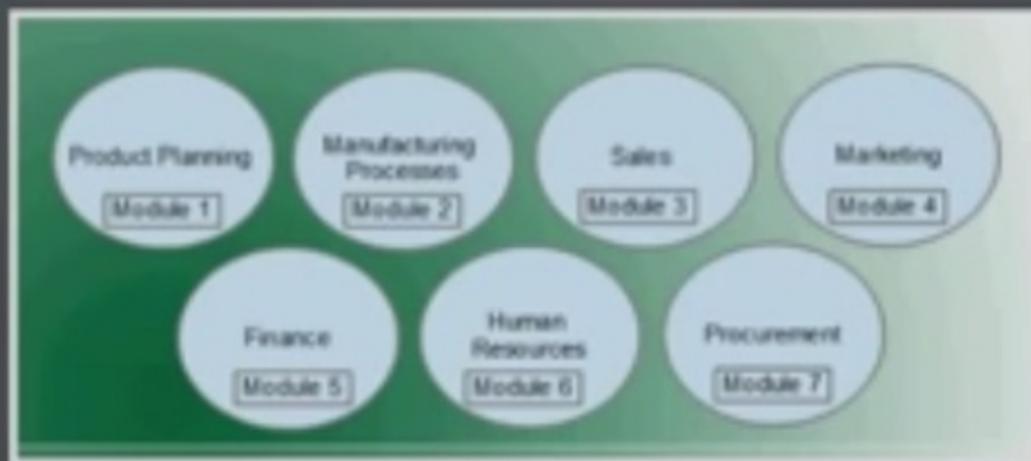
MODULAR AND INTEGRATED

An ERP software is typically modular but integrated. Meaning it consists of multiple modules that are connected to each other.



FOCUS OF A MODULE

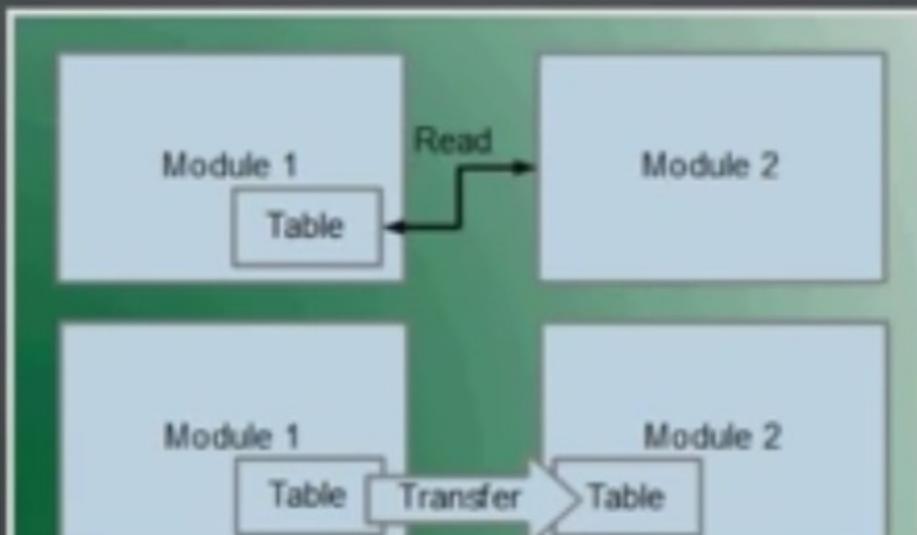
Each module is focused on one area of business processes e.g. finance, human resources etc.



COMMUNICATION AMONG MODULES

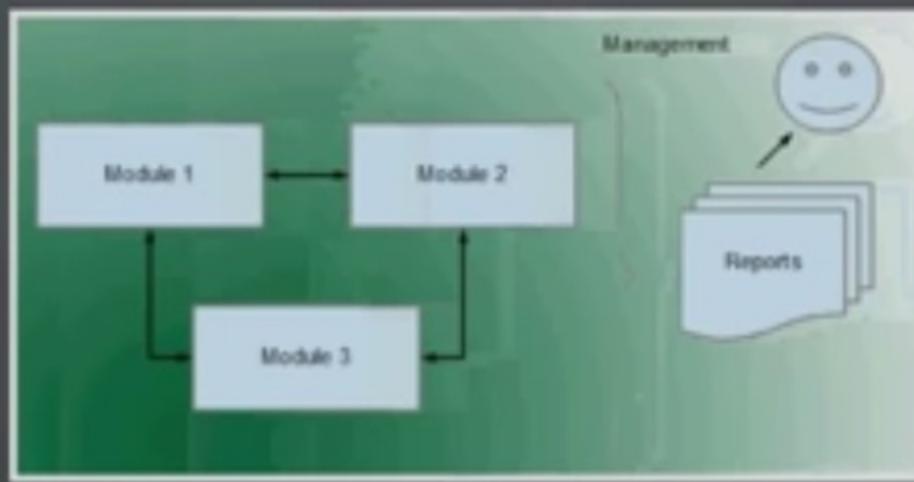
When we say modules are integrated that means:

- A module could share information stored in another module e.g. list of suppliers etc.
- Also information could flow from one module to the other e.g. accounting entries etc.



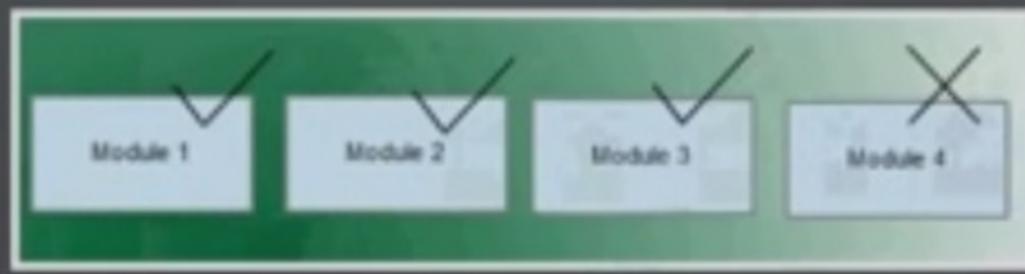
MANAGEMENT REPORTING

Since modules are connected, management of a company could run reports on any aspects of the business to get a complete view of activities. Reports help executives make strategic decisions.



LICENSING

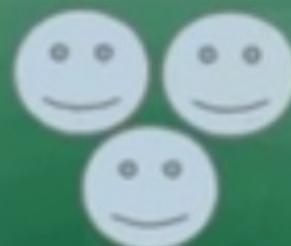
Modules could be individually purchased based on what best meets the specific needs and technical capabilities of the company.



TYPES OF USERS

The end-users of ERP software could be divided into these groups:

- Business users: Performs day to day operations e.g. data entry, operational reports etc
- Management or executives: Run reports and perform inquiries that would help them in decision making



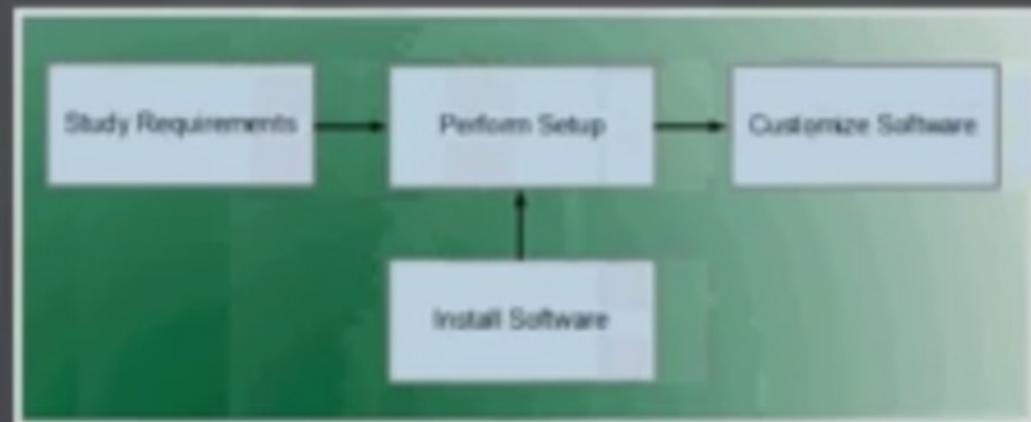
Business Users



Management

IMPLEMENTATION

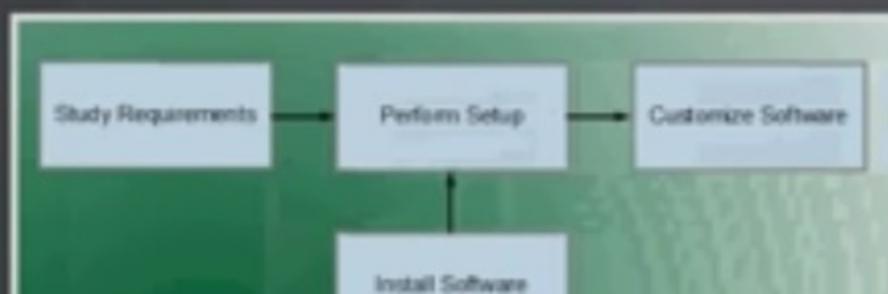
Term "implementation" is used to make the ERP software ready to be used by the company.



IMPLEMENTATION PROCESS

The process involves:

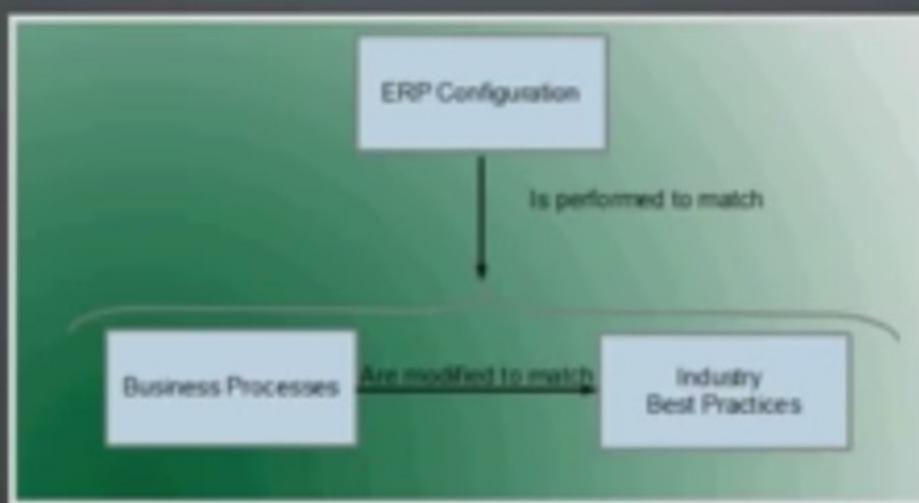
- Study the business requirements. Find out how the ERP system should be behaving.
- Setup or configure the software such that it starts working as per business requirements. By this time the software must be installed and available for setup.
- Fill the gaps between business requirements and the functionality offered by ERP software.



INDUSTRY BEST PRACTICES

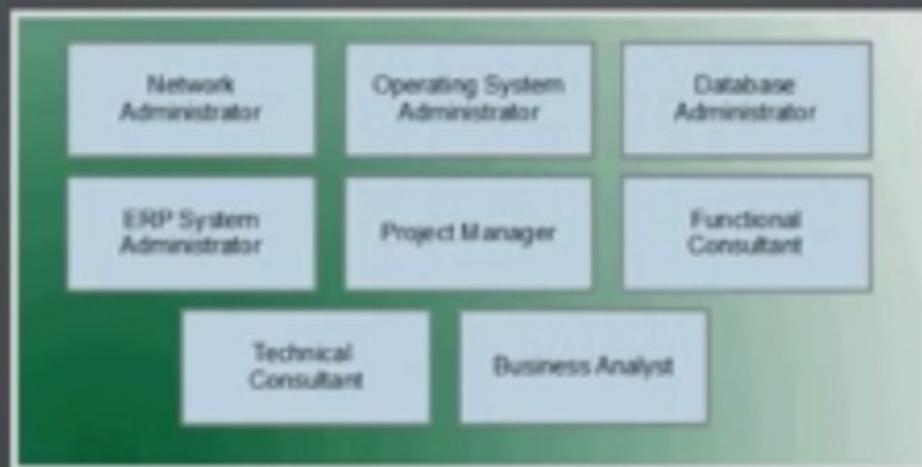
The software is configured to honor company's current business processes. However the company also alters processes where necessary, to bring them align to industry best practices.

Companies do take the implementation of ERP as an opportunity to streamline their business process.



ROLES IN A PROJECT

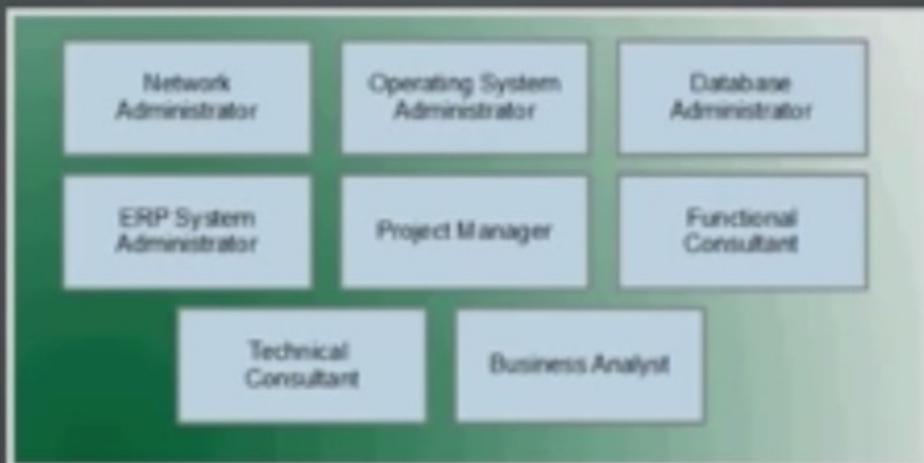
Various roles play part in a typical ERP project. A role may be filled by one or more people depending on the needs. Sometimes one person may be given more than one roles.



ROLES EXAMPLES

Here are important roles:

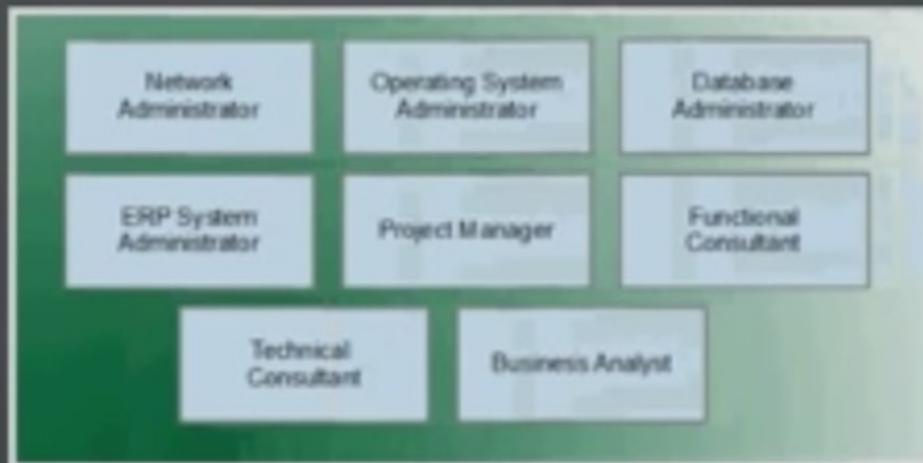
- Network Administrator (Usually 1)
- Operating System Administrator (Usually 1)
- Database Administrator (Usually a team)
- ERP System Administrator (Usually 1)



ROLES EXAMPLES

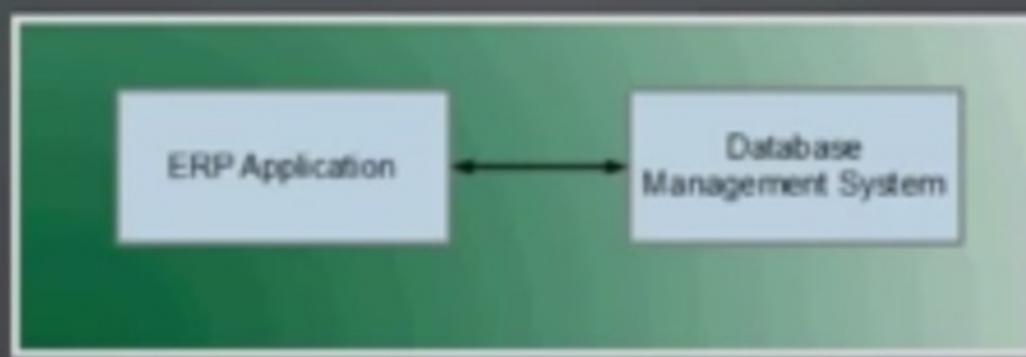
... Continued

- Project Manager (Usually 1)
- Functional Consultant (Usually a team)
- Technical Consultant (Usually a team)
- Business Analyst (Usually 1 per functional area)



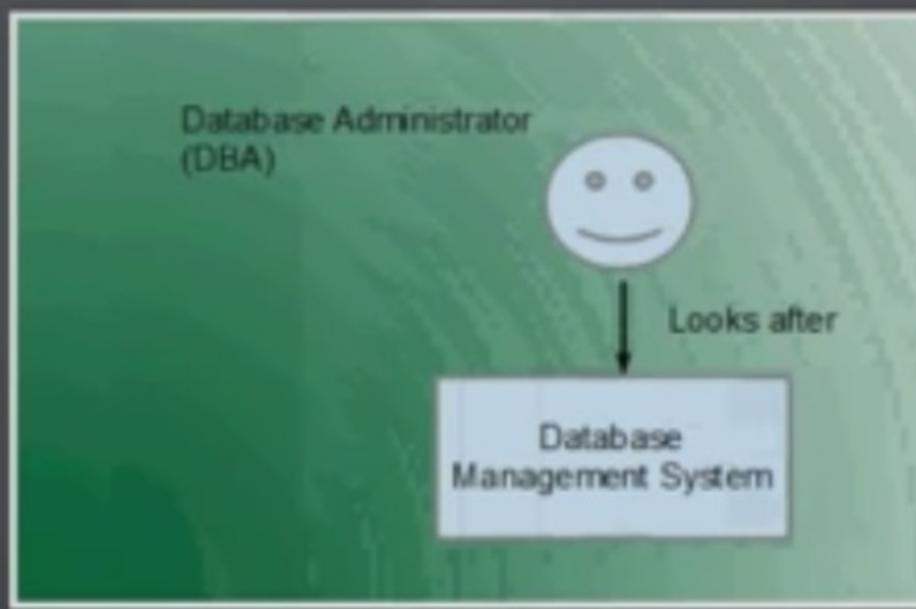
DATABASE MANAGEMENT SYSTEM

ERP software connects to a database software at the back end.
The data is managed in the database.



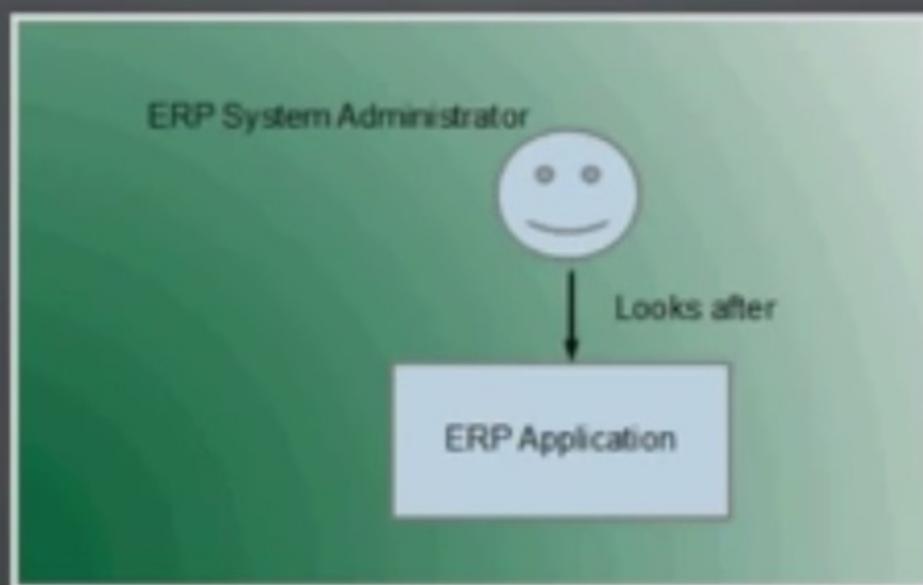
DATABASE ADMINISTRATOR

Database Administrator also known as DBA is the person who looks after the health of the database. He also performs installation of ERP software.



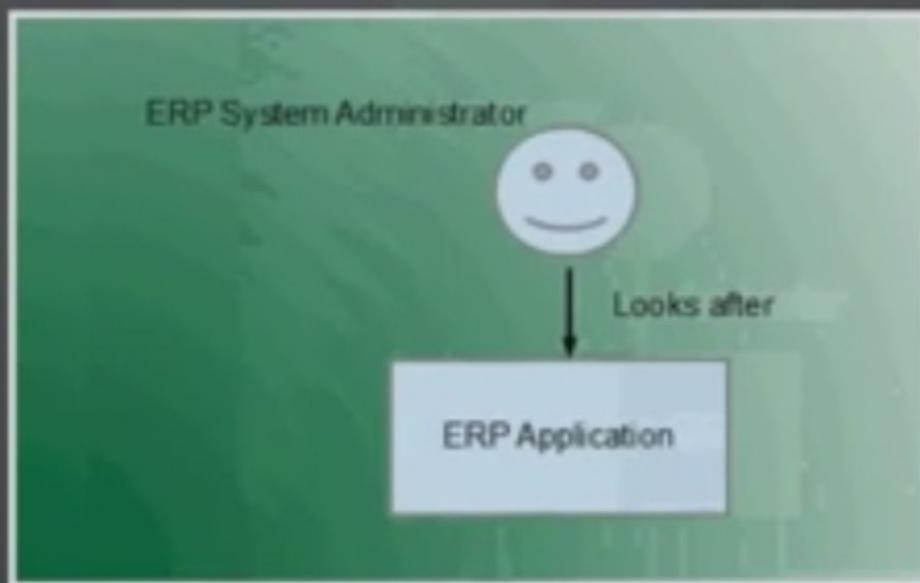
ERP SYSTEM ADMINISTRATOR

ERP System Administrator is the person who looks after the health of the ERP software.



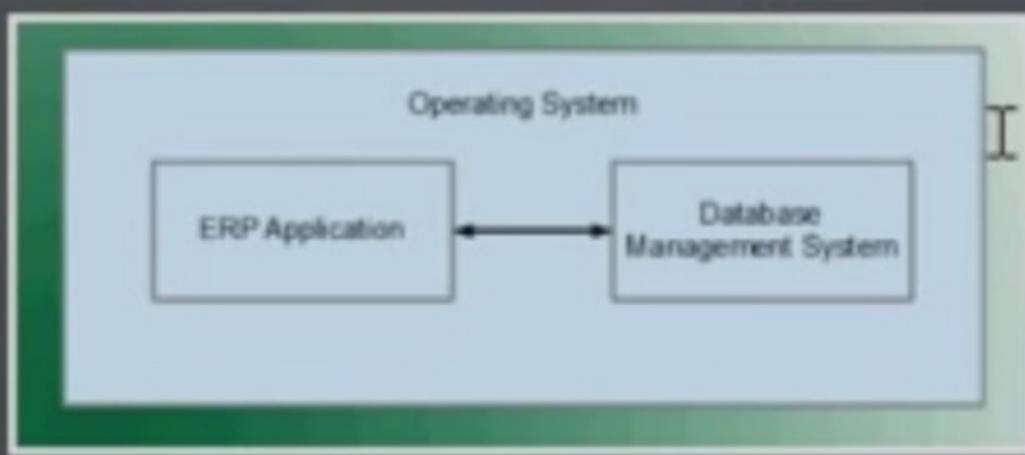
ERP SYSTEM ADMINISTRATOR

ERP System Administrator is the person who looks after the health of the ERP software.



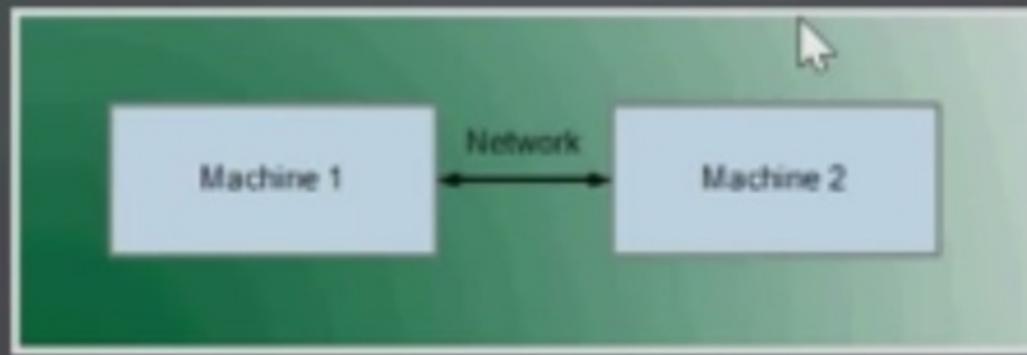
OPERATING SYSTEM

Both database and ERP software runs on an operating system like Linux, Unix, or Windows.



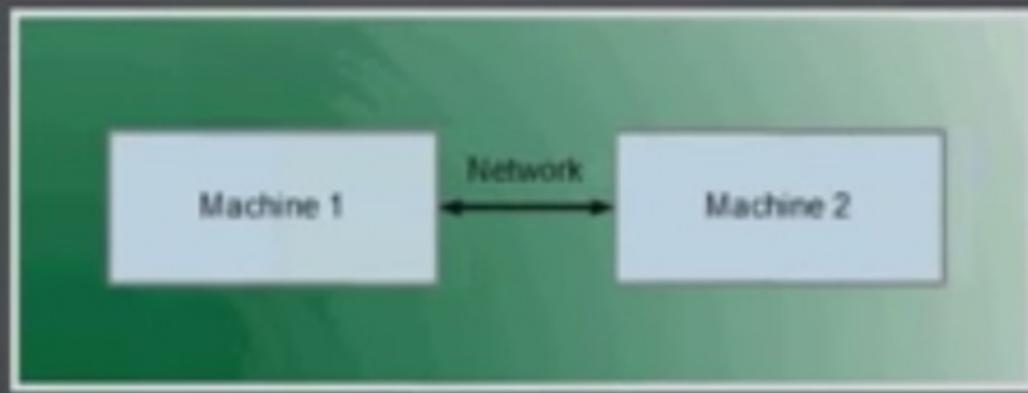
OPERATING SYSTEM ADMINISTRATOR

Operating System Administrator is the person who looks after the health of the operating system.



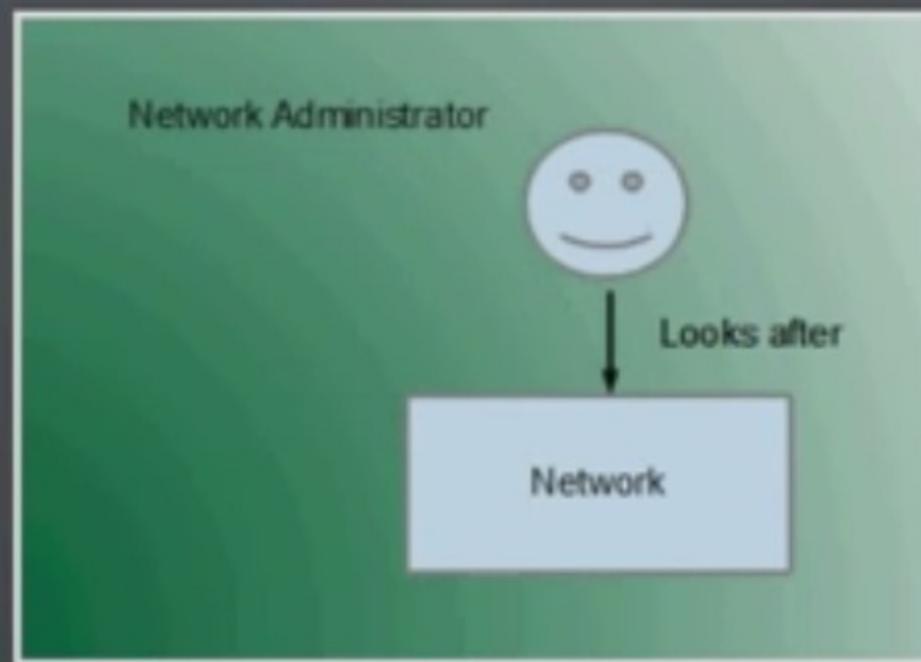
NETWORK

Network connects all the machines together in a system.



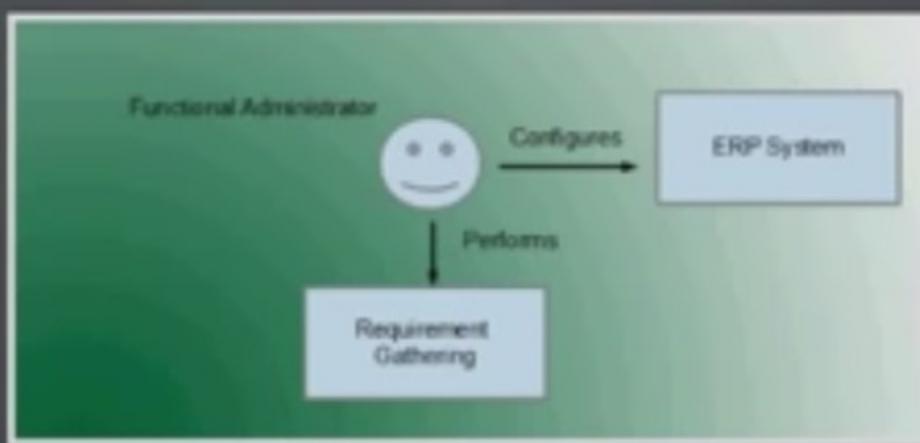
NETWORK ADMINISTRATOR

Network Administrator is the person who looks after the health of the network connecting all the computers together.



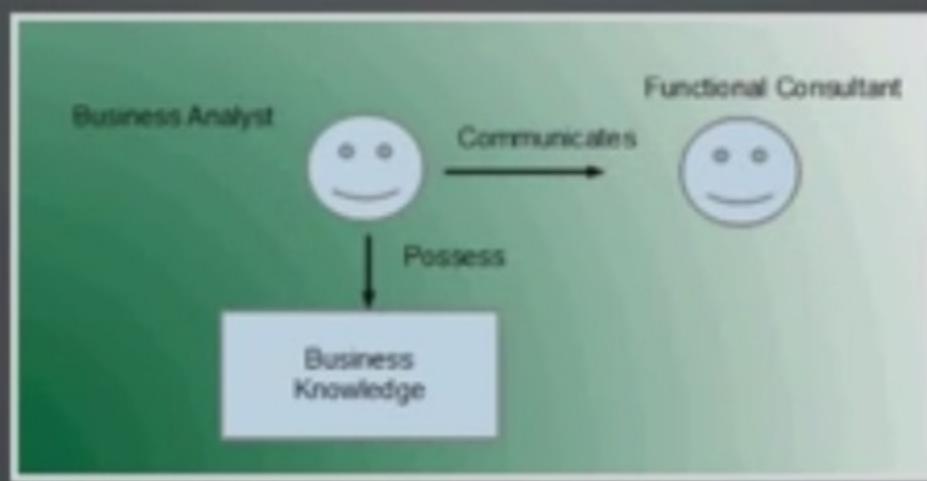
FUNCTIONAL CONSULTANT

Functional Consultant gathers the business requirements and performs ERP setup accordingly.



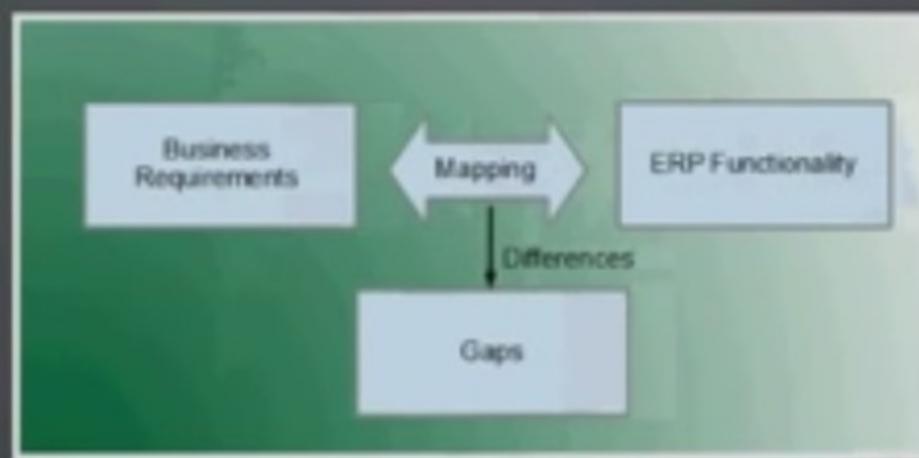
BUSINESS ANALYST

Business Analyst is the person who is an expert of business knowledge. He is in touch with the business users and verifies that the requirements clear to functional consultants.



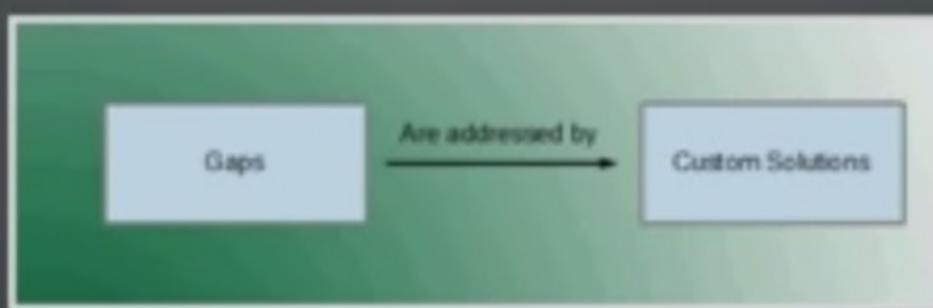
FUNCTIONAL GAPS

In almost all the cases some business requirements are so unique that the ERP system has no built-in functionality to handle those unique cases. The term "Gap" is used for such business requirements.



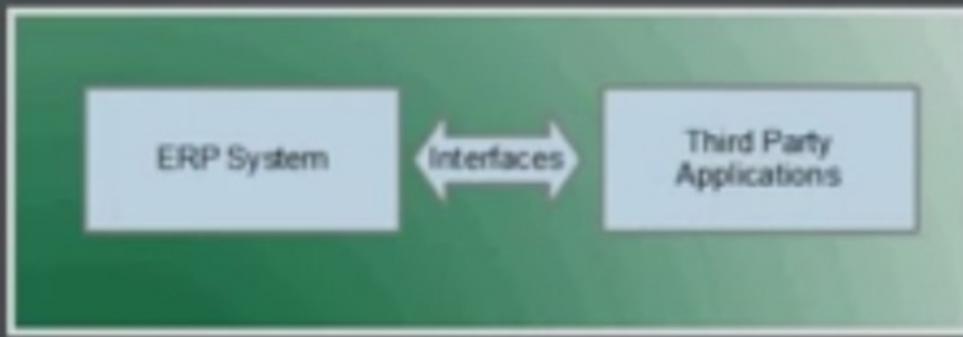
CUSTOMIZATIONS

This is where technical consultants come in. They modify the software by going under the hood and add the missing functionality e.g. new reports are created in the system that were needed by the business. This step is called customization or extension.



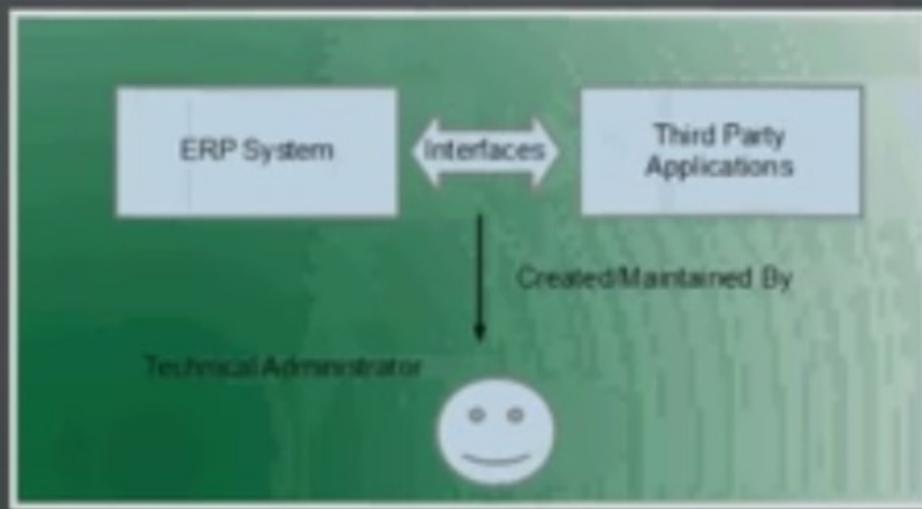
COMMUNICATION AMONG APPLICATIONS

In most cases ERP system talks to other third party system running within the same company or in an external company e.g. suppliers and customers.



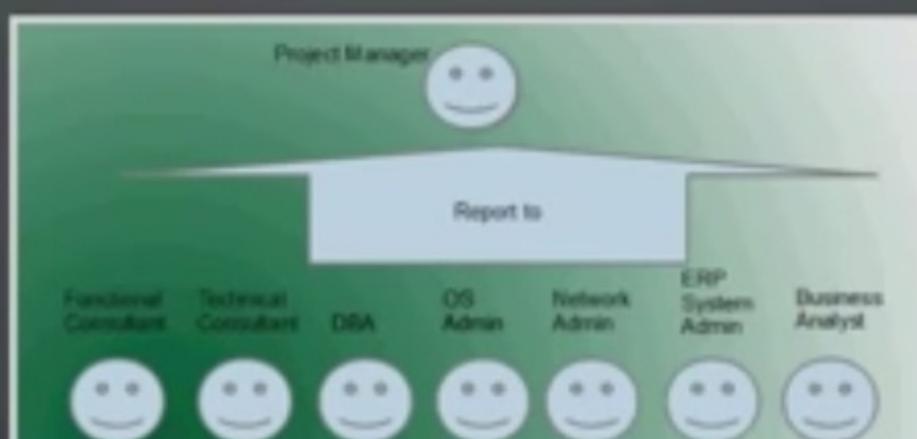
INTERFACES

Technical consultants assist in writing programs that help communicate information back and forth between the ERP software and the third party software either within the company or outside it. Programs that aid communication between two software is called "interface".



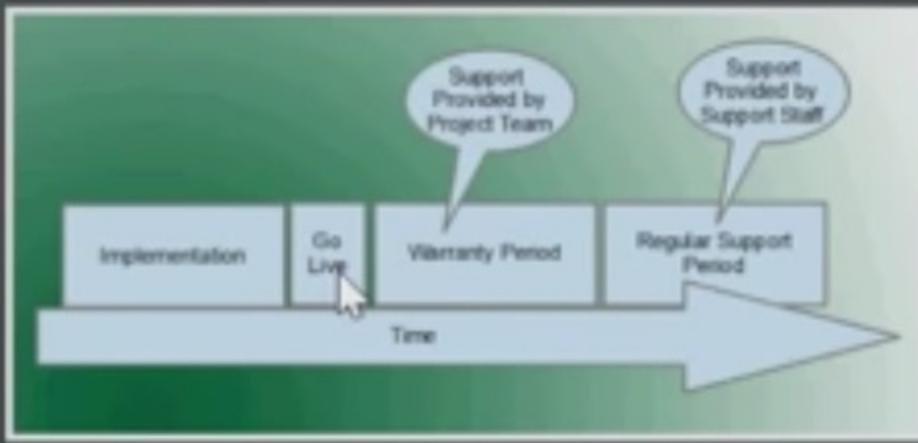
PROJECT MANAGER

Project manager is the person who is managing the project. All other team members report to the project manager during the project. Most members also report to their regular bosses as well. For example, a database administrators will be reporting to the manager or director of the IT department as well as to the project manager during the life of the project.



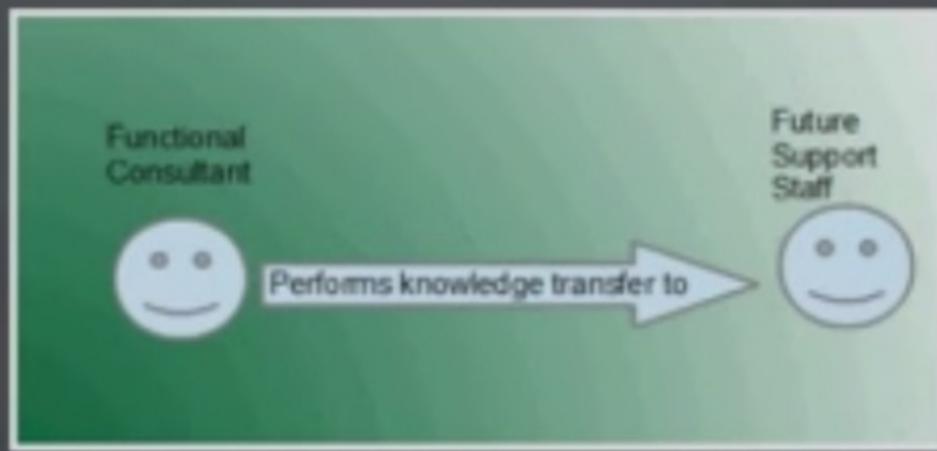
WARRANTY AND SUPPORT PERIODS

After the go-live, warranty period begins. Any problems that come up will be handled by the implementation team. After that the support role will be transferred to another team, usually permanent staffs or a third party company specialized in support.



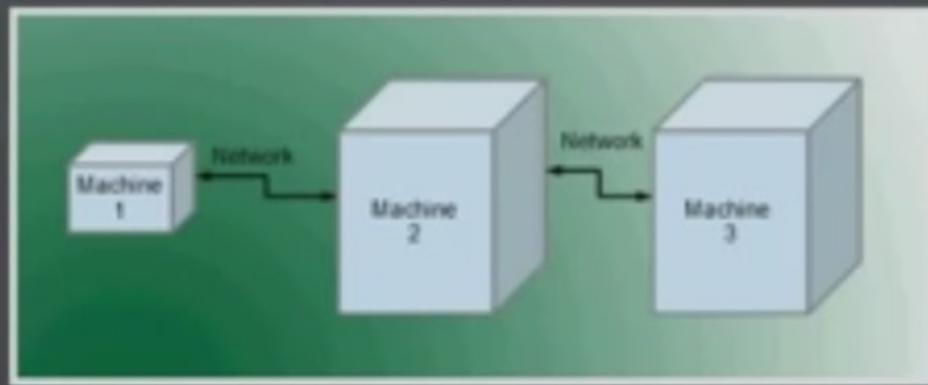
TRAINING

Consultants would provide training to the new person or team who will be providing support.



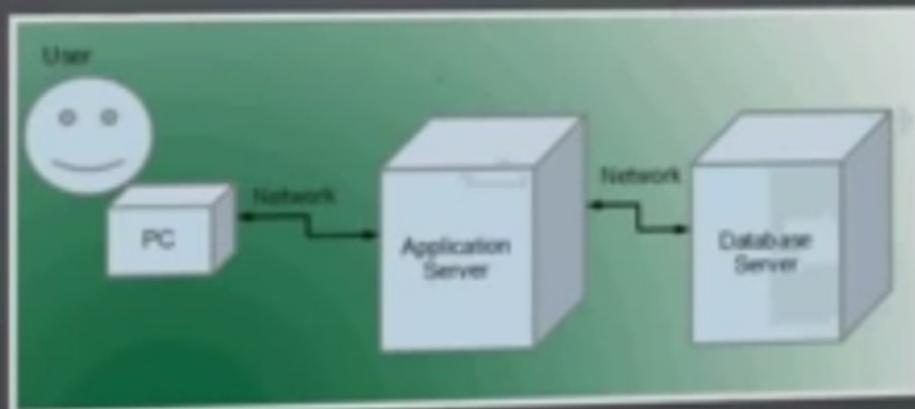
MULTIPLE SERVERS

Usually there are few computers, called servers, that are connected and work together to produce workable environment for ERP systems.



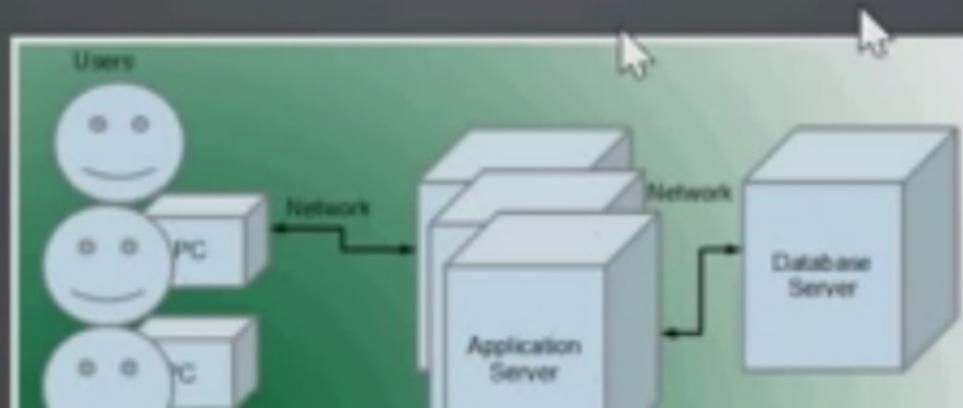
THREE TIER ENVIRONMENT

One computer hosts the application (the actual ERP software) and another one hosts the database. Users connect to the computer running ERP System through their browsers. This is called a three tier environment.



LOAD BALANCING

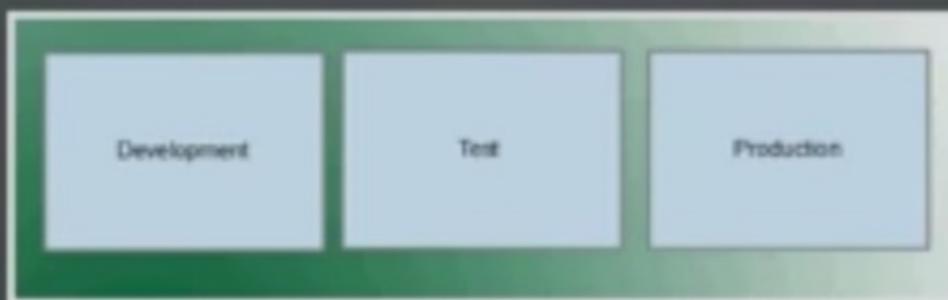
In an environment where numerous users connect, there could be even multiple application servers installed, communicating with the same database server. This way the load is shared among various machines. The term used for this setup is "Load Balancing".



ENVIRONMENTS

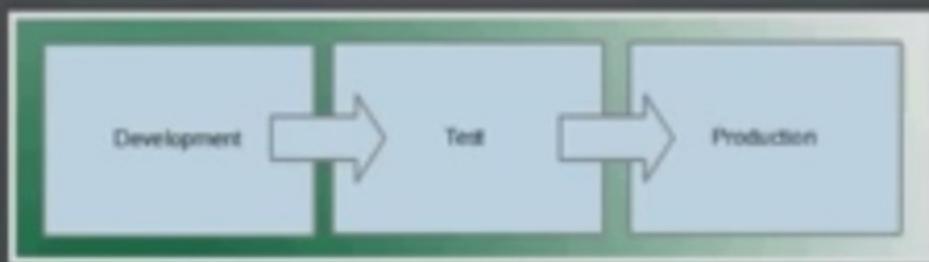
Companies running ERP systems usually keep three sets of environments:

- Development
- Test
- Production



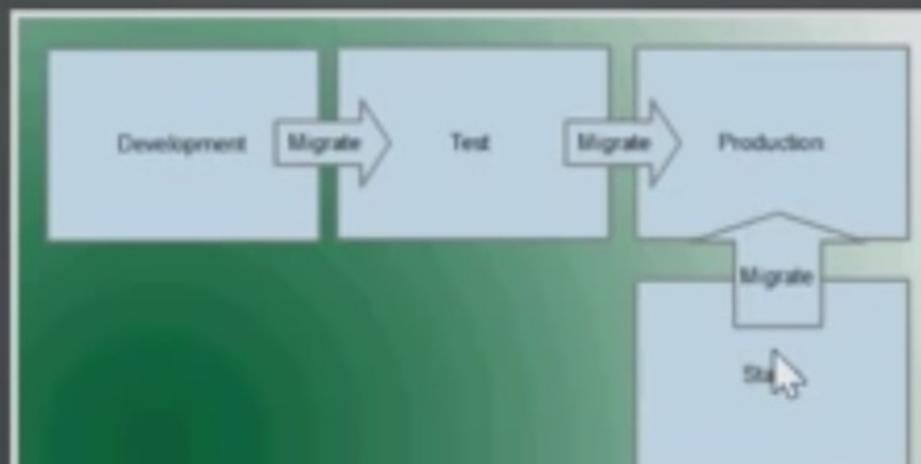
TRANSPORTING SETUP

The configuration is performed in the development environment first by the team. The setup is copied over to the Test environment where users perform their testing. Once users are happy the setup is copied to the production (Go-live), where the system is actually used to manage day to day operations of the company. The term used for copying configuration from one environment to the other is "Transport".



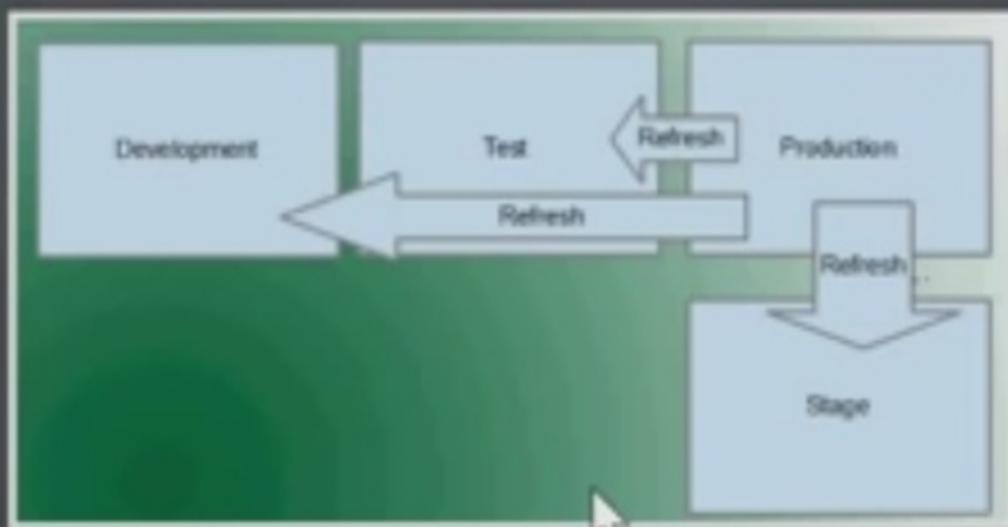
STAGE ENVIRONMENT

In some companies few other environments are also maintained
e.g. "Stage" where troubleshooting is performed, and bug
fixes/patches are applied and tested first before moving them to
production.



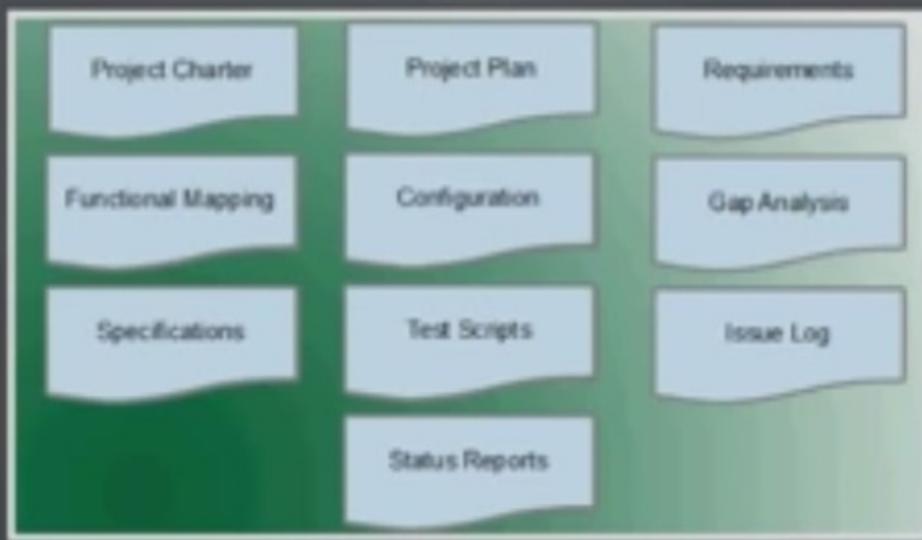
REFRESH

All environments are periodically refreshed with recent production data.



PROJECT DOCUMENTATIONS

Documents are created through the project and are stored in the central place for the project.



EXAMPLES OF DOCUMENTS

Some examples are:

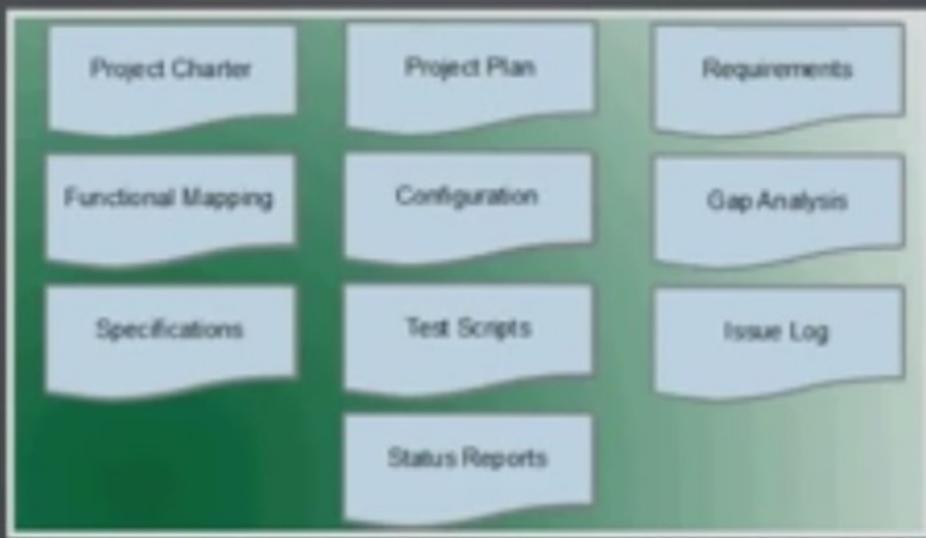
- Project Charter by Project Manager
- Project Plan by Project Manager
- Requirement Gathering document by Functional Consultant



EXAMPLES OF DOCUMENTS

... Continued

- Functional Mapping by Functional Consultant
- Configuration document by Functional Consultant
- Gap Analysis document by Functional Consultant



EXAMPLES OF DOCUMENTS

... Continued

- Functional/Technical Specifications by Technical Consultant
- Test Scripts by Functional Consultant/Business Analyst
- Issue log by Project Manager
- Period Status Reports by all members



PROJECT CHARTER

Document: Project Charter by Project Manager

Contents: What is involved in the project from a high level perspective, which business needs are behind this project, what benefits it will bring, who is sponsoring the project, what are the risks, what are dependencies and assumptions etc.

Project Charter

PROJECT PLAN

Document: Project Plan by Project Manager

Contents: Tasks that will take place during the project.

Project Plan

REQUIREMENT GATHERING

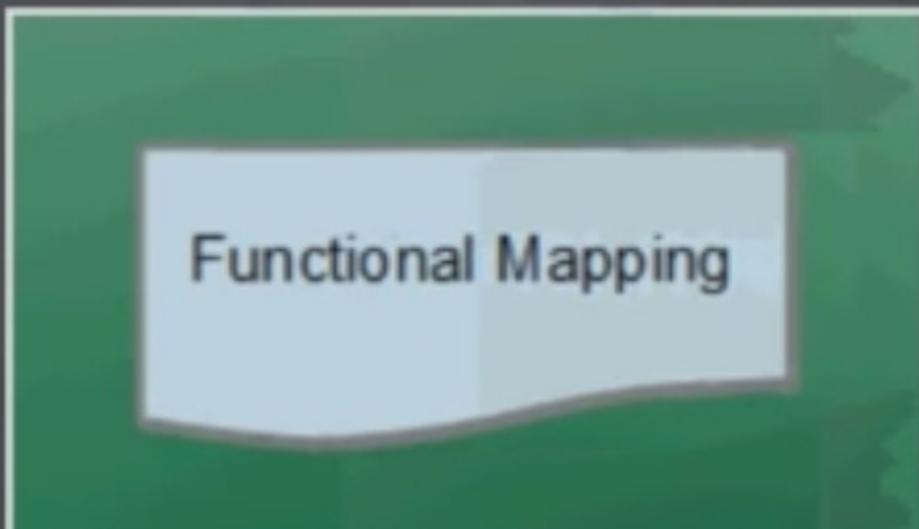
Document: Requirement Gathering document by Functional Consultant

Contents: How the new system should be implemented.

Requirements

FUNCTIONAL MAPPING

Document: Functional Mapping by Functional Consultant
Contents: Mapping of requirements to the ERP software features; Which features should be enabled in ERP.

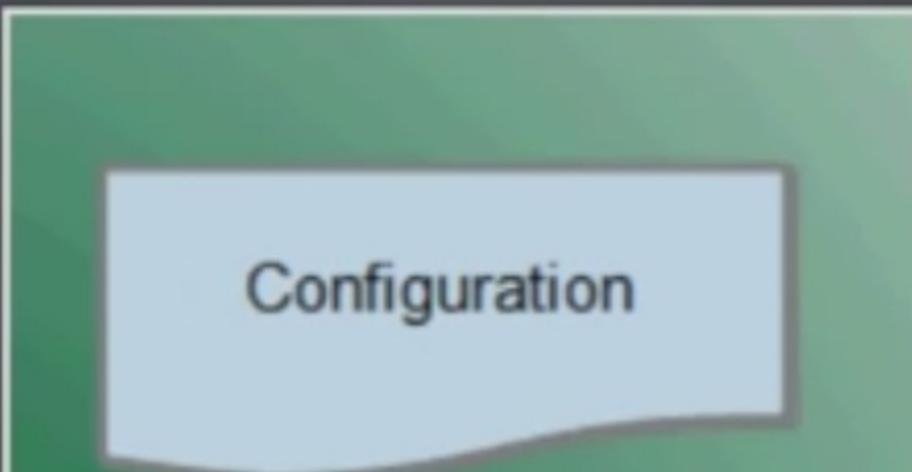


Functional Mapping

SYSTEM CONFIGURATION

Document: Configuration document by Functional Consultant

Contents: How the new system will be configured so that the required features are available.



Configuration

GAP ANALYSIS

Document: Gap Analysis document by Functional Consultant
Contents: Which requirements are not matched to any of the available features in ERP.

Gap Analysis

SPECIFICATIONS

Document: Functional/Technnical Specifications by Technical Consultant

Contents: How custom developed features will work.

Specifications

TEST SCRIPTS

Document: Test Scripts by Functional Consultant/Business Analyst

Contents: What steps users will perform during testing. Test Results are also captured by the Business in the Test Scripts document.

Test Scripts

ISSUE LOG

Document: Issue log by Project Manager

Contents: What problems came up and how they were handled

Issue Log

STATUS REPORTS

Document: Period Status Reports by All

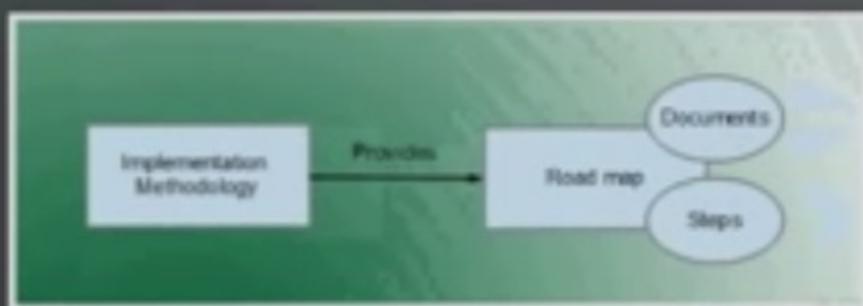
Contents: What was completed last week and what is due next week

Status Reports

IMPLEMENTATION METHODOLOGY

The project teams usually follows an implementation methodology which provides a roadmap through the project. A methodology dictates:

- Which documents will be created at which point
- How the documents will look like (Templates are provided)
- What will be the sequence of configuration tasks.



STATUS REPORTS

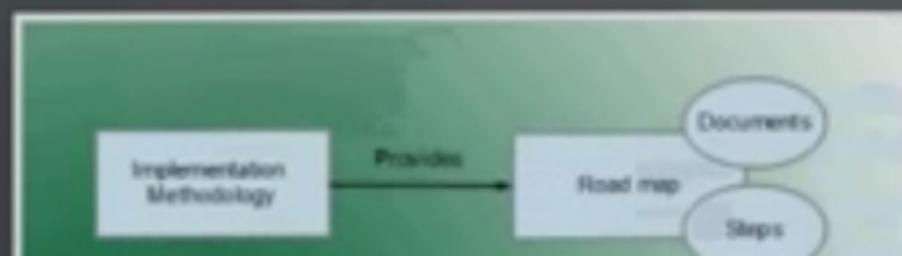
Document: Period Status Reports by All
Contents: What was completed last week and what is due next week

Status Reports

IMPLEMENTATION METHODOLOGY

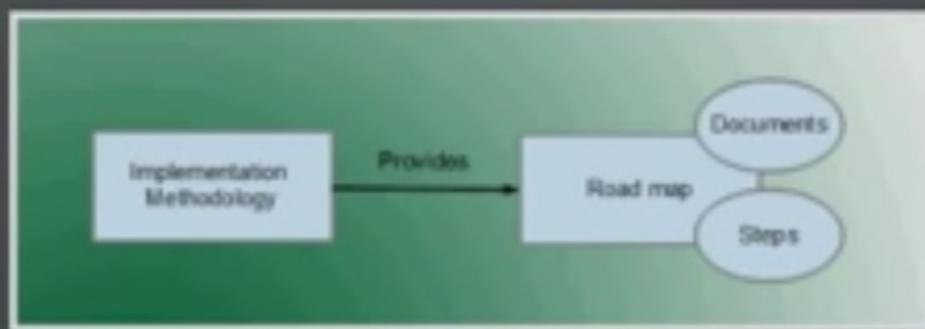
The project teams usually follows an implementation methodology which provides a roadmap through the project. A methodology dictates:

- Which documents will be created at which point
- How the documents will look like (Templates are provided)
- What will be the sequence of configuration tasks.



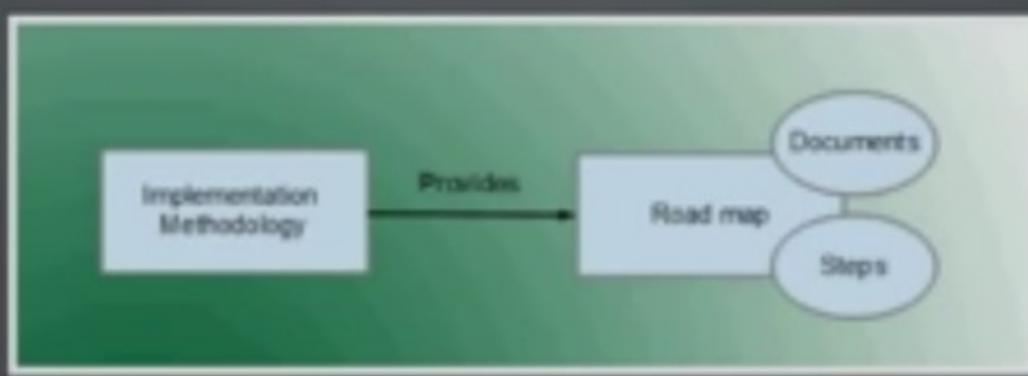
IMPLEMENTATION METHODOLOGY AND PROJECT PLAN

The steps are incorporated in desired sequence, as dictated by the methodology, in the overall project plan managed by project manager.



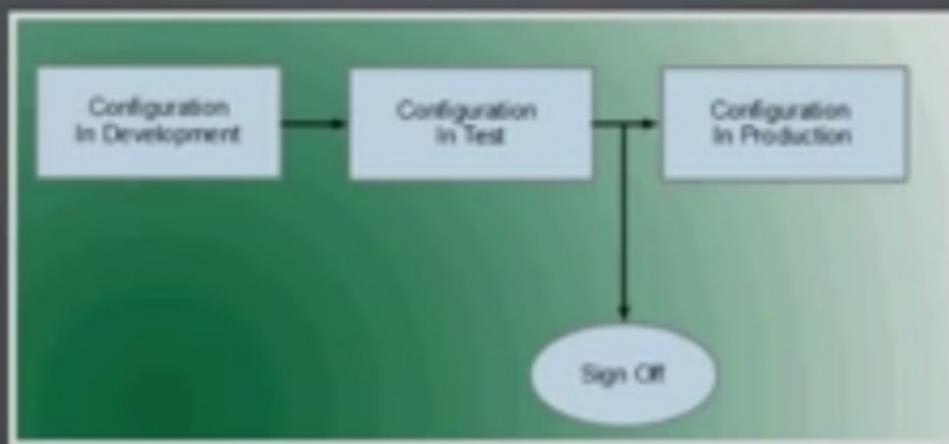
PROPRIETARY METHODOLOGIES

Large and reputed consultings firms usually have their own proprietary methodology that they follow.



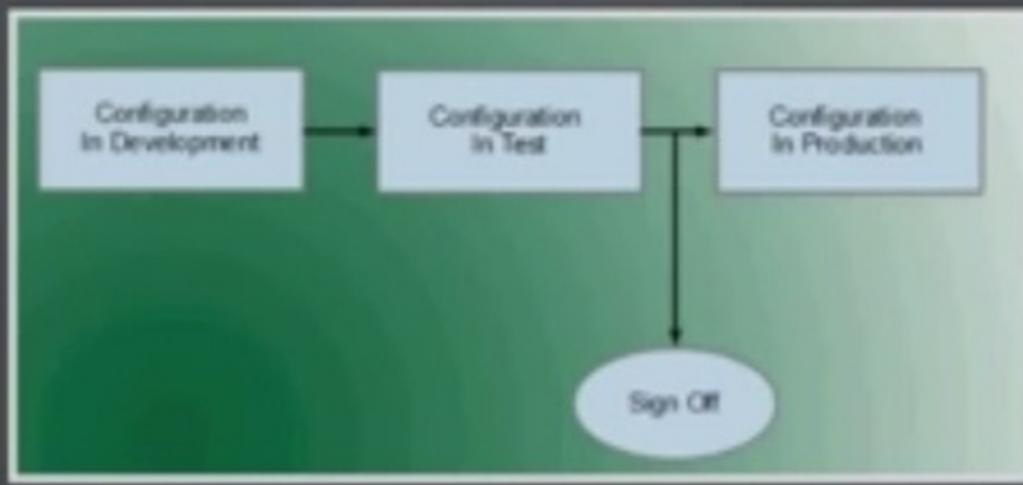
SIGN-OFF

Written sign-offs are required from the business at various stages e.g. before moving configuration from Test to Production.



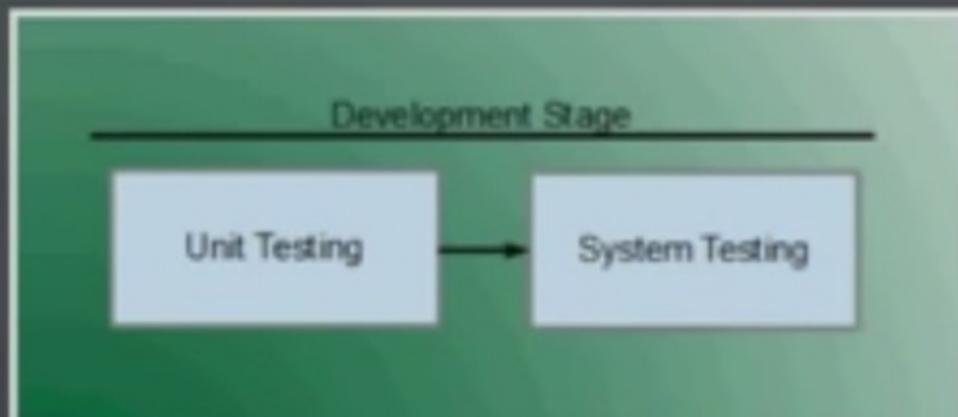
SIGN-OFF

Written sign-offs are required from the business at various stages e.g. before moving configuration from Test to Production



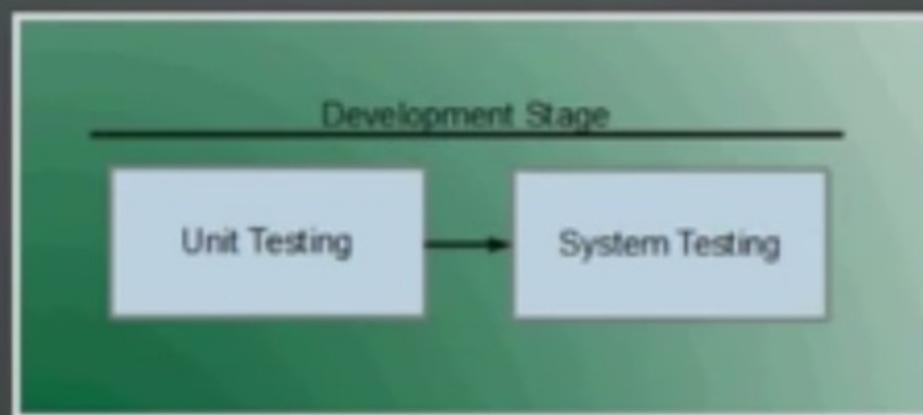
UNIT TESTING

Consultants perform unit testing and system testing during development stage. Unit testing refers to testing of one module (or unit) individually. The focus is on the functionality of the modules.



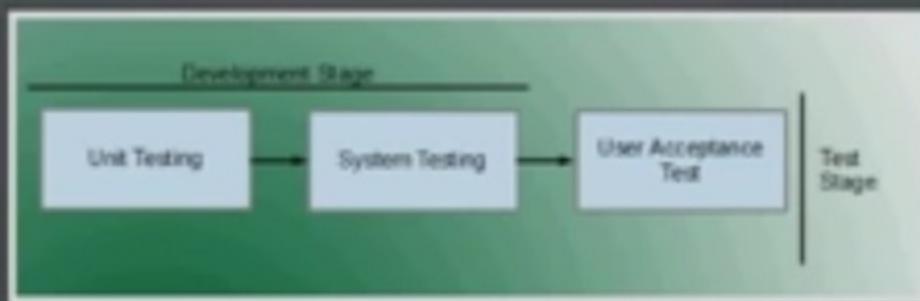
SYSTEM TESTING

System testing refers to the testing of the whole system (all modules together). The focus is on the integrity of the system as a whole e.g. if modules are communicating with each other properly. The testings are performed on development environment, however a separate test environment could be requested as well for this purpose.



USER ACCEPTANCE TESTING

User Acceptance Testing (UAT) is a mandatory testing. Here users drive the system. The consultants train and guide the users. The focus is to verify whatever was promised, is delivered or not. At the end of the testing users must provide a sign-off. All outstanding problems need to be fixed. The testing takes place in Test environment dedicated for this purpose.



Three Different Types of Enterprise Systems

Small businesses implement enterprise systems to gain company-wide access to business knowledge, increase employee productivity and minimize the duplication of company data. Enterprise systems may also enable a business to reduce the cost of information technology and minimize the manual input of data. These enterprise system attributes offer particular benefits, such as the support of teamwork, an improved response to the marketplace, increased work quality and greater employee collaboration and efficiency.

Customer Relationship Management

Customer relationship management systems were developed to address the need to raise a sales department's productivity and make the management of a company's customers an effective way to increase sales. With CRM functions, such as sales opportunity management, a company learns more about its customers' needs and buying behavior and combines this information with market information to enhance the quality of the company's marketing plans and sales forecasts. Other attributes of the CRM system, including the integration of this system with other systems and system access via mobile devices, allow employees to update and compare data regardless of the system it's in and to access information from any client site or other location. Equally important, CRM supports mass e-mail communications and automates the sales process workflow to improve employee productivity.

Supply Chain Management

A supply chain refers to the collection of people, tasks, equipment, data and other resources required to produce and move products from a vendor to a customer. Dr. Robert Hanfield of Bank of America describes supply chain management as the management of supply chain activities by the supply chain firms in an effective and efficient way. According to Hanfield, such activities include product development, material sourcing, production and logistics as well as the information systems that coordinate these activities. Information flows allow supply chain partners to coordinate their strategic and operational plans as well as the day-to-day flow of goods and materials through the supply chain. The physical flows include the manufacture, transport and storage of goods or materials.

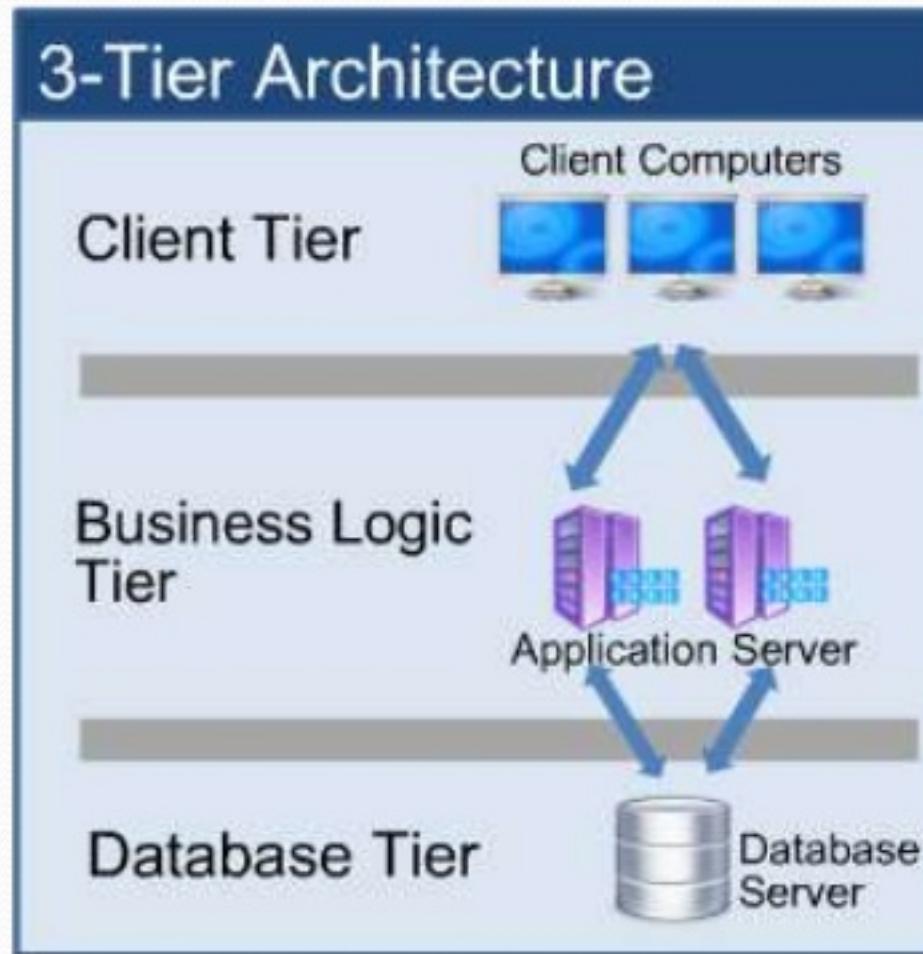
Enterprise Resource Planning

The enterprise resource planning system integrates software applications just as a company integrates business processes, such as purchasing, finance, human resources and inventory management. Within an ERP system, the integrated software modules, such as sales, quality management and accounts receivable, communicate and share data. Each of these modules consists of multiple applications that perform the functions required to execute particular end-to-end business processes. For example, the sales module includes the applications necessary to create and manage sales contracts, sales orders, sales invoices and sales order pricing. ERP applications support not only various operational and administrative tasks, such as the creation of an account payable or a time sheet, they may also be customized to support a number of different industries, including oil and gas, retail and banking.

Three-tier Client/Server implementation Architecture

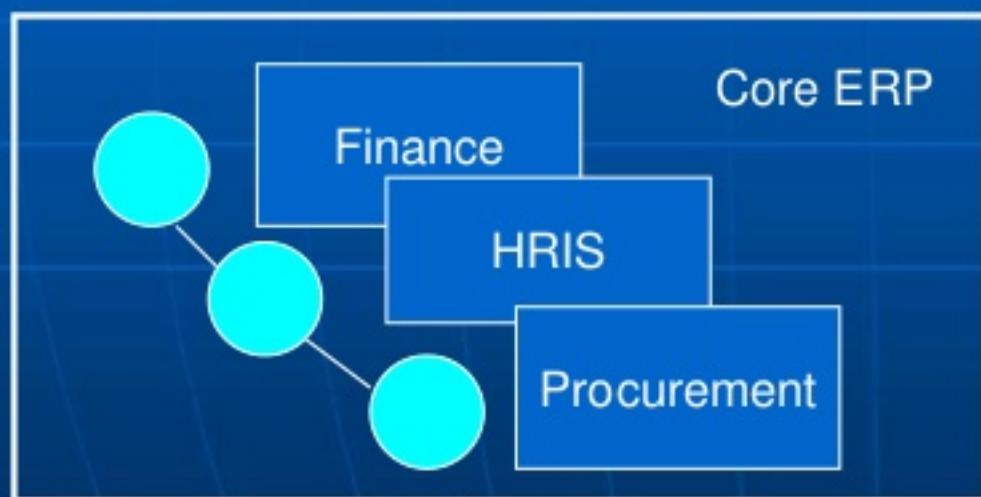
- the database and application functions are separated.
- This is very typical of large production ERP deployments.
- satisfying client requests requires two or more network connections.
- the client establishes communications with the application server which then creates a second connection to the database server.

Three-tier Client/Server Implementation Architecture



Integrated System ERP Program

Integrated System

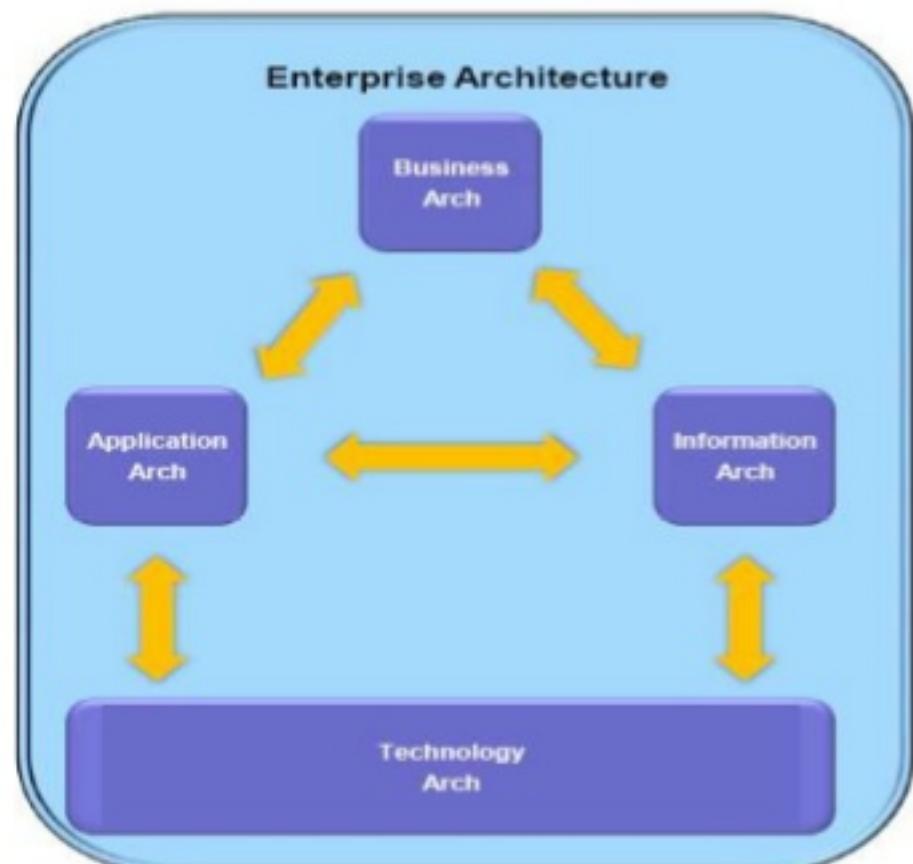


Enterprise Application Interfaces



Authoritative
Data Source

Traditional View & Transitional View



Design Patterns

Alan Goude

Useful resources

- **Books**

Design patterns : elements of reusable object-oriented software, by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.

Design Patterns: AND Applying UML and Patterns, an Introduction to Object-Oriented Analysis and Design and Iterative Development: Elements of Reusable Object-oriented Software by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

Design Patterns Explained: A New Perspective on Object-Oriented Design (Software Patterns) by Alan Shalloway , James Trott

Modern C++ Design: Applied Generic and Design Patterns (C++ in Depth Series) (Paperback) by Andrei Alexandrescu , Scott Meyers , John Vlissides.

- **Web links**

- <http://www.odesign.com/>
- <http://www.dofactory.com/Patterns/Patterns.aspx>
- http://www.developer.com/design/article.php/10925_1502691_1
- [http://en.wikipedia.org/wiki/Design_pattern_\(computer_science\)](http://en.wikipedia.org/wiki/Design_pattern_(computer_science))

The Originator of Patterns

- "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice." -- Christopher Alexander 1977

What are Patterns

- Current use comes from the work of the architect Christopher Alexander
- Alexander studied ways to improve the process of designing buildings and urban areas
- “Each pattern is a three-part rule, which expresses a relation between a certain context, a problem and a solution.”
- Hence, the common definition of a pattern: “A solution to a problem in a context.”
- Patterns can be applied to many different areas of human endeavour, including software development

Patterns in Software

- "Designing object-oriented software is hard and designing reusable object-oriented software is even harder." - Erich Gamma
- Experienced designers reuse solutions that have worked in the past
- Well-structured object-oriented systems have recurring patterns of classes and objects
- Knowledge of the patterns that have worked in the past allows a designer to be more productive and the resulting designs to be more flexible and reusable

The Gang of Four

- The authors of the [DesignPatternsBook](#) (1994) came to be known as the "Gang of Four." The name of the book ("Design Patterns: Elements of Reusable Object-Oriented Software") is too long for e-mail, so "book by the gang of four" became a shorthand name for it. After all, it isn't the ONLY book on patterns. That got shortened to "GOF book", which is pretty cryptic the first time you hear it.
- The GOF are :- Gamma, Erich; Richard Helm, Ralph Johnson, and John Vlissides

Design Patterns

- *The Design Patterns* book described 23 patterns that were based on the experience of the authors at that time. These patterns were selected because they represented solutions to common problems in software development. Many more patterns have been documented and catalogued since the publishing of *Design Patterns*. However, these 23 are probably the best known and certainly the most popular.

Design Patterns

- Design patterns describe the relations and interactions of different class or objects or types.
- They do not specify the final class or types that will be used in any software code, but give an abstract view of the solution.
- Patterns show us how to build systems with good object oriented design qualities by reusing successful designs and architectures.
- Expressing proven techniques speed up the development process and make the design patterns, more accessible to developers of new system.

Classification of Design Patterns

- Design patterns were originally classified into three types
 - Creational patterns
 - Structural patterns
 - Behavioural patterns.
- A fourth has since been added
 - Concurrency patterns

Creational Patterns

- Creational design patterns are design patterns that deal with object creation mechanisms, trying to create objects in a manner suitable to the situation.
- The basic form of object creation could result in design problems or added complexity to the design.
- Creational design patterns solve this problem by somehow controlling this object creation.

Structural Patterns

- **structural design patterns** are design patterns that ease the design by identifying a simple way to realise relationships between entities.
- These describe how objects and classes combine themselves to form a large structure

Behavioural Patterns

- Design patterns that identify common communication patterns between objects and realize these patterns.
- These patterns increase flexibility in carrying out this communication.

Structure of a Design Pattern

- Design pattern documentation is highly structured.
- The patterns are documented from a template that identifies the information needed to understand the software problem and the solution in terms of the relationships between the classes and objects necessary to implement the solution.
- There is no uniform agreement within the design pattern community on how to describe a pattern template.

Pattern Documentation

- **Pattern Name and Classification:** A descriptive and unique name that helps in identifying and referring to the pattern.
- **Intent:** A description of the goal behind the pattern and the reason for using it.
- **Also Known As:** Other names for the pattern.
- **Motivation (Forces):** A scenario consisting of a problem and a context in which this pattern can be used.
- **Applicability:** Situations in which this pattern is usable; the context for the pattern.
- **Structure:** A graphical representation of the pattern. [Class diagrams](#) and [Interaction diagrams](#) may be used for this purpose.
- **Participants:** A listing of the classes and objects used in the pattern and their roles in the design.

Pattern Documentation - cont.

- **Collaboration:** A description of how classes and objects used in the pattern interact with each other.
- **Consequences:** A description of the results, side effects, and trade offs caused by using the pattern.
- **Implementation:** A description of an implementation of the pattern; the solution part of the pattern.
- **Sample Code:** An illustration of how the pattern can be used in a programming language
- **Known Uses:** Examples of real usages of the pattern.
- **Related Patterns:** Other patterns that have some relationship with the pattern; discussion of the differences between the pattern and similar patterns.

Creational Patterns

- Abstract Factory - Creates an instance of several families of classes
- Builder - Separates object construction from its representation
- Factory Method - Creates an instance of several derived classes
- Prototype - A fully initialized instance to be copied or cloned
- Singleton - A class of which only a single instance can exist

Structural Patterns

- Adapter - Match interfaces of different classes
- Bridge - Separates an object's interface from its implementation
- Composite - A tree structure of simple and composite objects
- Decorator - Add responsibilities to objects dynamically
- Facade - A single class that represents an entire subsystem
- Flyweight - A fine-grained instance used for efficient sharing
- Proxy - An object representing another object

Behavioral Patterns

- Chain of Resp. - A way of passing a request between a chain of objects
- Command - Encapsulate a command request as an object Interpreter A way to include language elements in a program
- Iterator - Sequentially access the elements of a collection
- Mediator - Defines simplified communication between classes
- Memento - Capture and restore an object's internal state
- Observer - A way of notifying change to a number of classes
- State - Alter an object's behavior when its state changes
- Strategy - Encapsulates an algorithm inside a class
- Template Method - Defer the exact steps of an algorithm to a subclass
- Visitor - Defines a new operation to a class without change

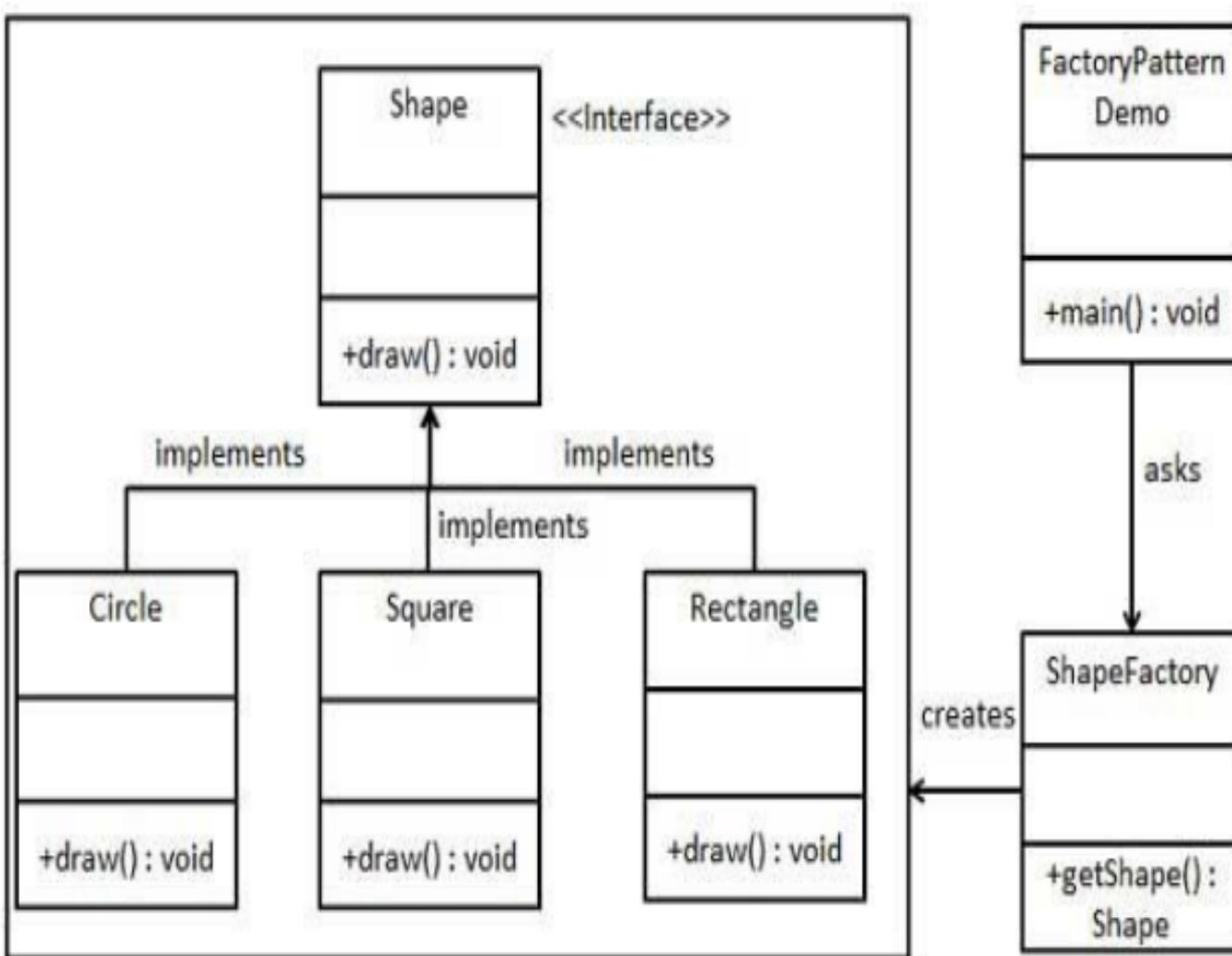
Factory Pattern

- This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.
- In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.

Implementation

- We're going to create a ***Shape interface*** and concrete classes implementing the Shape interface. A factory class ***ShapeFactory*** is defined as a next step.
- ***FactoryPatternDemo***, our demo class will use ***ShapeFactory*** to get a Shape object.
- It will pass information (CIRCLE / RECTANGLE / SQUARE) to ShapeFactory to get the type of object it needs.

Class Diagram



Shape.java

- Step 1
 - Create an interface(Shape.java)
 - Public interface Shape{
void draw();
};

Rectangle.java

- Step 2
 - Create concrete classes implementing the same interface
 - //Rectangle.java

```
public class Rectangle implements Shape
{
    @Override
    public void draw()
    {
        System.out.println("Inside Rectangle::draw()
method.");
    }
};
```

Circle.java

```
public class Circle implements Shape {  
    @Override  
    public void draw()  
    {  
        System.out.println("draw method in circle class");  
    }  
};
```

Square.java

```
public class Square implements Shape{
    @Override
    public void draw()
    {
        System.out.println("Draw method
in Square class");
    }
};
```

ShapeFactory.java

```
public class ShapeFactory {  
  
    //use getShape method to get object of type shape  
    public Shape getShape(String shapeType){  
        if(shapeType == null){  
            return null;  
        }  
        if(shapeType.equalsIgnoreCase("CIRCLE")){  
            return new Circle();  
        } else if(shapeType.equalsIgnoreCase("RECTANGLE")){  
            return new Rectangle();  
        } else if(shapeType.equalsIgnoreCase("SQUARE")){  
            return new Square();  
        }  
        return null;  
    }  
}
```

FactoryPatternDemo.java

```
public class FactoryPatternDemo {  
  
    public static void main(String[] args) {  
        ShapeFactory shapeFactory = new ShapeFactory();  
  
        //get an object of Circle and call its draw method.  
        Shape shape1 = shapeFactory.getShape("CIRCLE");  
  
        //call draw method of Circle  
        shape1.draw();  
  
        //get an object of Rectangle and call its draw method.  
        Shape shape2 = shapeFactory.getShape("RECTANGLE");  
  
        //call draw method of Rectangle  
        shape2.draw();  
  
        //get an object of Square and call its draw method.  
        Shape shape3 = shapeFactory.getShape("SQUARE");  
  
        //call draw method of circle  
        shape3.draw();  
    }  
}
```

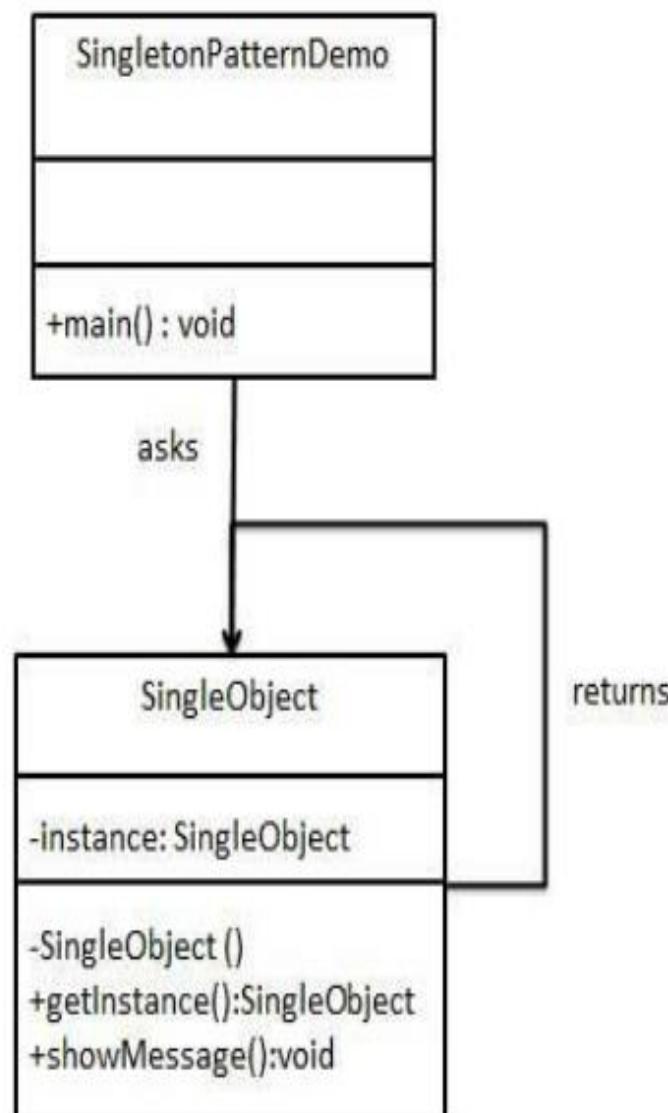
Singleton Design Pattern

- Singleton pattern is one of the simplest design patterns in Java.
- This type of design pattern comes under creational pattern as this pattern provides one of the best way to create an object.
- This pattern involves a single class which is responsible to creates own object while making sure that only single object get created.
- This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.

Implementation

- We're going to create a ***SingleObject*** class.
SingleObject class have its constructor as private and have a static instance of itself.
- ***SingleObject*** class provides a static method to get its static instance to outside world.
- ***SingletonPatternDemo***, our demo class will use SingleObject class to get a SingleObject object.

Class Diagram



Step 1:

Create a Singleton Class.

SingleObject.java

```
public class SingleObject {  
  
    //create an object of SingleObject  
    private static SingleObject instance = new SingleObject();  
  
    //make the constructor private so that this class cannot be  
    //instantiated  
    private SingleObject() {}  
  
    //Get the only object available  
    public static SingleObject getInstance(){  
        return instance;  
    }  
  
    public void showMessage(){  
        System.out.println("Hello World!");  
    }  
}
```

Step 2

Get the only object from the singleton class.

SingletonPatternDemo.java

```
public class SingletonPatternDemo {  
    public static void main(String[] args) {  
  
        //illegal construct  
        //Compile Time Error: The constructor SingleObject() is not  
visible  
        //SingleObject object = new SingleObject();  
  
        //Get the only object available  
        SingleObject object = SingleObject.getInstance();  
  
        //show the message  
        object.showMessage();  
    }  
}
```

Step 3

Verify the output.

```
Hello World!
```

WEB APP ARCHITECTURES:

MULTI-TIER (2-TIER, 3-TIER)

MODEL-VIEWER-CONTROLLER (MVC)

REST ARCHITECTURAL STYLE

Slides created by Manos Papagelis

Based on materials by Marty Stepp, M. Ernst, S. Reges, D. Notkin, R. Mercer, R. Boswell, Wikipedia

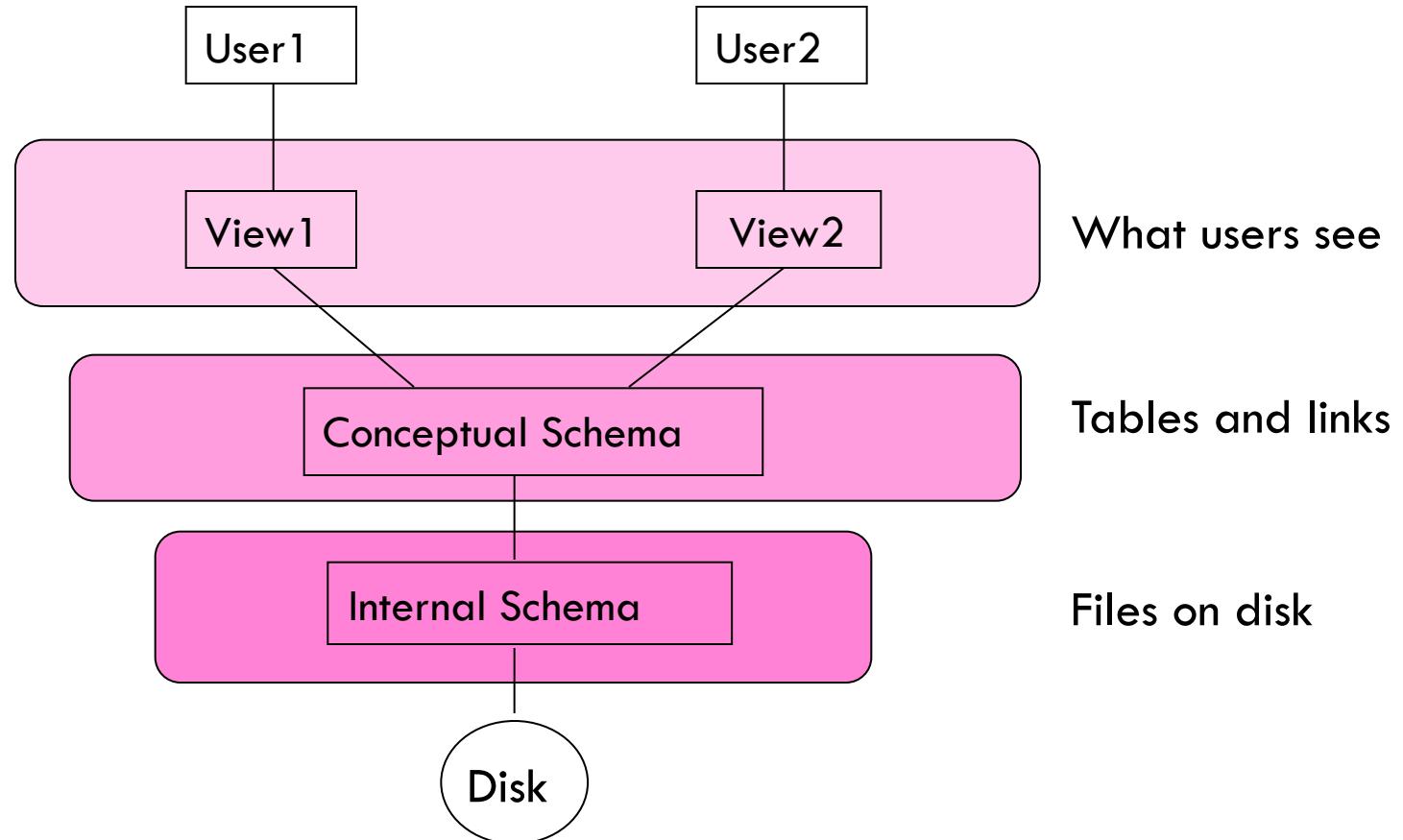
Overview

- Data Independence in Relational Databases
- N-tier Architectures
- Design Patterns
 - The MVC Design Pattern
- REST Architectural Style

Data Independence in Rel. DBMS

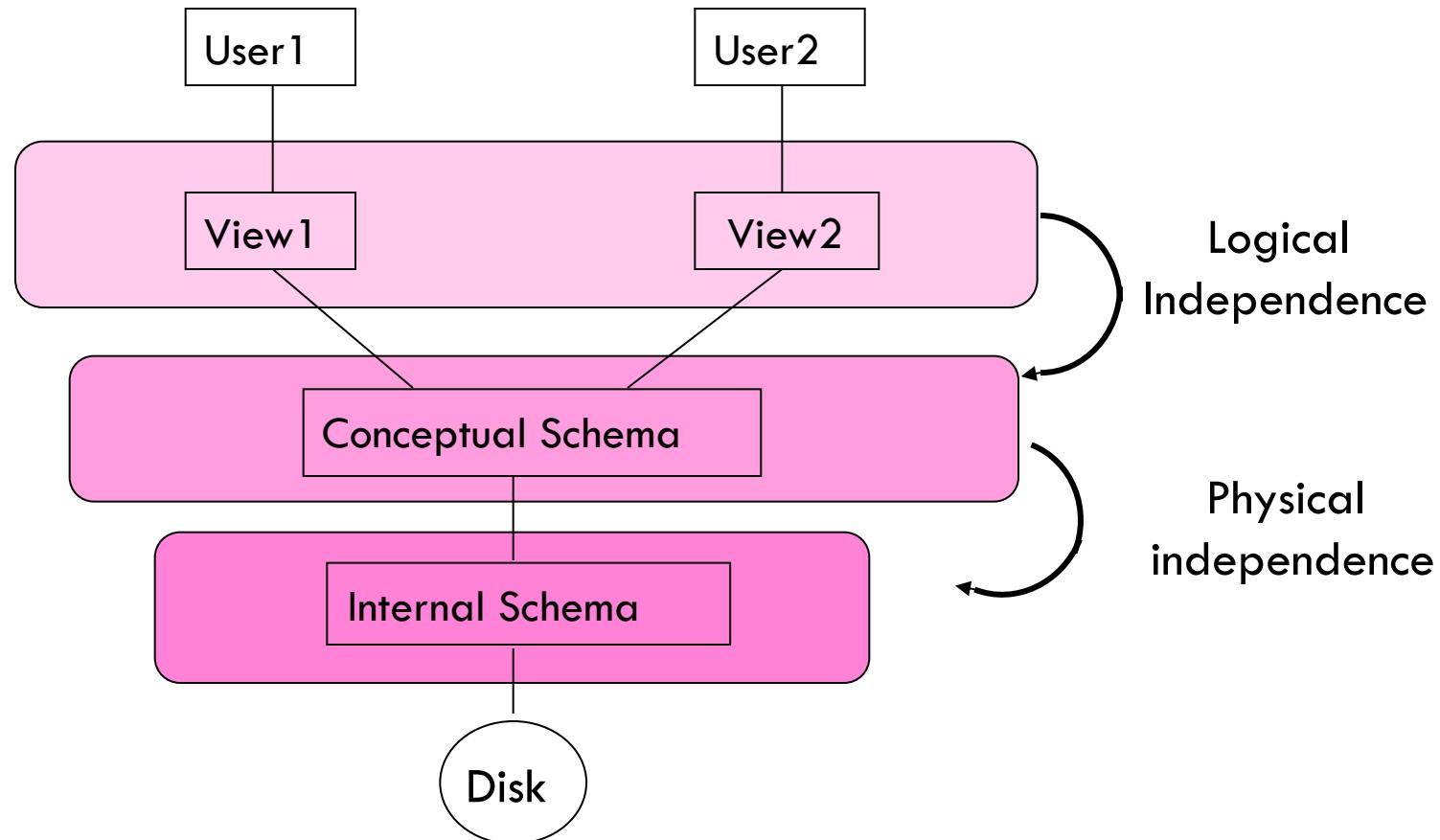
Database Architecture With Views

4



Each level is independent of the levels below

Logical and Physical Independence



Each level is independent of the levels below

Data Independence

- **Logical Independence:** The ability to change the logical schema without changing the external schema or application programs
 - Can add new fields, new tables without changing views
 - Can change structure of tables without changing view
- **Physical Independence:** The ability to change the physical schema without changing the logical schema
 - Storage space can change
 - Type of some data can change for reasons of optimization

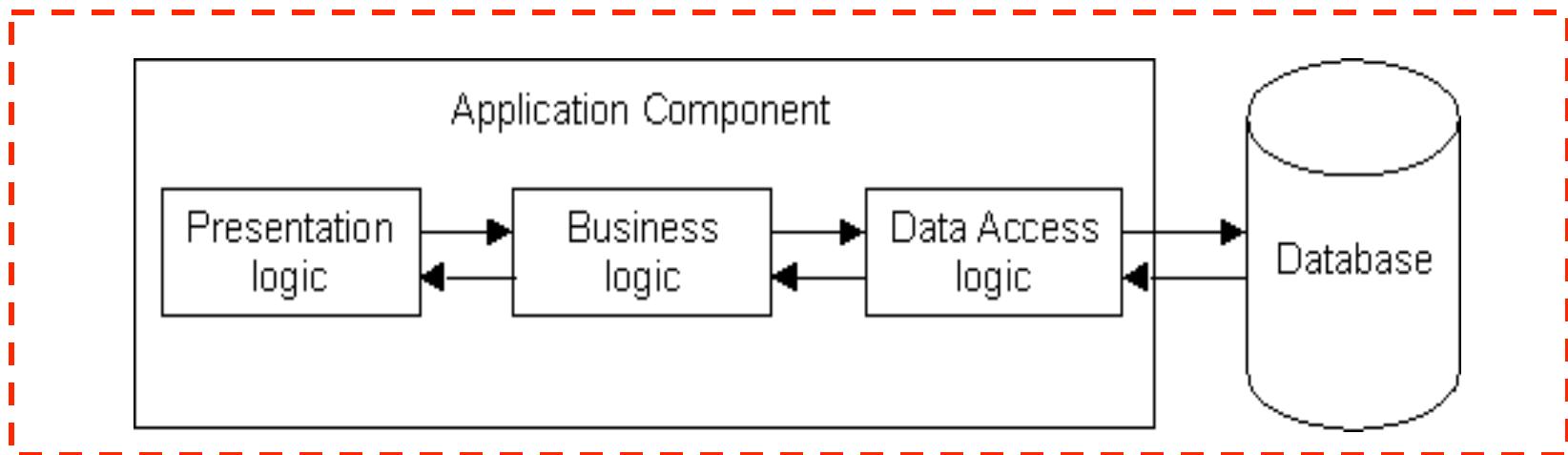
LESSON: Keep the VIEW (what the user sees) independent of the MODEL (domain knowledge)

N-tier architectures

Significance of “Tiers”

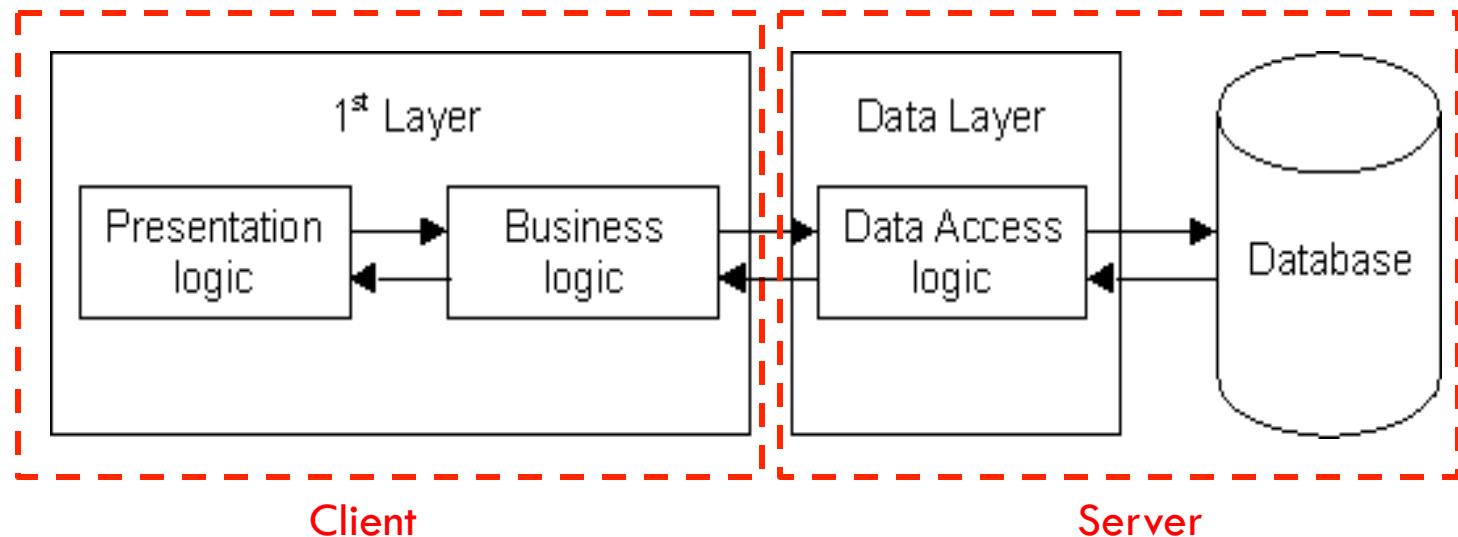
- N-tier architectures have the same components
 - Presentation
 - Business/Logic
 - Data
- N-tier architectures try to separate the components into different tiers/layers
 - Tier: physical separation
 - Layer: logical separation

1-Tier Architecture



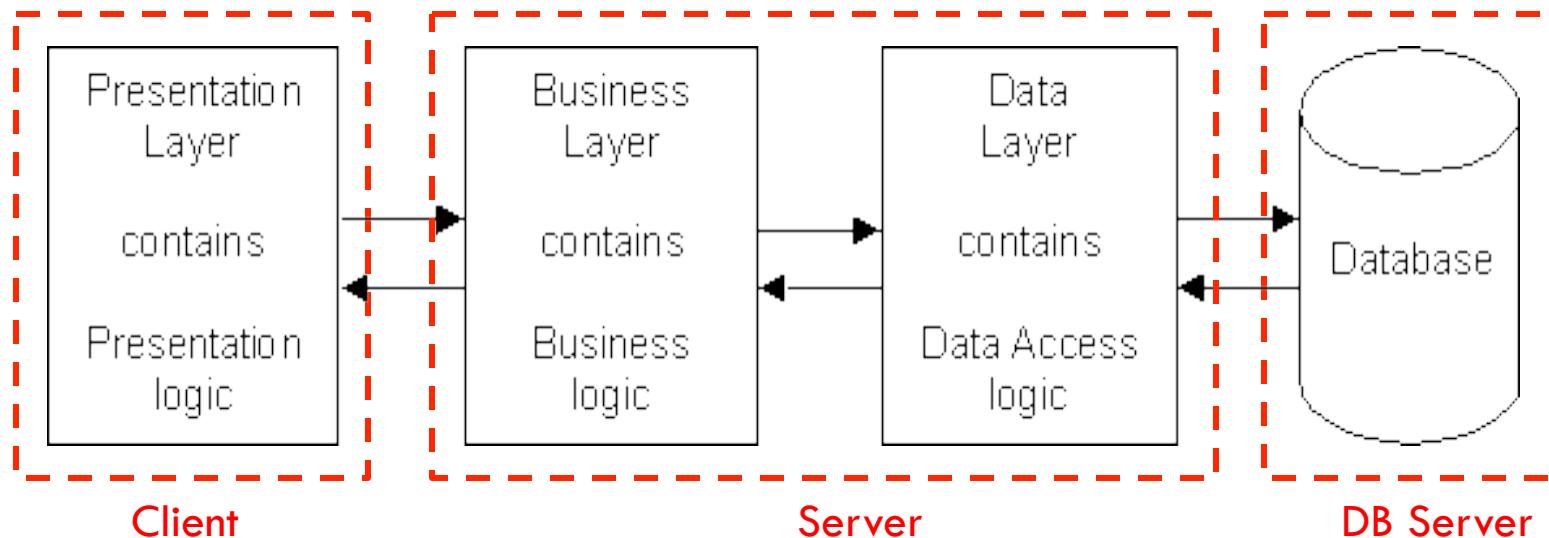
- All 3 layers are on the same machine
 - All code and processing kept on a single machine
- Presentation, Logic, Data layers are tightly connected
 - Scalability: Single processor means hard to increase volume of processing
 - Portability: Moving to a new machine may mean rewriting everything
 - Maintenance: Changing one layer requires changing other layers

2-Tier Architecture



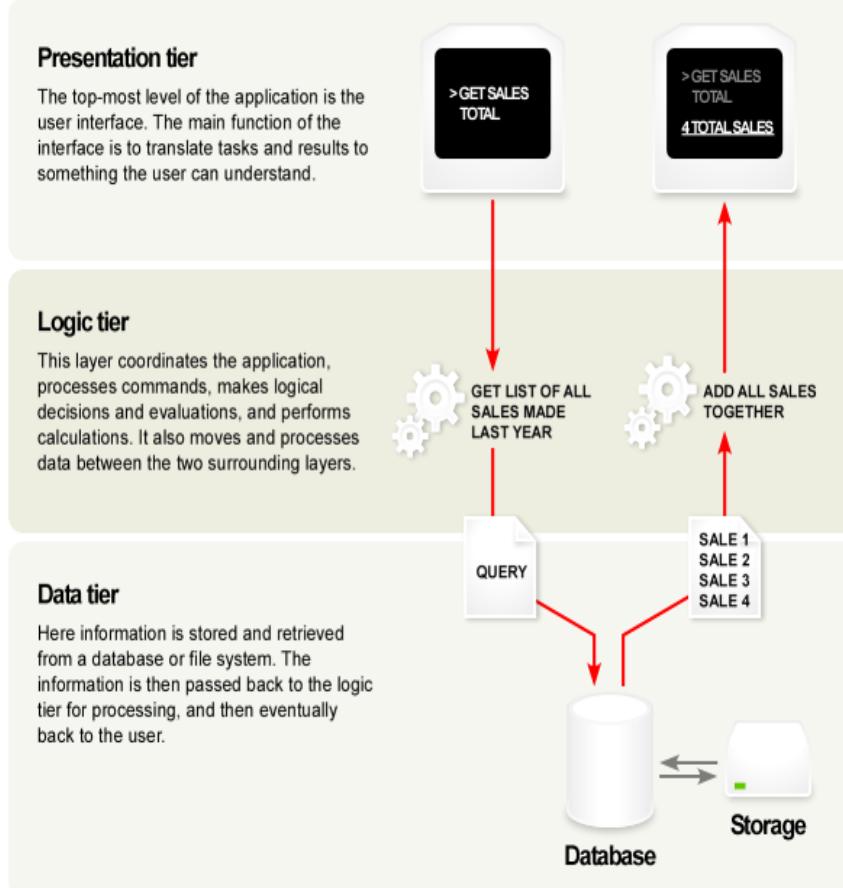
- Database runs on Server
 - Separated from client
 - Easy to switch to a different database
- Presentation and logic layers still tightly connected
 - Heavy load on server
 - Potential congestion on network
 - Presentation still tied to business logic

3-Tier Architecture



- Each layer can potentially run on a different machine
- Presentation, logic, data layers disconnected

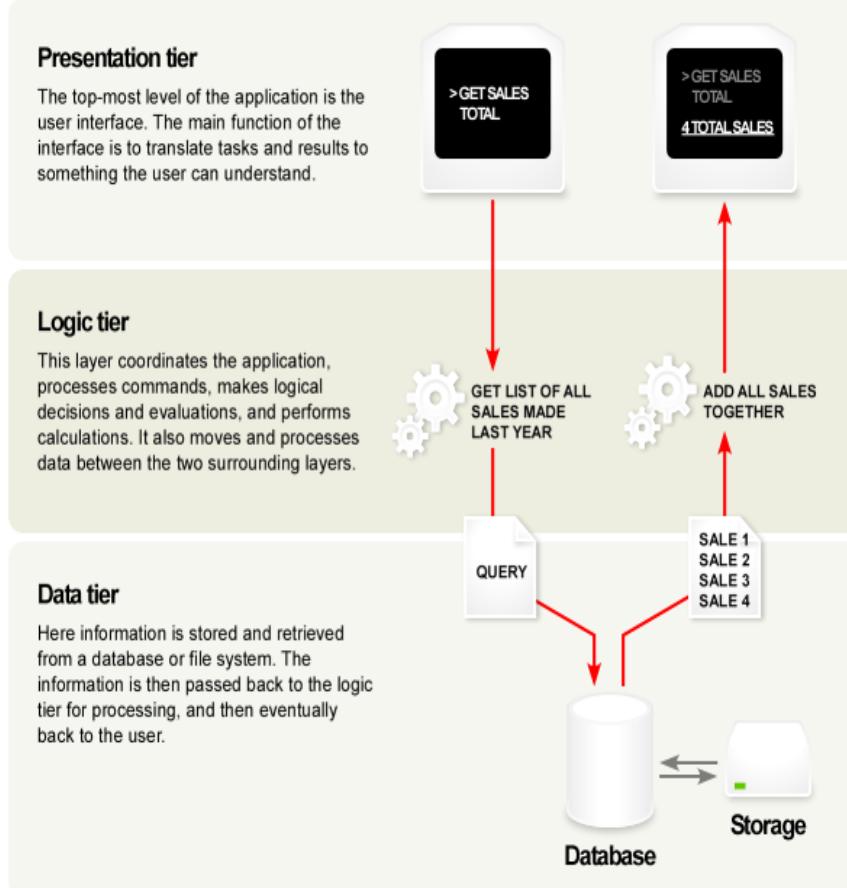
A Typical 3-tier Architecture



Architecture Principles

- Client-server architecture
- Each tier (Presentation, Logic, Data) should be **independent** and should not expose dependencies related to the implementation
- Unconnected tiers should not communicate
- Change in platform affects only the layer running on that particular platform

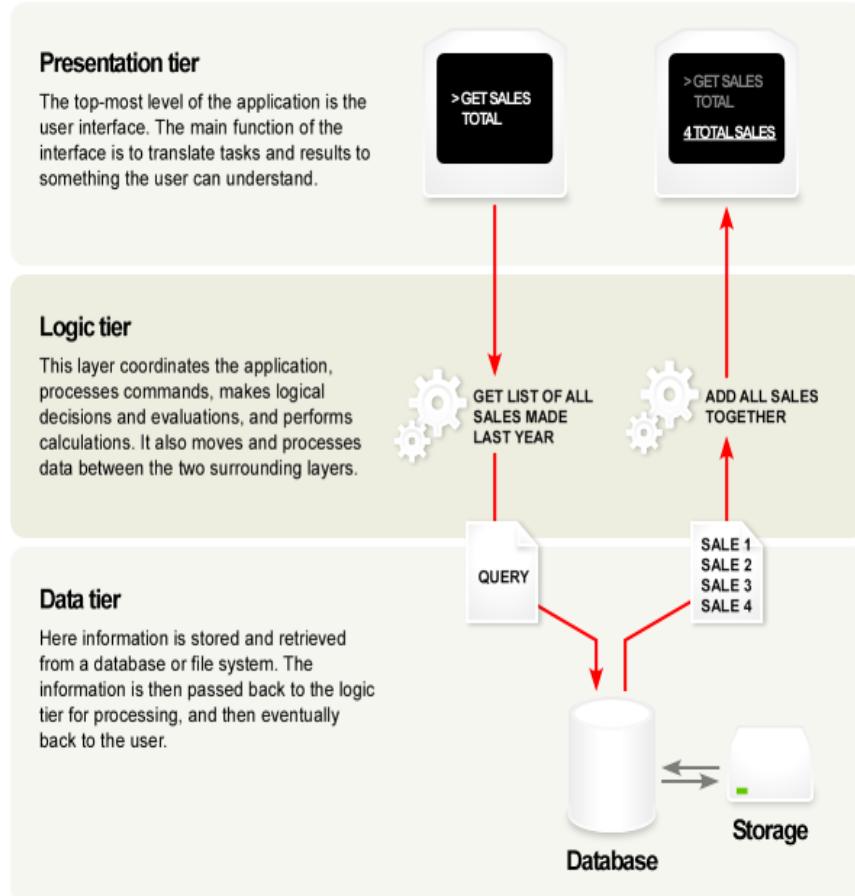
A Typical 3-tier Architecture



Presentation Layer

- Provides user interface
- Handles the interaction with the user
- Sometimes called the GUI or client view or **front-end**
- Should not contain business logic or data access code

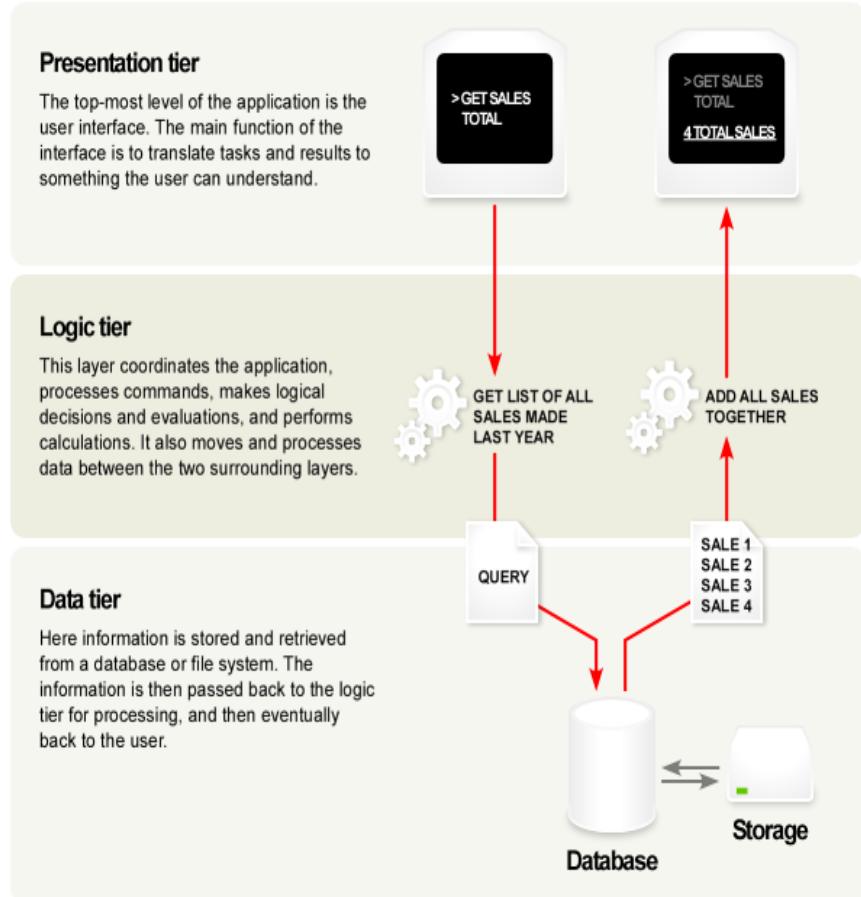
A Typical 3-tier Architecture



Logic Layer

- The set of rules for processing information
- Can accommodate many users
- Sometimes called middleware/back-end
- Should not contain presentation or data access code

A typical 3-tier Architecture



Data Layer

- The physical storage layer for data persistence
- Manages access to DB or file system
- Sometimes called **back-end**
- Should not contain presentation or business logic code

The 3-Tier Architecture for Web Apps

- Presentation Layer
 - Static or dynamically generated content rendered by the browser (front-end)
- Logic Layer
 - A dynamic content processing and generation level application server, e.g., Java EE, ASP.NET, PHP, ColdFusion platform (middleware)
- Data Layer
 - A database, comprising both data sets and the database management system or RDBMS software that manages and provides access to the data (back-end)

3-Tier Architecture - Advantages

- Independence of Layers
 - Easier to maintain
 - Components are reusable
 - Faster development (division of work)
 - Web designer does presentation
 - Software engineer does logic
 - DB admin does data model

Design Patterns

Design Problems & Decisions

- Construction and testing
 - how do we build a web application?
 - what technology should we choose?
- Re-use
 - can we use standard components?
- Scalability
 - how will our web application cope with large numbers of requests?
- Security
 - how do we protect against attack, viruses, malicious data access, denial of service?
- Different data views
 - user types, individual accounts, data protection

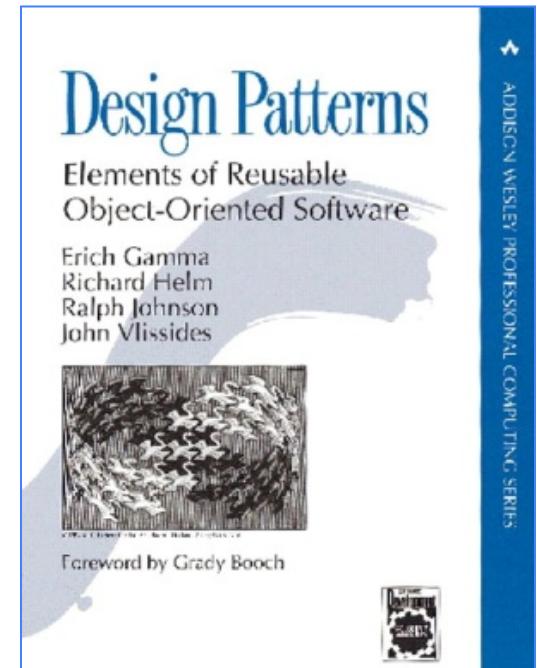
Need for general and reusable solution: **Design Patterns**

What is a Design Pattern?

- A **general** and **reusable solution** to a commonly occurring problem in the design of software
- A **template** for how to solve a problem that has been used in many different situations
- NOT a finished design
 - the pattern must be adapted to the application
 - cannot simply translate into code

Origin of Design Patterns

- Architectural concept by Christopher Alexander (1977/79)
- Adapted to OO Programming by Beck and Cunningham (1987)
- Popularity in CS after the book: “Design Patterns: Elements of Re-useable Object-oriented software”, 1994. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
- Now widely-used in software engineering



The MVC Design Pattern

Design Problem

- Need to change the **look-and-feel** without changing the **core/logic**
- Need to **present data under different contexts** (e.g., powerful desktop, web, mobile device).
- Need to **interact** with/access data under **different contexts** (e.g., touch screen on a mobile device, keyboard on a computer)
- Need to maintain **multiple views** of the **same data** (list, thumbnails, detailed, etc.)

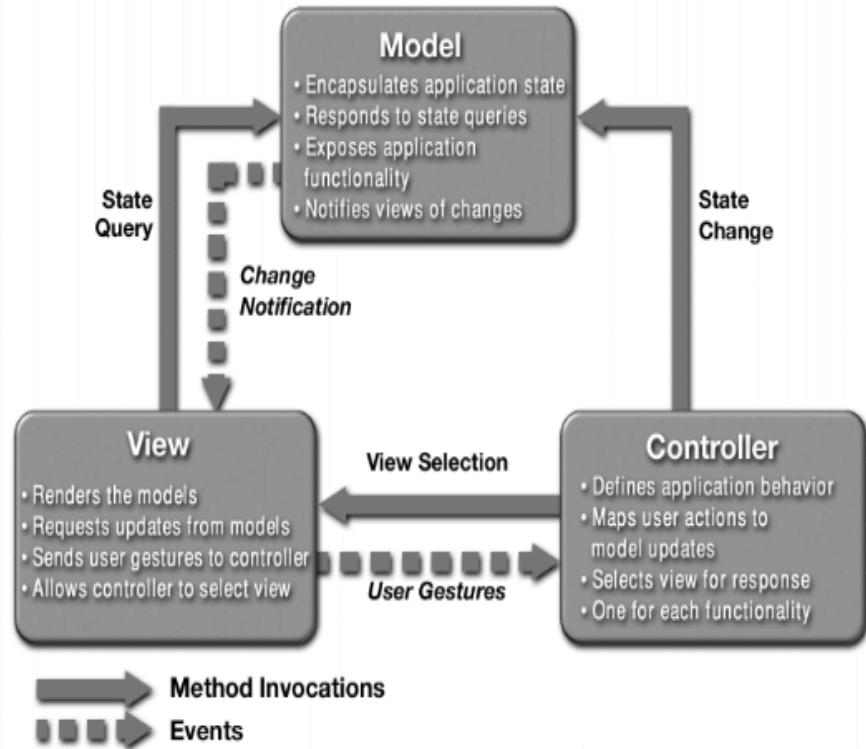
Design Solution

- Separate core functionality from the presentation and control logic that uses this functionality
- Allow multiple views to share the same data model
- Make supporting multiple clients easier to implement, test, and maintain

The Model-View-Controller Pattern

Design pattern for graphical systems
that **promotes separation between**
model and view

With this pattern the logic required
for data **maintenance** (database,
text file) **is separated** from how the
data is **viewed** (graph, numerical)
and how the data can be **interacted**
with (GUI, command line, touch)



The MVC Pattern

Model

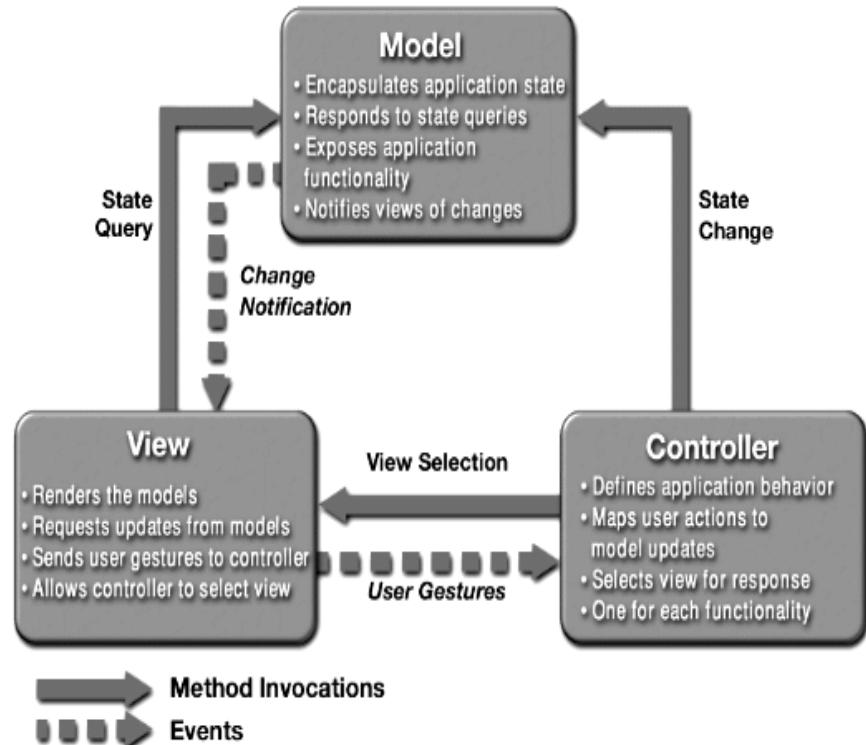
- manages the behavior and data of the application domain
- responds to requests for information about its state (usually from the view)
- follows instructions to change state (usually from the controller)

View

- renders the model into a form suitable for interaction, typically a user interface (multiple views can exist for a single model for different purposes)

Controller

- receives user input and initiates a response by making calls on model objects
- accepts input from the user and instructs the model and view to perform actions based on that input



The MVC Pattern (in practice)

- **Model**
 - Contains domain-specific knowledge
 - Records the state of the application
 - E.g., what items are in a shopping cart
 - Often linked to a database
 - Independent of view
 - One model can link to different views
- **View**
 - Presents data to the user
 - Allows user interaction
 - Does no processing
- **Controller**
 - defines how user interface reacts to user input (events)
 - receives messages from view (where events come from)
 - sends messages to model (tells what data to display)

The MVC for Web Applications

- Model
 - database tables (persistent data)
 - session information (current system state data)
 - rules governing transactions
- View
 - (X)HTML
 - CSS style sheets
 - server-side templates
- Controller
 - client-side scripting
 - http request processing
 - business logic/preprocessing

MVC Advantages

- Clarity of Design
 - model methods give an API for data and state
 - eases the design of view and controller
- Efficient Modularity
 - any of the components can be easily replaced
- Multiple Views
 - many views can be developed as appropriate
 - each uses the same API for the model
- Easier to Construct and Maintain
 - simple (text-based) views while constructing
 - more views and controllers can be added
 - stable interfaces ease development
- Distributable
 - natural fit with a distributed environment

3-tier Architecture vs. MVC Architecture

- Communication
 - ▣ **3-tier:** The presentation layer never communicates directly with the data layer-only through the logic layer (linear topology)
 - ▣ **MVC:** All layers communicate directly (triangle topology)
- Usage
 - ▣ **3-tier:** Mainly used in web applications where the client, middleware and data tiers ran on physically separate platforms
 - ▣ **MVC:** Historically used on applications that run on a single graphical workstation (applied to separate platforms as *Model 2*)

Chapter 5: SOA and RESTFUL Web services

Service-oriented architecture (SOA) and development is a paradigm where software components are created with concise interfaces, and each component performs a discrete set of related functions. With its well-defined interface and contract for usage, each component, provides a service to other software components. This is analogous to an accountant who provides a service to a business, even though that service consists of many related functions—bookkeeping, tax filing, investment management, and so on.

There are no technology requirements or restrictions with SOA. You can build a service in any language with standards such as CORBA, remote procedure calls (RPC), or XML. Although SOA has been around as a concept for years, its vague definition makes it difficult to identify. The client/server development model of the early '90s was a simple example of an SOA-based approach to software development.

A web service is an example of an SOA with a well-defined set of implementation choices. In general, the technology choices are SOAP and the Web Service Definition Language (WSDL); both XML-based. WSDL describes the interface (the "contract"), while SOAP describes the data that is transferred. Because of the platform-neutral nature of XML, SOAP, and WSDL, Java is a popular choice for web-service implementation due to its OS-neutrality.

Web-service systems are an improvement of client/server systems, and proprietary object models such as CORBA or COM, because they're standardized and free of many platform constraints. Additionally, the standards, languages, and protocols typically used to implement web services helps systems built around them to scale better.

Representational State Transfer (REST)

However, there exists an even less restrictive form of SOA than a web service—representational state transfer (REST). Described by Roy Fielding in his doctoral dissertation, REST is a collection of principals that are technology independent, except for the requirement that it be based on HTTP.

A system that conforms to the following set of principals is said to be "RESTful":

- All components of the system communicate through interfaces with clearly defined methods and dynamic code.
- Each component is uniquely identified through a hypermedia link (URL).
- A client/server architecture is followed (web browser and web server).
- All communication is stateless.
- The architecture is tiered, and data can be cached at any layer.

These principals map directly to those used in the development of the Web, and according to Fielding, account for much of the Web's success. HTTP protocol, its interface of methods (GET, POST, HEAD, and so on), the use of URLs, HTML, and JavaScript, as well as the clear distinction between what is a web server and web browser, all map directly to the first four principals. The final principal (involving tiers) allows for the common network technology found in most website implementations: load balancers, in-memory caches, firewalls, routers, and so on. These devices are acceptable because they don't affect the interfaces between the components; they merely enhance their performance and communication.

The Web is the premier example of a RESTful system, which makes sense since much of the Web's architecture preceded the definition of REST. What the Web makes clear, however, is that complex remote procedure call protocols are not needed to create successful, scalable, understandable, and reliable distributed software systems. Instead, the principals of REST are all you need.

Overall, REST can be described as a technology and platform-independent architecture where loosely coupled components communicate via interfaces over standard web protocols. Software, hardware, and data-centric designs maximize system efficiency, scalability, and network throughput. The underlying principal is simplicity.

Figure 1 illustrates the REST architecture, combining both logical software architecture and physical network elements. Communication is performed over HTTP, clients contain optional server caches for efficiency, services employ caches to backend databases, there are no restrictions on maximum clients, or maximum services per client, services can call services, load-balancing hardware is used for scalability, and firewalls can be used for security.

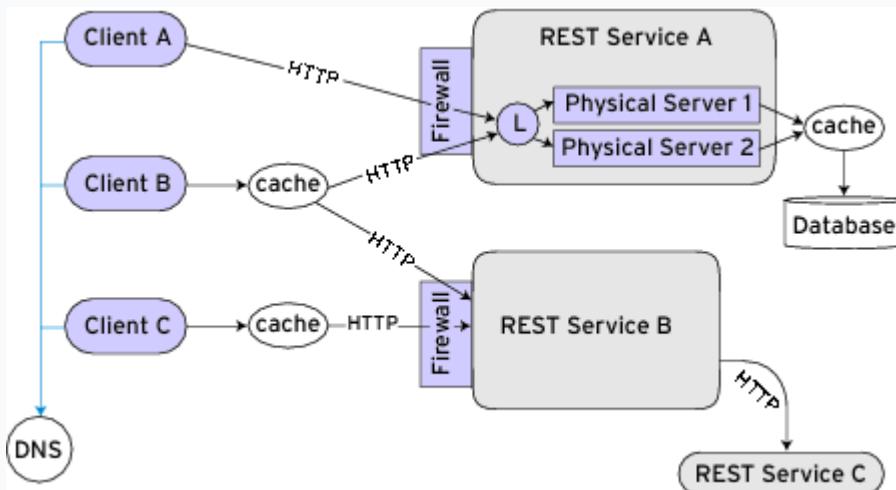


Figure 1: This architectural diagram provides a visual overview of the REST principals.

There are some interesting points on data caching that need to be made. First, data must be marked, either implicitly or explicitly, as cacheable, or noncacheable. Second, although specialized caches may be used (custom, in-memory data structures), general-purpose caches, such as web browser caches or third-party web caches (such as Akamai), can also be used.

Resource Oriented Architectures

Main Concepts

Resource Oriented Architectures (ROA) are based upon the concept of resource; each resource is a directly accessible distributed component that is handled through a standard, common interface making possible resources handling. RESTful platforms based on REST development technology enable the creation of ROA. The main ROA concepts are the following:

- Resource – anything that's important enough to be referenced as a thing itself
- Resource name – unique identification of the resource

- Resource representation – useful information about the current state of a resource
- Resource link – link to another representation of the same or another resource
- Resource interface – uniform interface for accessing the resource and manipulating its state . The resource interface semantics is based on the one of HTTP operations. The following table summarizes the resource methods and how they could be implemented¹ using the HTTP protocol:

Resource method	Description	HTTP operation
createResource	Create a new resource (and the corresponding unique identifier)	PUT
getResourceRepresentation	Retrieve the representation of the resource	GET
deleteResource	Delete the resource (optionally including linked resources)	DELETE (referred resource only), POST (can be used if the delete is including linked resources)
modifyResource	Modify the resource	POST
getMetaInformation	Obtain meta information about the resource	HEAD

Table 1

In terms of platform implementation specification, each resource must be associated to a unique identifier that usually consists of the URL exhibiting the resource interface.

Designing Resource Oriented Architectures

One of the most important decisions that have to be taken in the design of a Resource Oriented Architecture is what must be considered a resource (by definition, each component deserving to be directly represented and accessed). This is one of the main differences between ROA and SOA where in the latter one the single, directly accessible distributed component, represents one or more business functionalities that often process different potential resources. Such resources are credited candidates for being considered resources in a ROA, thus deserving to be represented as distributed components (e.g. features offered by WFS, registers or items of registries/catalogues, WFS and Registry/Catalogue services functionalities).

Once having defined the granularity of the resources composing the ROA it is necessary to define, for each resource type, the content of the messages for invoking the methods as well as the corresponding responses. More in detail, beside the definition of resource types (and sub types), an addressing schema for accessing instances of those resource types, a response schema (response is not binary) and a mapping of logical functions to the HTTP operations. All resources in a ROA are accessed via the same common interface which is plain HTTP. It is worth

noting that the usage of a common interface does not necessarily mean that all the necessary information enabling interoperability and collaboration among resources are available. The necessity of integrating a standard common interface description with some specific service instance aspects has been already addressed in other existing solutions such as, for instance, for the OGC WPS execute operation [7] where the specific processing detailed information are offered through the describe Process operation. In a ROA, this information completes the description of the resources and their interface thus enabling: i) system integration and interoperability through tools for the creation of resource clients, ii) message validation processes, and iii) model driven development (top-down approach) for which a typed description of the interface and its operation is necessary. In the REST technology such a description is based on WADL documents that play the same role of WSDL in the W3C Web Services platform; both languages use XML schema for expressing the structure of exchanged messages.

Designing Read-Only Resource-Oriented Services:

Once you have an idea of what you want your program to do. It produces a set of resources that respond to a read-only subset of HTTP's uniform interface: GET and possibly HEAD. Once you get to the end of this procedure, you should be ready to implement your resources in whatever language and framework you like. The following steps are to be done to design read-only resource-oriented services:

1. Figure out the data set
2. Split the data set into resources

For each kind of resource:

3. Name the resources with URIs
4. Expose a subset of the uniform interface
5. Design the representation(s) accepted from the client
6. Design the representation(s) served to the client
7. Integrate this resource into existing resources, using hypermedia links and forms
8. Consider the typical course of events: what's supposed to happen?
9. Consider error conditions: what might go wrong?

In the sections to come, I'll show, step by step, how following this procedure results in a RESTful web service that works like the Web.

Figure Out the Data Set:

A web service starts with a data set, or at least an idea for one. This is the data set you'll be exposing and/or getting your users to build.

Split the Data Set into Resources:

Once you have a data set in mind, the next step is to decide how to expose the data as HTTP resources. Remember that a resource is *anything interesting enough to be the target of a hypertext link*. Anything that might be referred to by name ought to have a name. Web services commonly expose three kinds of resources:

Predefined one-off resources for especially important aspects of the application.

This includes top-level directories of other available resources. Most services expose few or no one-off resources.

Example: A web site's homepage. It's a one-of-a-kind resource, at a well-known URI, which acts as a portal to other resources.

A resource for every object exposed through the service.

One service may expose many kinds of objects, each with its own resource set. Most services expose a large or infinite number of these resources.

Resources representing the results of algorithms applied to the data set.

This includes collection resources, which are usually the results of queries. Most services either expose an infinite number of algorithmic resources, or they don't expose any.

Example: A search engine exposes an infinite number of algorithmic resources. There's one for every search request you might possibly make. The Google search engine exposes one resource at <http://google.com/search?q=jellyfish> (that'd be "a directory of resources about jellyfish") and another at <http://google.com/search>?q=chocolate ("a directory of resources about chocolate"). Neither of these resources were explicitly defined ahead of time: Google translates any URI of the form <http://google.com/search?query> into an algorithmic resource "a directory of resources about {query}."

Name the Resources:

Now the resources need names. Resources are named with URIs, so let's pick some. Remember, in a resource-oriented service the URI contains all the scoping information. Our URIs need to answer questions like: "Why should the server operate on this map instead of that map?" and "Why should the server operate on this place instead of that place?"

Now let's consider the resources. The most basic resource is the list of planets. It makes sense to put this at the root URI, <http://maps.example.com/>. Since the list of planets encompasses the entire service, there's no scoping information at all for this resource (unless you count the service version as scoping information).

For the other resources I'd like to pick URIs that organize the scoping information in a natural way. There are three basic rules for URI design, born of collective experience:

1. Use path variables to encode hierarchy: /parent/child
2. Put punctuation characters in path variables to avoid implying hierarchy where none exists: /parent/child1;child2
3. Use query variables to imply inputs into an algorithm, for example:
</search?q=jellyfish&start=20>

Map URIs:

Now that I've designed the URI to a geographic point on a planet, what about the corresponding point on a road map or satellite map? After all, the main point of this service is to serve maps.

Earlier I said I'd expose a resource for every point on a map. For simplicity's sake, I'm not exposing maps of named places, only points of latitude and longitude. In addition to a set of coordinates or the name of a place, I need the name of the planet and the type of map (satellite map, road map, or whatever). Here are some URIs to maps of planets, places, and points:

- o <http://maps.example.com/radar/Venus>
- o <http://maps.example.com/radar/Venus/65.9,7.00>

- <http://maps.example.com/geologic/Earth/43.9,-103.46>

Design Your Representations:

I've decided which resources I'm exposing, and what their URIs will look like. Now I need to decide what data to send when a client requests a resource, and what data format to use.

Integrate this resource in to existing resources, using hypermedia links and forms(Link the Resources to Each Other):

Since I designed all my resources in parallel, they're already full of links to each other (see Figure 5-3). A client can get the service's "home page" (the planet list), follow a link to a specific planet, follow another link to a specific map, and then follow navigation and zoom links to jump around the map. A client can do a search for places that meet certain criteria, click one of the search results to find out more about the place, then follow another link to locate the place on a map.

One thing is still missing, though. How is the client supposed to get to a list of search results? I've set up rules for what the URI to a set of search results looks like, but if clients have to follow rules to generate URIs, my service isn't well connected. HTML solves this problem with forms. By sending an appropriate form in a representation, I can tell the client how to plug variables into a query string. The form represents infinitely many URIs, all of which follow a certain pattern. I'm going to extend my representations of places by including this HTML form.

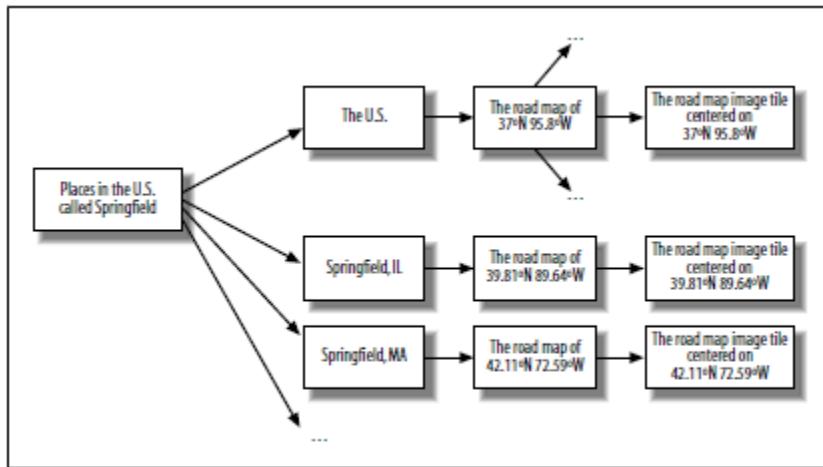


Figure 5-3. Where can you go from the list of search results?

What's Supposed to Happen?

This step of the design is conceptually simple, but it's the gateway to where you're going to spend much of your implementation time: making sure that client requests are correctly turned into responses. Most read-only resources have a pretty simple typical course of events. The user sends a GET request to a URI, and the server sends back a happy response code like 200 ("OK"), some HTTP headers, and a representation. A HEAD request works the same way, but the server omits the representation. The only main question is which HTTP headers the client should send in the request, and which ones the server should send in the response.

What Might Go Wrong?

I also need to plan for requests I can't fulfill. When I hit an error condition I'll send a response code in the 3xx, 4xx, or 5xx range, and I may provide supplementary data in HTTP headers. If

they provide an entity-body, it'll be a document describing an error condition, not a representation of the requested resource (which, after all, couldn't be served).

Here are some likely error conditions for the map application:

- The client may try to access a map that doesn't exist, like /road/Saturn. I understand what the client is asking for, but I don't have the data. The proper response code in this situation is 404 ("Not Found"). I don't need to send an entity-body along with this response code, though it's helpful for debugging.

Designing Read/Write Resource-Oriented Services:

- Same process, but now we examine full range of uniform interface operations
 - Build matrix with resource types as rows, and operations as columns
 - Indicate what operations apply to which types
 - provide example URIs and discussion of what will happen
 - especially in the case of POST and PUT
 - PUT: create or modify resource
 - POST: append content to existing resource OR append child resource to parent resource (blog entries)
 - Two questions to help
 - Will clients be creating new resources of this type?
 - Who's in charge of determining the new resource's URI? Client or Server? If the former, then PUT. If the latter, then POST.

New Issues: Authentication and Authorization

- Now that we are allowing a client to change stuff on our server, we need
 - Authentication: problem of tying a request to a user
 - Authorization: problem of determining which requests to let through for a given user
- HTTP provides mechanisms to enable this (HTTP Basic/Digest) and other web services roll their own (Amazon's public/private key on subset of request)
- Another Issue: Privacy
 - Can't transmit "private information" in the clear; need to use HTTPS
- Another Issue: Trust
 - How do you trust your client software to do the right thing?
 - Especially in today's environment with malware becoming harder and harder to discern

Chapter 6: Software Security:

Software security is an idea implemented to protect software against malicious attack and other hacker risks so that the software continues to function correctly under such potential risks. Security is necessary to provide integrity, authentication and availability.

Any compromise to integrity, authentication and availability makes a software unsecure. Software systems can be attacked to steal information, monitor content, introduce vulnerabilities and damage the behavior of software. Malware can cause DoS (denial of service) or crash the system itself.

Basic Attacks:

Buffer overflow, stack overflow, command injection and SQL injections are the most common attacks on the software.

Buffer and stack overflow attacks overwrite the contents of the heap or stack respectively by writing extra bytes.

Command injection can be achieved on the software code when system commands are used predominantly. New system commands are appended to existing commands by the malicious attack. Sometimes system command may stop services and cause DoS.

SQL injections use malicious SQL code to retrieve or modify important information from database servers. SQL injections can be used to bypass login credentials. Sometimes SQL injections fetch important information from a database or delete all important data from a database.

The only way to avoid such attacks is to practice good programming techniques. System-level security can be provided using better firewalls. Using intrusion detection and prevention can also aid in stopping attackers from easy access to the system.

Cross-site Scripting (XSS)

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page

Cross-Site Scripting (XSS) attacks occur when:

1. Data enters a Web application through an untrusted source, most frequently a web request.
2. The data is included in dynamic content that is sent to a web user without being validated for malicious content.

The malicious content sent to the web browser often takes the form of a segment of JavaScript, but may also include HTML, Flash, or any other type of code that the browser may execute. The variety

of attacks based on XSS is almost limitless, but they commonly include transmitting private data, like cookies or other session information, to the attacker, redirecting the victim to web content controlled by the attacker, or performing other malicious operations on the user's machine under the guise of the vulnerable site.

Stored and Reflected XSS Attacks

XSS attacks can generally be categorized into two categories: stored and reflected..

Stored XSS Attacks

Stored attacks are those where the injected script is permanently stored on the target servers, such as in a database, in a message forum, visitor log, comment field, etc. The victim then retrieves the malicious script from the server when it requests the stored information. Stored XSS is also sometimes referred to as Persistent or Type-I XSS.

Reflected XSS Attacks

Reflected attacks are those where the injected script is reflected off the web server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request. Reflected attacks are delivered to victims via another route, such as in an e-mail message, or on some other web site. When a user is tricked into clicking on a malicious link, submitting a specially crafted form, or even just browsing to a malicious site, the injected code travels to the vulnerable web site, which reflects the attack back to the user's browser. The browser then executes the code because it came from a "trusted" server. Reflected XSS is also sometimes referred to as Non-Persistent or Type-II XSS.

Dos and Don'ts of client authentication:

Client authentication is a common requirement for modern Web sites as more and more personalized and access-controlled services move online. Unfortunately, many sites use authentication schemes that are extremely weak and vulnerable to attack. These problems are most often due to careless use of authenticators stored on the client. We observed this in an informal survey of authentication mechanisms used by various popular Web sites. Of the twenty-seven sites we investigated, we weakened the client authentication of two systems, gained unauthorized access on eight, and extracted the secret key used to mint authenticators from one.

This is perhaps surprising given the existing client authentication mechanisms within HTTP and SSL/TLS , two well-studied mechanisms for providing authentication secure against a range of adversaries. However, there are many reasons that these mechanisms are not suitable for use on the Web at large. Lack of a central infrastructure such as a public-key infrastructure or a uniform Kerberos contributes to the proliferation of weak schemes. We also found that many Web sites would design their own authentication mechanism to provide a better user experience.

Unfortunately, designers and implementers often do not have a background in security and, as a result, do not have a good understanding of the tools at their disposal. Because of this lack of control over user interfaces and unavailability of a client authentication infrastructure, Web sites continue to reinvent weak home-brew client authentication schemes.

Clients want to ensure that only authorized people can access and modify personal information that they share with Web sites. Similarly, Web sites want to ensure that only authorized users have access to the services and content it provides. Client authentication addresses the needs of both parties.

Client authentication involves proving the identity of a *client* (or user) to a *server* on the Web. We

will use the term ``authentication'' to refer to this problem. Server authentication, the task of authenticating the server to the client, is also important but is not the focus this paper.

Practical limitations

Deployability:

- Technology must be widely deployed
- HTTP is stateless and sessionless
- Client must provide authentication token
- Useful but high overhead
- Javascript, Flash, Shockwave...

User Acceptability: Web sites must also consider user acceptability. Because sites want to attract many users, the client authentication must be as non-confrontational as possible. Users will be discouraged by schemes requiring work such as installing a plug-in or clicking away dialog boxes.

Performances: Stronger security protocols generally cost more in performance. Service providers naturally want to respond to as many requests as possible. Cryptographic solutions will usually degrade server performance. Authentication should not needlessly consume valuable server resources such as memory and clock cycles. With current technology, SSL becomes unattractive because of the computational cost of its initial handshaking.

Hints for Web client authentication

- Use Cryptography Appropriately
- Protect Passwords
- Handle Authenticators Carefully
- Use cryptography appropriately
- Appropriate amount of security

Keep It Simple, Stupid

- Do not be inventive

Designers should be security experts

- Do not rely on the secrecy of a protocol

Vulnerable to exposure

- Understand the properties of cryptographic tools

Example: Crypt()

- Do not compose security schemes

Hard to foresee the effects

Protect Passwords

Limit exposure

Don't send it back to the user (much less in the clear)

Authenticate using SSL vs. HTTP

Prohibit guessable passwords

No dictionary passwords

Reauthenticate before changing passwords

Avoid replay attack

Handle authenticators carefully

Make authenticators unforgeable

highschoolalumni.com

If using keys as session identifier: should be cryptographically random

Protect from tampering (MAC)

Protect authenticators that must be secret

Authenticator as cookie

Sent by SSL

Don't forget the flag! (SprintPCS)

Authenticator as part of URL

Avoid using persistent cookies

Persistent vs. ephemeral cookies

Cookie files on the web

Limit the lifetime of authenticators

Encrypt the timestamp

Secure binding limits the damage from stolen authenticators

Bind authenticators to specific network addresses

Increases the difficulty of a replay attack

SQL INJECTION:

SQL injection is a type of [web application security](#) vulnerability in which an attacker is able to submit a database SQL command that is executed by a web application, exposing the back-end database. A SQL injection attack can occur when a web application utilizes user-supplied data without proper validation or encoding as part of a command or query. The specially crafted user data tricks the application into executing unintended commands or changing data. SQL injection allows an attacker to create, read, update, alter or delete data stored in the back-end database. In its most common form, a SQL injection attack gives access to sensitive information such as social security numbers, credit card numbers or other financial data.

Key Concepts of a SQL Injection Attack

- SQL injection is a software vulnerability that occurs when data entered by users is sent to the SQL interpreter as a part of a SQL query.
- Attackers provide specially crafted input data to the SQL interpreter and trick the interpreter to execute unintended commands.
- Attackers utilize this vulnerability by providing specially crafted input data to the SQL interpreter in such a manner that the interpreter is not able to distinguish between the intended commands and the attacker's specially crafted data. The interpreter is tricked into executing unintended commands.
- A SQL injection attack exploits security vulnerabilities at the database layer. By exploiting the SQL injection flaw, attackers can create, read, modify or delete sensitive data.

How SQL Injection works

In order to run malicious SQL queries against a database server, an attacker must first find an input within the web application that is included inside of an SQL query.

In order for an SQL injection attack to take place, the vulnerable website needs to directly include user input within an SQL statement. An attacker can then insert a payload that will be included as part of the SQL query and run against the database server.

Lets look at an example code

```
String SQLQuery ="SELECT Username, Password  
FROM users WHERE Username=' " + Username +  
" ' AND Password=' " + Password + " ' ;  
  
Statement stmt = connection.createStatement();  
ResultSet rs = stmt.executeQuery(SQLQuery);  
while (rs.next()) { ... }
```

You will notice that user input is required to run this query. The interpreter will execute the

```
String SQLQuery ="SELECT Username, Password  
FROM users WHERE Username=' " + Username +  
" ' AND Password=' " + Password + " ' ;
```

command based on the inputs received for the username and password

fields.

If an attacker provides ‘or 0=0’ as the username and password, then the query will be constructed as:

*String SQLQuery ="SELECT Username, Password FROM users WHERE Username=" or 0=0"
AND Password=" or 0=0";*

```
String SQLQuery ="SELECT Username, Password  
FROM users WHERE Username=" or 0=0"  
" ' AND Password=" or 0=0" " ' ;
```

Since under all circumstances, zero will be equal to zero, the query will return all records in the database. In this way, an unauthorized user will be able to view sensitive information.

Preventing SQL Injection

- You can prevent SQL injection if you adopt an input validation technique in which user input is authenticated against a set of defined rules for length, type and syntax and also against business rules.

- You should ensure that users with the permission to access the database have the least privileges. Additionally, do not use system administrator accounts like “sa” for web applications. Also, you should always make sure that a database user is created only for a specific application and this user is not able to access other applications. Another method for preventing SQL injection attacks is to remove all stored procedures that are not in use.
- Use strongly typed parameterized query APIs with placeholder substitution markers, even when calling stored procedures.
- Show care when using stored procedures since they are generally safe from injection. However, be careful as they can be injectable (such as via the use of exec() or concatenating arguments within the stored procedure).

State-Based attacks:

The concept of state, or the ability to remember information as a user travels from page to page within a site, is an important one for Web testers. The Web is stateless in the sense that it does not remember which page a user is viewing or the order in which pages may be viewed. A user is always free to click the Back button or to force a page to reload. Thus, developers of Web applications must take it upon themselves to code state information so they can enforce rules about page access and session management.

The first option is using **forms** and **CGI parameters**, which allow the transfer of small amounts of data and information to be passed from page to page, essentially allowing the developer to bind together pairs of pages. More sophisticated state requirements mean that data needs to be stored, either on the client or the server, and then made available to various pages that have to check these values when they are loaded.

Attack: Hidden Fields

- Something that is part of page, but not shown
 - Defined parameters & assigned values
 - Sent back to server to communicate state
 - Part of a form

Hidden Field Example

```
<html>
<head>
<title>My Page</title>
</head>
<body>
<form name="myform" action="http://www.foo.com/form.php" method="POST">
<div align="center">
<input type="text" size="25" value="Enter your name here!">
<input type="hidden" name="Language" value="English">
<br><br>
</div>
</form>
</body>
</html>
```

Hidden Field Attack

- Look for them (scan source)
- Change their values

Protection - Hidden Fields

- Use misleading names for hidden fields
- Use hashed/encrypted values
- Don't use them!
- Use them for innocuous input only
- Treat as user input!
- Check their values (content filtering again)

CGI Parameters

- Part of the GET request with URL
 - Name-Value pairs
 - At end of URL (after ?)
 - Pairs separated by &
- `http://www.foo.com/script.php?user=mike&passwd=guessWho`
- Data is sent to server for processing

Parameter Attack

- View source
 - Observe URL targets
 - In status bar
 - In tool tip
 - Edit request URL by hand
- <http://www.foo.com/script.php?user=mike&passwd=bigDummy>
- Try changing values
 - record=notMine
 - item=6789
 - Adding parameters
 - debug=on
 - debug=true
 - debug=1

Protection - Parameters

- Treat as user input!
 - Check values (content filtering again)
- GET - POST, Who-What?
- Both send info to server
 - GET
 - Per W3C, for idempotent form processing
 - No side effect (e.g., database search)
 - POST
 - For transactions that change state on server
 - Have a side effect (e.g., database update/insertion)
 - For sending large requests

Reload/refresh a GET vsa POST form – different behaviors

Cookie Poisoning: How to Off the Cookie Monster

- Cookies revisited
 - Local file to store data
 - Data about you, session, etc...
 - Plain text
- Change information in cookie file
 - Expiration date
 - Authentication data
 - Shopping cart

Protection - Cookies

- Don't use cookies for any authentication data
- Encrypt critical data - not fool proof
- Treat cookies as user input
- Be careful what you trust!

Chapter 7: Software Development Methodologies

A software development methodology is a way of managing a software development project. This typically address issues like selecting features for inclusion in the current version, when software will be released, who works on what, and what testing is done.

No one methodology is best for all situations. Even the much maligned waterfall method is appropriate for some organizations. In practice, every organization implements their software development project management in a different way, which is often slightly different from one project to the next. None the less, nearly all are using some subset or combination of the ones discussed here.

Choosing an appropriate management structure can make a big difference in achieving a successful end result when measured in terms of cost, meeting deadlines, client happiness, robustness of software, or minimizing expenditures on failed projects. As such, it is worth your time to learn about a number of these and make your best effort to choose wisely.

Agile family - Agile methods are meant to adapt to changing requirements, minimize development costs, and still give reasonable quality software. Agile projects are characterized by many incremental releases each generated in a very short period of time. Typically all members of the team are involved in all aspects of planning, implementation, and testing. This is typically used by small teams, perhaps nine or fewer, who can have daily face-to-face interaction. Teams may include a client representative. There is a strong emphasis on testing as software is written. The disadvantages of the Agile methods are that they work poorly for projects with hundreds of developers, or lasting decades, or where the requirements emphasize rigorous documentation and well documented design and testing.

- SCRUM - is currently the most popular implementation of the agile ideals. Features are added in short sprints (usually 7-30 days), and short frequent meetings keep people focused. Tasks are usually tracked on a scrum board. The group is self-organizing and collaboratively managed, although there is a scrum master tasked with enforcing the rules and buffering the team from outside distractions.
- Dynamic Systems Development Model (DSDM) - is an agile method that sets time, quality, and cost at the beginning of the project. This is accomplished by prioritizing features into musts, shoulds, coulds, and won't haves. Client involvement is critical to setting these priorities. There is a pre-project planning phase to give the project a well-considered initial direction. This works well if time, cost, and quality are more important than the completeness of the feature set.
- Rapid Application Development (RAD) - is a minimalist agile method with an emphasis on minimizing planning, and a focus on prototyping and using reusable components. This can be the best choice when a good prototype is good enough to serve as the final product. RAD has been criticized because the lack of structure leads to failed projects or poor quality products if there is not a team of good developers that feel personally committed to the project.
- Extreme Programming (XP) - is a frequent release development methodology in which developers work in pairs for continuous code review. This gives very robust, high quality software, at the expense of twice the development cost. There is a strong emphasis on test driven development.
- Feature-Driven Development (FDD) - is an iterative development process with more emphasis on planning out the overall architecture, followed by implementing features in a logical order.
- Internet-Speed Development - is an iterative format that emphasizes daily builds. It is tailored to the needs of open source projects where volunteer developers are geographically distributed, and working around the clock. The project is built from a

vision and scope statement, but there are no feature freezes. Development is separated into many small pieces that can be developed in parallel. The down side of this process is that the code is constantly in flux, so there are not necessarily stable release points where the code is particularly well tested and robust.

Agile: Strengths and Weaknesses

Strengths

- ... Iterative-incremental process
- ... Based on modeling the problem domain and the system
- ... Requirements are allowed to evolve over time.
- ... Traceability to requirements through the Product Backlog
- ... Architecture of the system drafted before the development engine is started.
- ... Iterative development engine governed by careful planning and reviewing planning and
- ... Active user involvement
- ... Simple and straightforward process
- ... Simple and straightforward process
- ... Early and frequent releases, demonstrating functionality at the end of each iteration (sprint) of the development cycle.

Weaknesses

- ... Integration is done after all increments are built
- ... Lack of scalability
- ... Lack of scalability
- ... Based on the assumption that human communication is sufficient for running projects of any size and keeping them focused
- ... Not necessarily seamless (details of tasks are not prescribed)
- ... No clear-cut design effort
- ... Model-phobic
- ... Models are not prescribed, leaving it to the developer to decide what model can be useful
- ... Lack of formalism

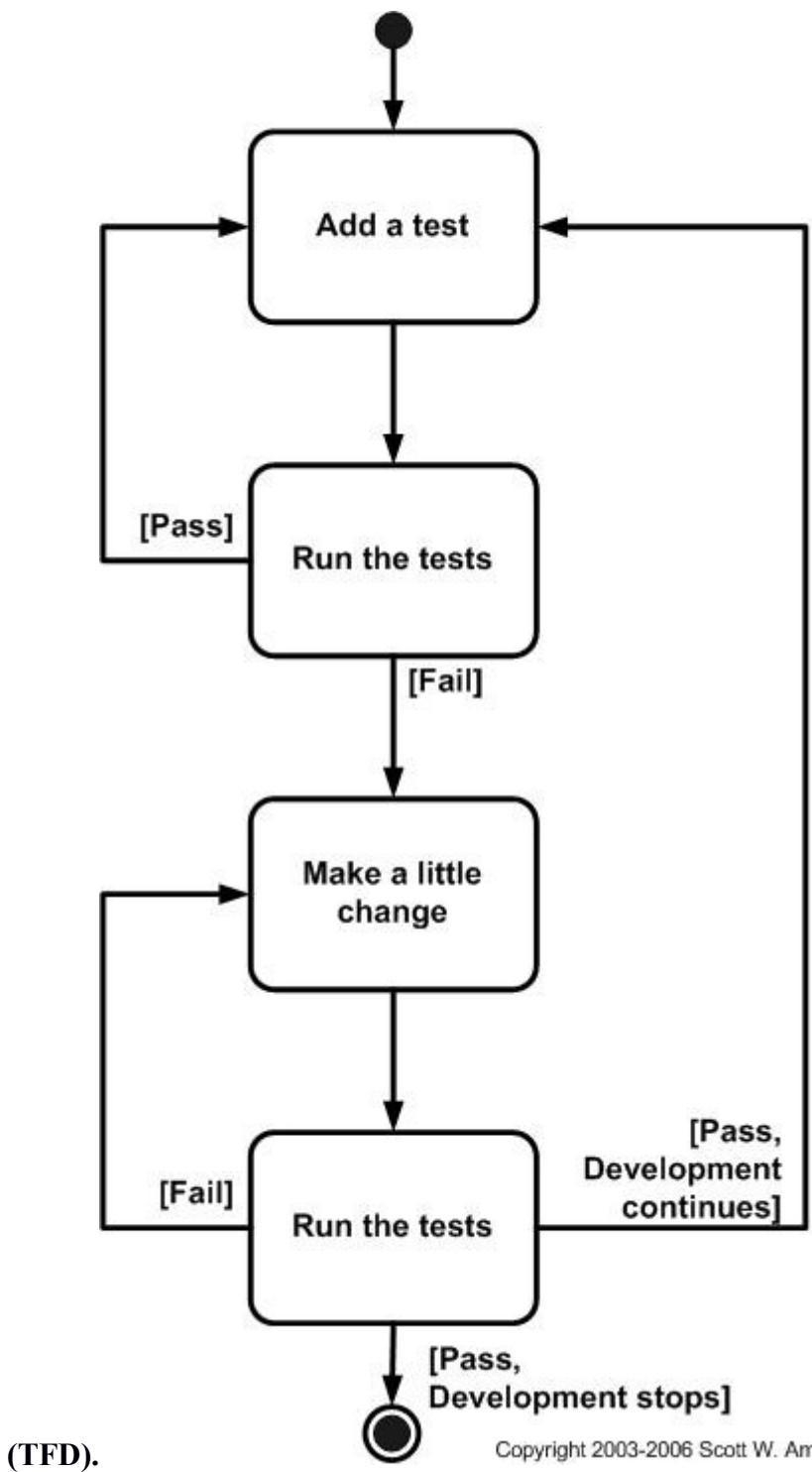
Test Driven Development

Test-driven development (TDD) ([Beck 2003](#); [Astels 2003](#)), is an evolutionary approach to development which combines test-first development where you write a test before you write just enough production code to fulfill that test and [refactoring](#). What is the primary goal of TDD? One view is the goal of TDD is specification and not validation ([Martin, Newkirk, and Kess 2003](#)). In other words, it's one way to think through your requirements or design before you write your functional code (implying that TDD is both an important [agile requirements](#) and [agile design](#) technique). Another view is that TDD is a programming technique.

What is TDD?

The steps of test first development (TFD) are overviewed in the [UML activity diagram of Figure 1](#). The first step is to quickly add a test, basically just enough code to fail. Next you run your tests, often the complete test suite although for sake of speed you may decide to run only a subset, to ensure that the new test does in fact fail. You then update your functional code to make it pass the new tests. The fourth step is to run your tests again. If they fail you need to update your functional code and retest. Once the tests pass the next step is to start over (you may first need to refactor any duplication out of your design as needed, turning TFD into TDD).

Figure 1. The Steps of test-first development



(TFD).

Copyright 2003-2006 Scott W. Ambler

TDD can be described with this simple formula:

$$\text{TDD} = \text{Refactoring} + \text{TFD}.$$

TDD completely turns traditional development around. When you first go to implement a new feature, the first question that you ask is whether the existing design is the best design possible that enables you to implement that functionality. If so, you proceed via a TFD approach. If not, you refactor it locally to change the portion of the design affected by the new feature, enabling you to add that feature as easy as possible. As a result you will always be improving the quality of your

design, thereby making it easier to work with in the future.

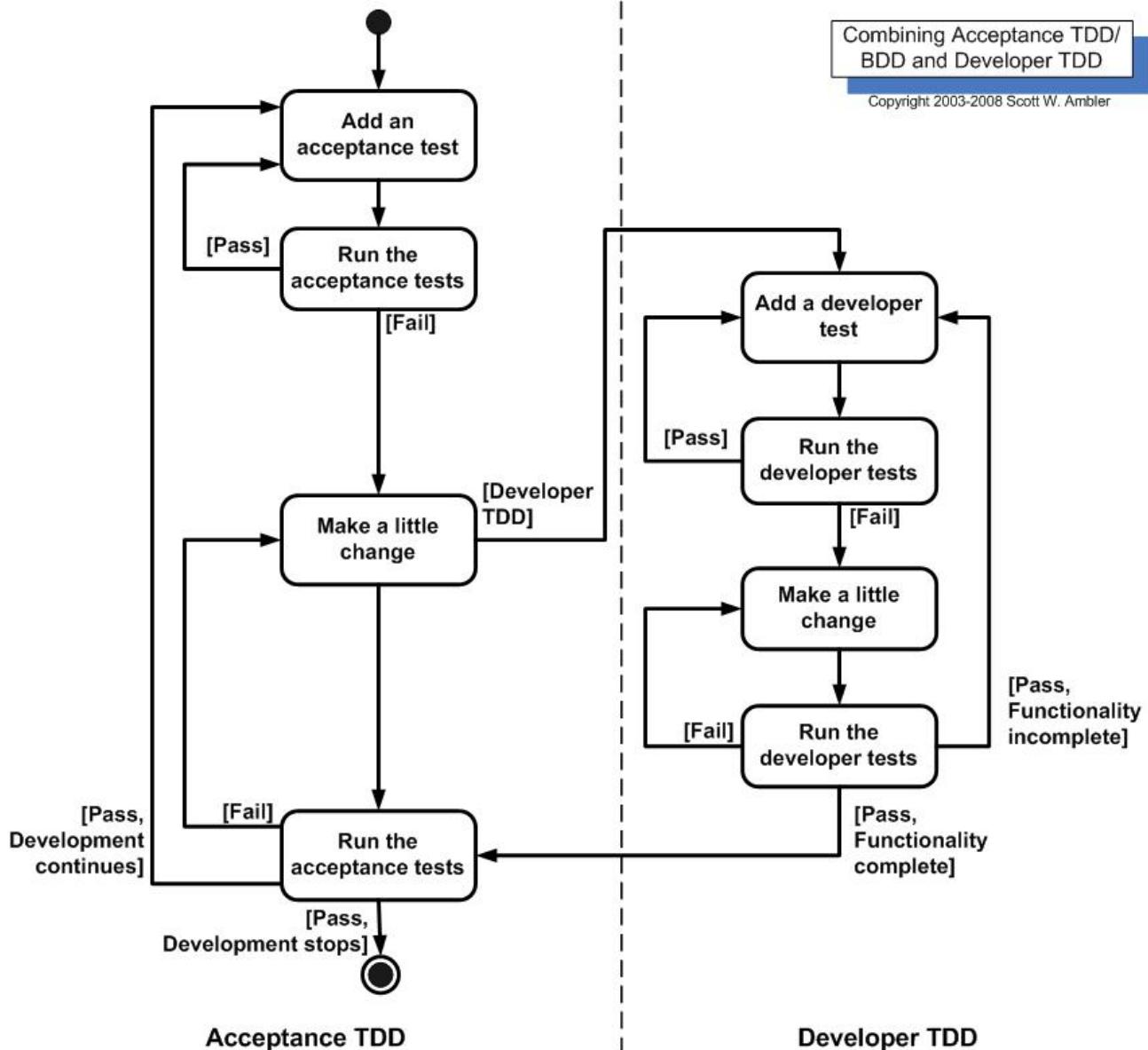
Instead of writing functional code first and then your testing code as an afterthought, if you write it at all, you instead write your test code before your functional code. Furthermore, you do so in very small steps – one test and a small bit of corresponding functional code at a time. A programmer taking a TDD approach refuses to write a new function until there is first a test that fails because that function isn't present. In fact, they refuse to add even a single line of code until a test exists for it. Once the test is in place they then do the work required to ensure that the test suite now passes (your new code may break several existing tests as well as the new one). This sounds simple in principle, but when you are first learning to take a TDD approach it proves require great discipline because it is easy to “slip” and write functional code without first writing a new test. One of the advantages of [pair programming](#) is that your pair helps you to stay on track.

There are two levels of TDD:

1. **Acceptance TDD (ATDD).** With ATDD you write a single [acceptance test](#), or behavioral specification depending on your preferred terminology, and then just enough production functionality/code to fulfill that test. The goal of ATDD is to specify detailed, executable requirements for your solution on a just in time (JIT) basis. ATDD is also called Behavior Driven Development (BDD).
2. **Developer TDD.** With developer TDD you write a single developer test, sometimes inaccurately referred to as a unit test, and then just enough production code to fulfill that test. The goal of developer TDD is to specify a detailed, executable design for your solution on a JIT basis. Developer TDD is often simply called TDD.

[Figure 2](#) depicts a UML activity diagram showing how ATDD and developer TDD fit together. Ideally, you'll write a single acceptance test, then to implement the production code required to fulfill that test you'll take a developer TDD approach. This in turn requires you to iterate several times through the write a test, write production code, get it working cycle at the developer TDD level. -

Figure 2. How acceptance TDD and developer TDD work together.

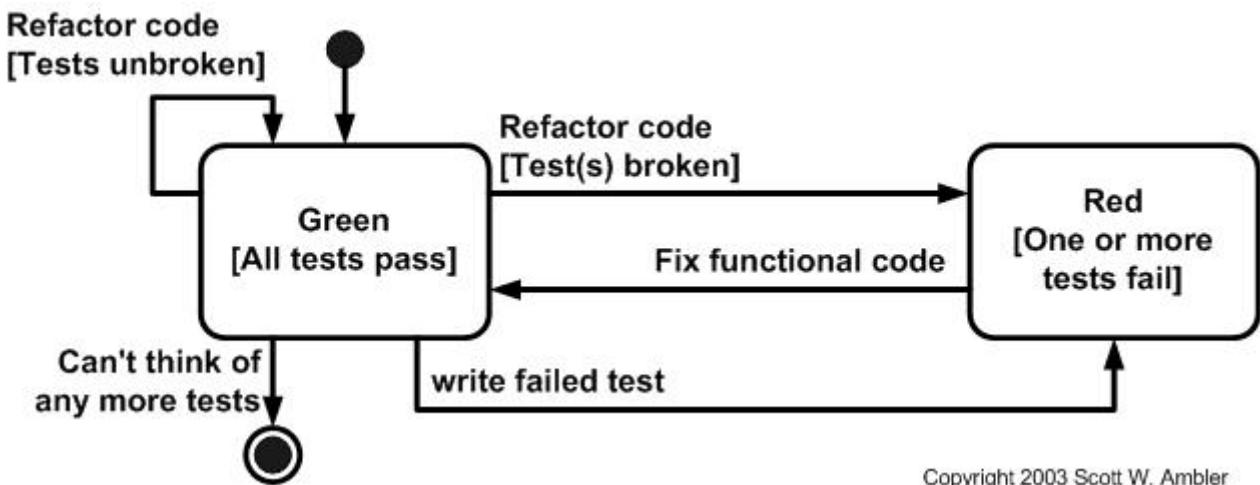


Note that [Figure 2](#) assumes that you're doing both, although it is possible to do either one without the other. In fact, some teams will do developer TDD without doing ATDD, see [survey results below](#), although if you're doing ATDD then it's pretty much certain you're also doing developer TDD. The challenge is that both forms of TDD require practitioners to have technical testing skills, skills that many requirement professionals often don't have (yet another reason why [generalizing specialists](#) are preferable to specialists).

An underlying assumption of TDD is that you have a testing framework available to you. For acceptance TDD people will use tools such as [Fitnesse](#) or [RSpec](#) and for developer TDD agile software developers often use the xUnit family of open source tools, such as [JUnit](#) or [VBUnit](#), although commercial tools are also viable options. Without such tools TDD is virtually impossible. [Figure 3](#) presents a UML state chart diagram for how people typically work with such tools. This

diagram was suggested by [Keith Ray](#). -

Figure 3. Testing via the xUnit Framework.



Kent Beck, who popularized TDD in eXtreme Programming (XP) ([Beck 2000](#)), defines two simple rules for TDD ([Beck 2003](#)). First, you should write new business code only when an automated test has failed. Second, you should eliminate any duplication that you find. Beck explains how these two simple rules generate complex individual and group behavior:

- You develop organically, with the running code providing feedback between decisions.
- You write your own tests because you can't wait 20 times per day for someone else to write them for you.
- Your development environment must provide rapid response to small changes (e.g. you need a fast compiler and regression test suite).
- Your designs must consist of highly cohesive, loosely coupled components (e.g. your design is highly normalized) to make testing easier (this also makes evolution and maintenance of your system easier too).

For developers, the implication is that they need to learn how to write effective unit tests. Beck's experience is that good unit tests:

- Run fast (they have short setups, run times, and break downs).
- Run in isolation (you should be able to reorder them).
- Use data that makes them easy to read and to understand.
- Use real data (e.g. copies of production data) when they need to.
- Represent one step towards your overall goal.

TDD and Traditional Testing

TDD is primarily a specification technique with a side effect of ensuring that your source code is thoroughly tested at a confirmatory level. However, there is more to testing than this. Particularly at scale you'll still need to consider other [agile testing](#) techniques such as [pre-production integration testing](#) and [investigative testing](#).

Much of this testing can also be done early in your project if you choose to do so (and you should).

With traditional testing a successful test finds one or more defects. It is the same with TDD; when a test fails you have made progress because you now know that you need to resolve the problem. More importantly, you have a clear measure of success when the test no longer fails. TDD increases your confidence that your system actually meets the requirements defined for it, that your system actually works and therefore you can proceed with confidence.

As with traditional testing, the greater the risk profile of the system the more thorough your tests need to be. With both traditional testing and TDD you aren't striving for perfection, instead you are testing to the importance of the system. To paraphrase [Agile Modeling \(AM\)](#), you should "test with a purpose" and know why you are testing something and to what level it needs to be tested. An interesting side effect of TDD is that you achieve 100% coverage test – every single line of code is tested – something that traditional testing doesn't guarantee (although it does recommend it). In general I think it's fairly safe to say that although TDD is a specification technique, a valuable side effect is that it results in significantly better code testing than do traditional techniques.

Comparing TDD and AMDD:

- TDD shortens the programming feedback loop whereas AMDD shortens the modeling feedback loop.
- TDD provides detailed specification (tests) whereas AMDD is better for thinking through bigger issues.
- TDD promotes the development of high-quality code whereas AMDD promotes high-quality communication with your stakeholders and other developers.
- TDD provides concrete evidence that your software works whereas AMDD supports your team, including stakeholders, in working toward a common understanding.
- TDD “speaks” to programmers whereas AMDD speaks to business analysts, stakeholders, and data professionals.
- TDD provides very finely grained concrete feedback on the order of minutes whereas AMDD enables verbal feedback on the order of minutes (concrete feedback requires developers to follow the practice Prove It With Code and thus becomes dependent on non-AM techniques).
- TDD helps to ensure that your design is clean by focusing on creation of operations that are

callable and testable whereas AMDD provides an opportunity to think through larger design/architectural issues before you code.

- TDD is non-visually oriented whereas AMDD is visually oriented.
- Both techniques are new to traditional developers and therefore may be threatening to them.
- Both techniques support evolutionary development.

Which approach should you take? The answer depends on your, and your teammates, cognitive preferences. Some people are primarily "visual thinkers", also called spatial thinkers, and they may prefer to think things through via drawing. Other people are primarily text oriented, non-visual or non-spatial thinkers, who don't work well with drawings and therefore they may prefer a TDD approach. Of course most people land somewhere in the middle of these two extremes and as a result they prefer to use each technique when it makes the most sense. In short, the answer is to use the two techniques together so as to gain the advantages of both.

How do you combine the two approaches? AMDD should be used to create models with your project stakeholders to help explore their requirements and then to explore those requirements sufficiently in architectural and design models (often simple sketches). TDD should be used as a critical part of your build efforts to ensure that you develop clean, working code. The end result is that you will have a high-quality, working system that meets the actual needs of your project stakeholders.

Why TDD?

A significant advantage of TDD is that it enables you to take small steps when writing software. This is a practice that have promoted for years because it is far more productive than attempting to code in large steps. For example, assume you add some new functional code, compile, and test it. Chances are pretty good that your tests will be broken by defects that exist in the new code. It is much easier to find, and then fix, those defects if you've written two new lines of code than two thousand. The implication is that the faster your compiler and regression test suite, the more attractive it is to proceed in smaller and smaller steps. I generally prefer to add a few new lines of functional code, typically less than ten, before I recompile and rerun my tests.

Behaviour Driven:

Behavior-driven development (BDD) is a software development methodology in which an application is specified and designed by describing how its behavior should appear to an outside observer.

A typical business application project would begin by having stakeholders offer concrete examples of the behavior they expect to see from the system. All coding efforts are geared toward delivering these desired behaviors. The real-life examples gleaned from stakeholders are converted into acceptance criteria with validation tests that are often automated. The results of these tests provide confidence to stakeholders that their desired business objectives for the software are being achieved. Ideally, the reports are generated in such a way that the average stakeholder can understand the business logic of the application. Living documentation is used throughout the system to ensure that all documentation is up to date and accurate.

In practice, behavior-driven development may be similar to [test-driven development](#) when all stakeholders have programming knowledge and skills. However, in many organizations, BDD offers the ability to enlarge the pool of input and feedback to include business stakeholders and end users who may have little software development knowledge. Because of this expanded feedback loop, BDD may more readily be used in [continuous integration](#) and continuous delivery environments.

BDD practices

The practices of BDD include:

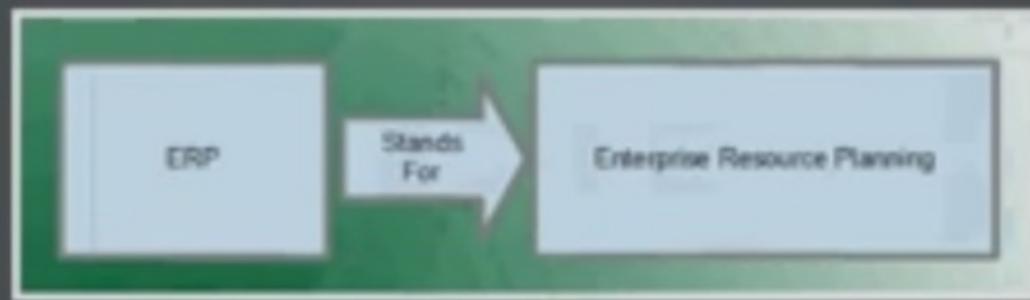
- Establishing the goals of different stakeholders required for a vision to be implemented
- Drawing out features which will achieve those goals using feature injection
- Involving stakeholders in the implementation process through outside-in software development
- Using examples to describe the behavior of the application, or of units of code
- Automating those examples to provide quick feedback and regression testing
- Using ‘should’ when describing the behavior of software to help clarify responsibility and allow the software’s functionality to be questioned
- Using ‘ensure’ when describing responsibilities of software to differentiate outcomes in the scope of the code in question from side-effects of other elements of code.
- Using mocks to stand-in for collaborating modules of code which have not yet been written

BDD vs TDD

- It is important to keep BDD distinct from TDD. These two practices are equally important but address different concerns and should be complementary in best development practices.
- BDD is concerned primarily with the specification of the behavior of the system under test as a whole, thus is particularly suited for acceptance and regression testing. TDD is concerned primarily with the testing of a component as a unit, in isolation from other dependencies, which are typically mocked or stubbed.
- BDD should talk the language of the business domain and not the language of the development technology, which on the other hand is “spoken” by TDD.

THE TERM ERP

The term "ERP" stands for "Enterprise Resource Planning".



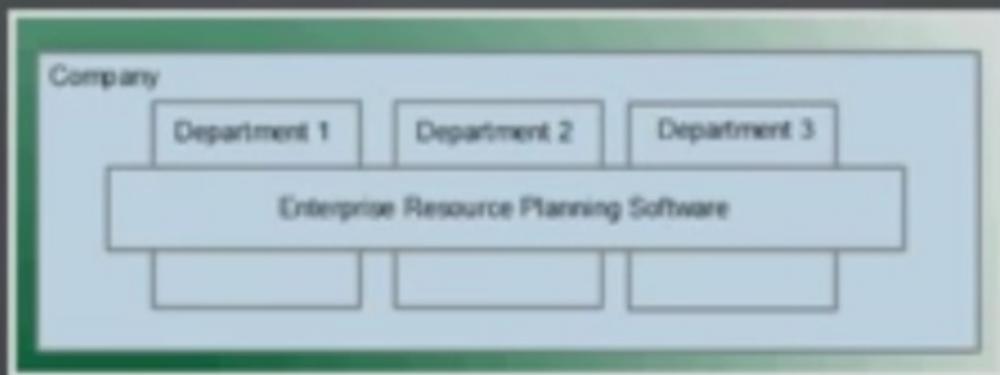
TYPE OF SOFTWARE

ERP is not a name of any software, instead it is a class (or type) of software.



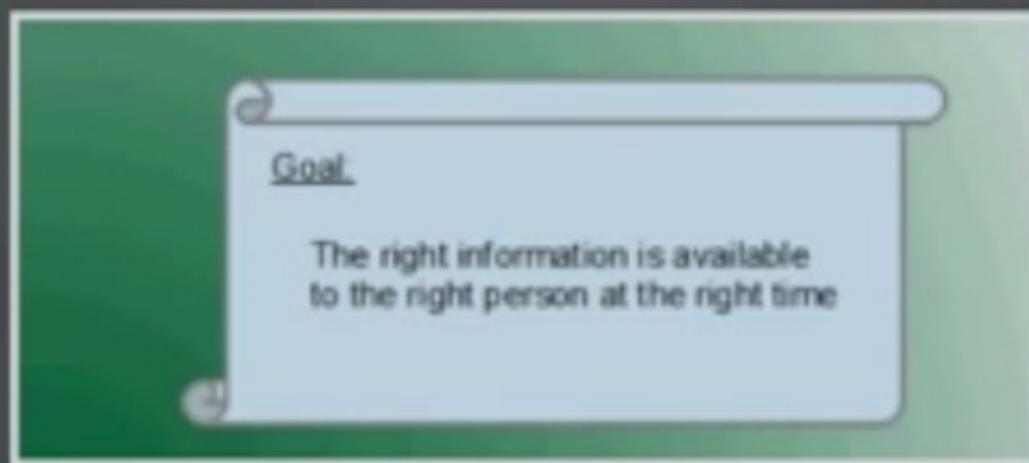
FUNCTIONALITY

An ERP software provides an end to end information management solution for a company. The software could be used by all departments of the company to manage the information.



OBJECTIVE

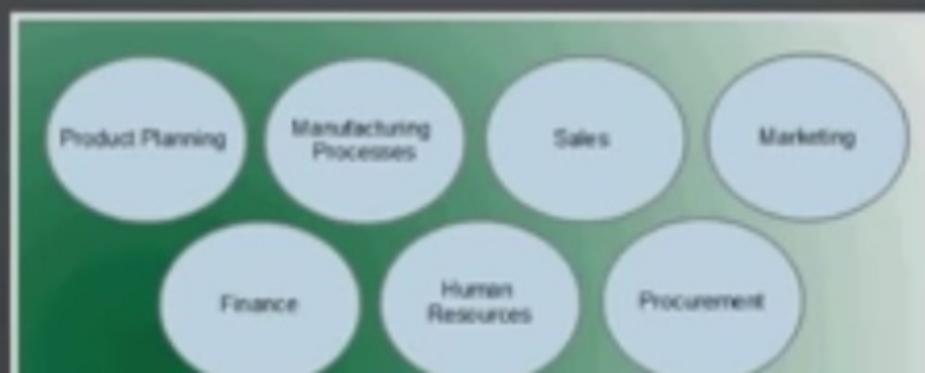
The goal is that right information is available to the right person at the right time.



BUSINESS AREAS

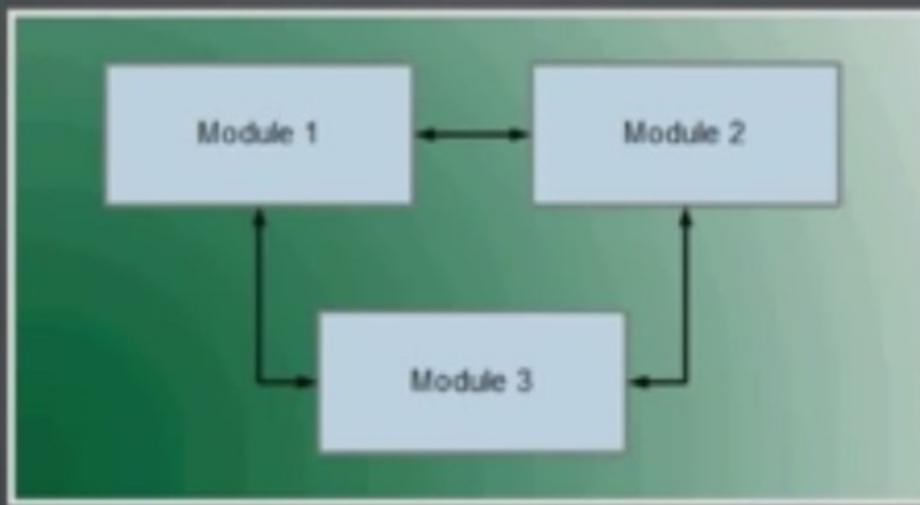
ERP software integrates all areas of operations including:

- Product Planning
- Manufacturing Processes
- Sales
- Marketing
- Finance
- Human Resources
- Procurement



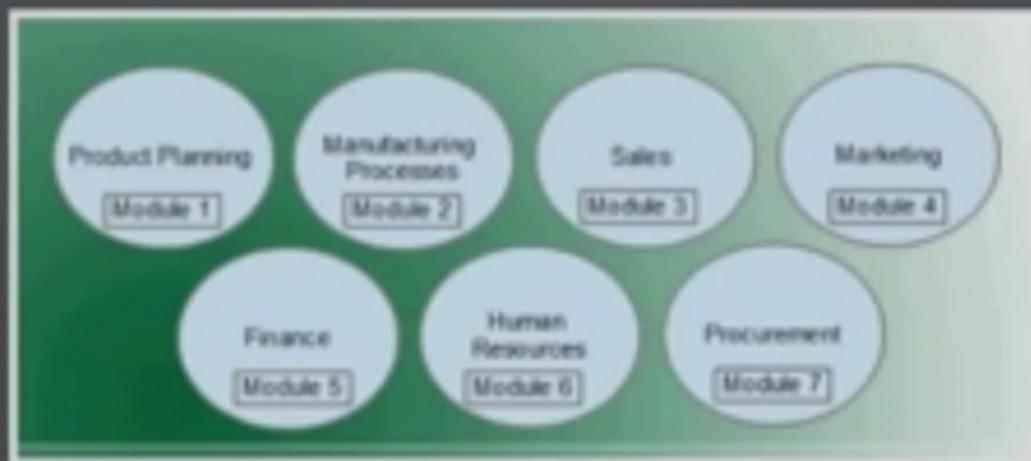
MODULAR AND INTEGRATED

An ERP software is typically modular but integrated. Meaning it consists of multiple modules that are connected to each other.



FOCUS OF A MODULE

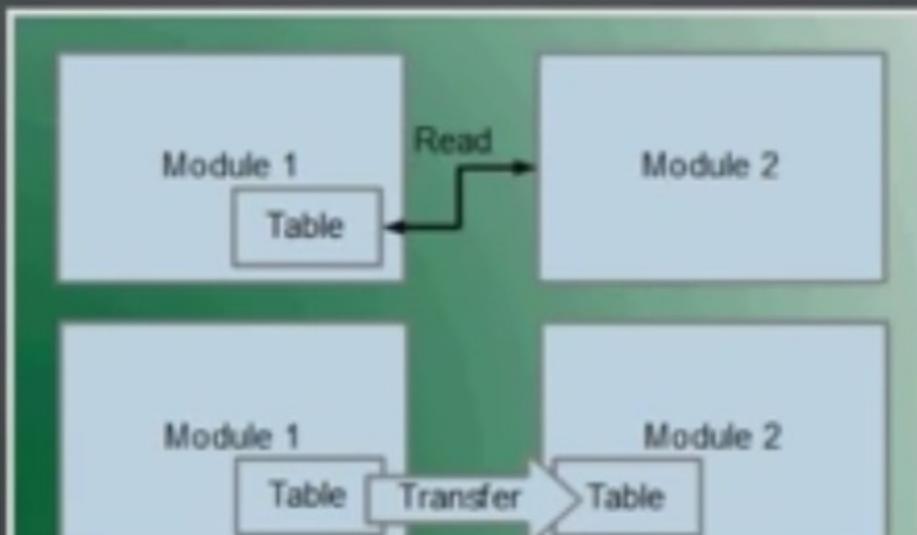
Each module is focused on one area of business processes e.g. finance, human resources etc.



COMMUNICATION AMONG MODULES

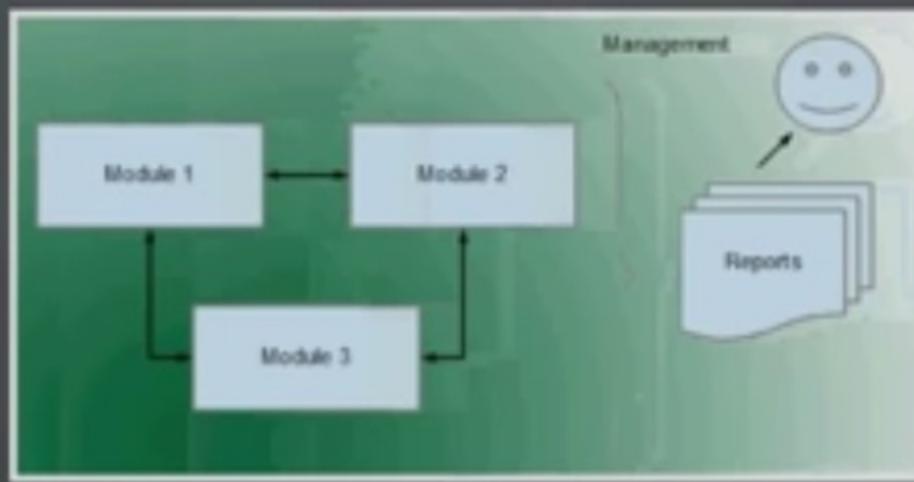
When we say modules are integrated that means:

- A module could share information stored in another module e.g. list of suppliers etc.
- Also information could flow from one module to the other e.g. accounting entries etc.



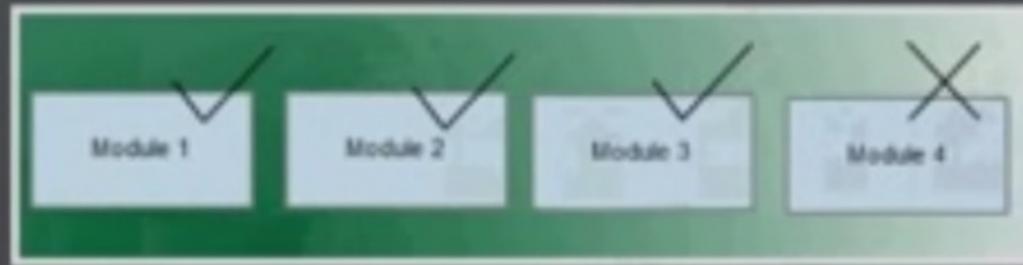
MANAGEMENT REPORTING

Since modules are connected, management of a company could run reports on any aspects of the business to get a complete view of activities. Reports help executives make strategic decisions.



LICENSING

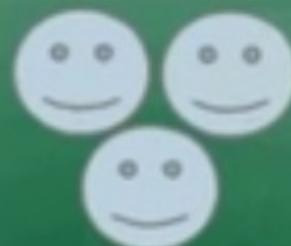
Modules could be individually purchased based on what best meets the specific needs and technical capabilities of the company.



TYPES OF USERS

The end-users of ERP software could be divided into these groups:

- Business users: Performs day to day operations e.g. data entry, operational reports etc
- Management or executives: Run reports and perform inquiries that would help them in decision making



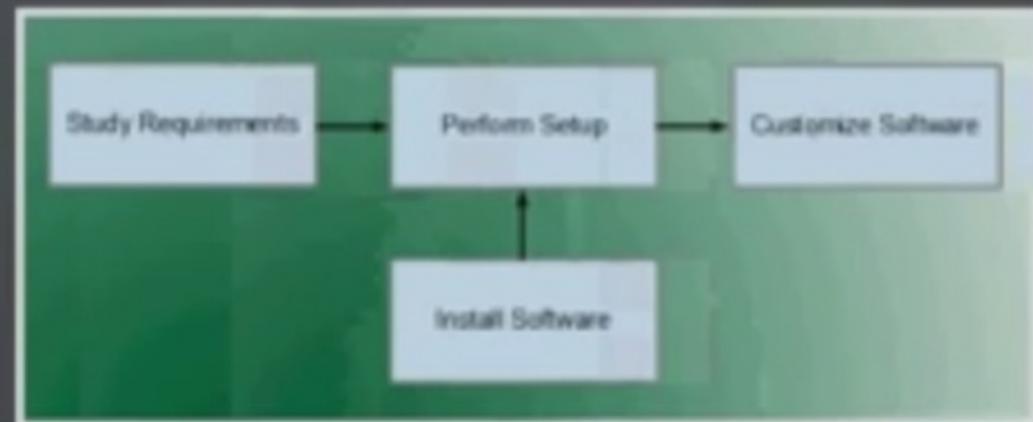
Business Users



Management

IMPLEMENTATION

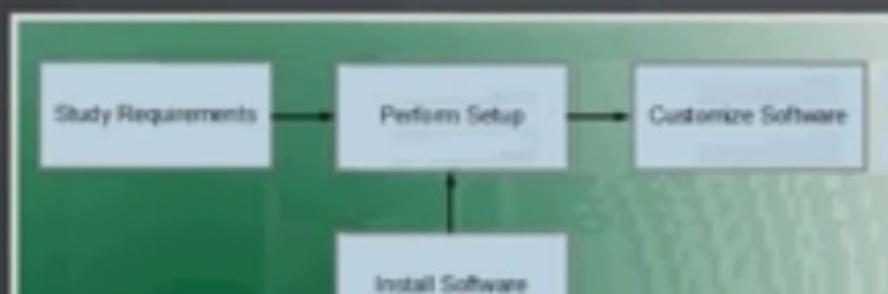
Term "implementation" is used to make the ERP software ready to be used by the company.



IMPLEMENTATION PROCESS

The process involves:

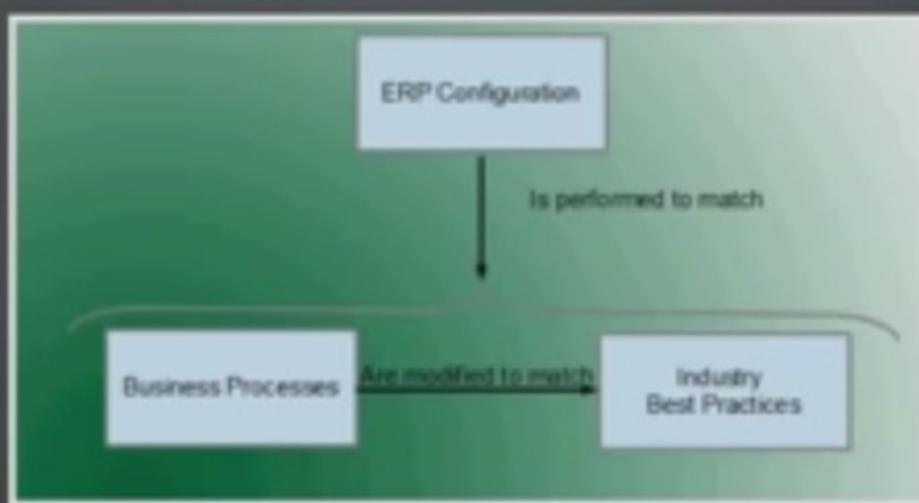
- Study the business requirements. Find out how the ERP system should be behaving.
- Setup or configure the software such that it starts working as per business requirements. By this time the software must be installed and available for setup.
- Fill the gaps between business requirements and the functionality offered by ERP software.



INDUSTRY BEST PRACTICES

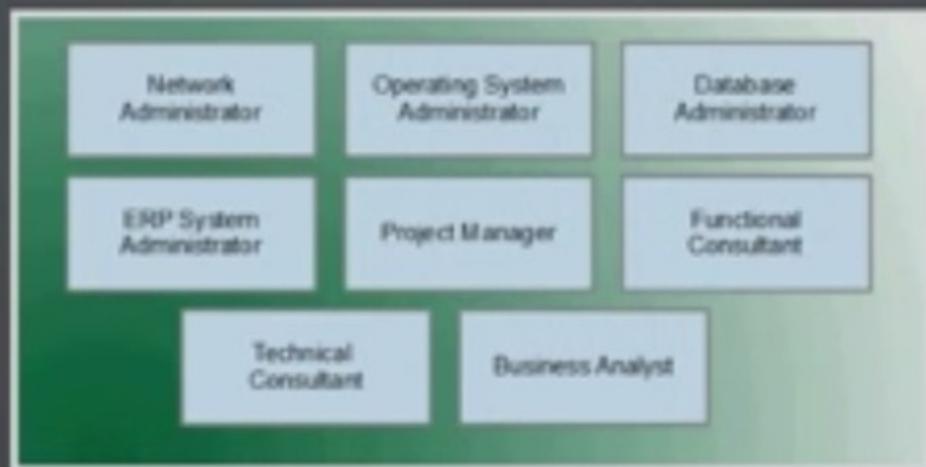
The software is configured to honor company's current business processes. However the company also alters processes where necessary, to bring them align to industry best practices.

Companies do take the implementation of ERP as an opportunity to streamline their business process.



ROLES IN A PROJECT

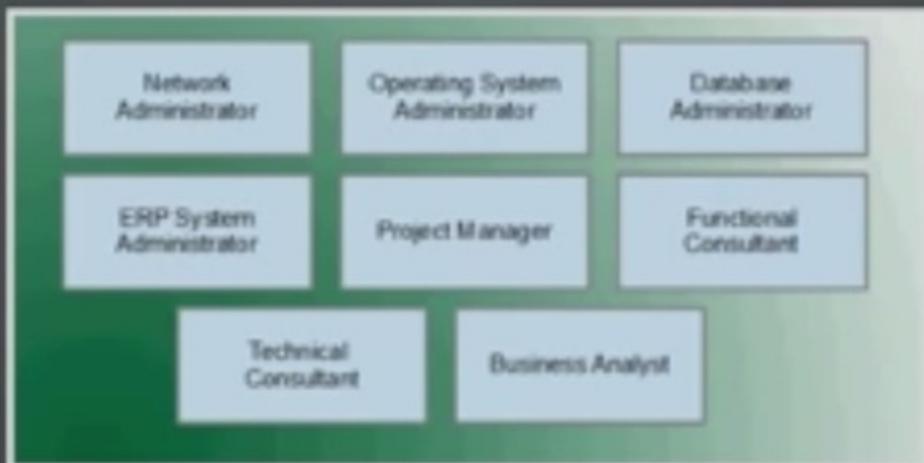
Various roles play part in a typical ERP project. A role may be filled by one or more people depending on the needs. Sometimes one person may be given more than one roles.



ROLES EXAMPLES

Here are important roles:

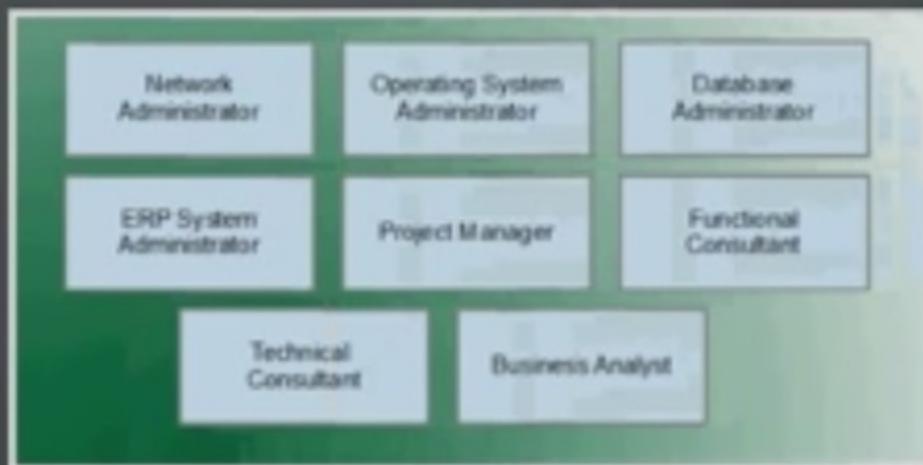
- Network Administrator (Usually 1)
- Operating System Administrator (Usually 1)
- Database Administrator (Usually a team)
- ERP System Administrator (Usually 1)



ROLES EXAMPLES

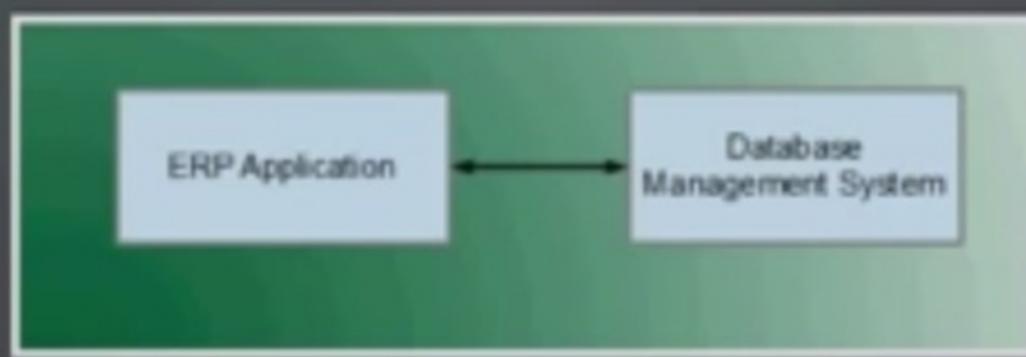
... Continued

- Project Manager (Usually 1)
- Functional Consultant (Usually a team)
- Technical Consultant (Usually a team)
- Business Analyst (Usually 1 per functional area)



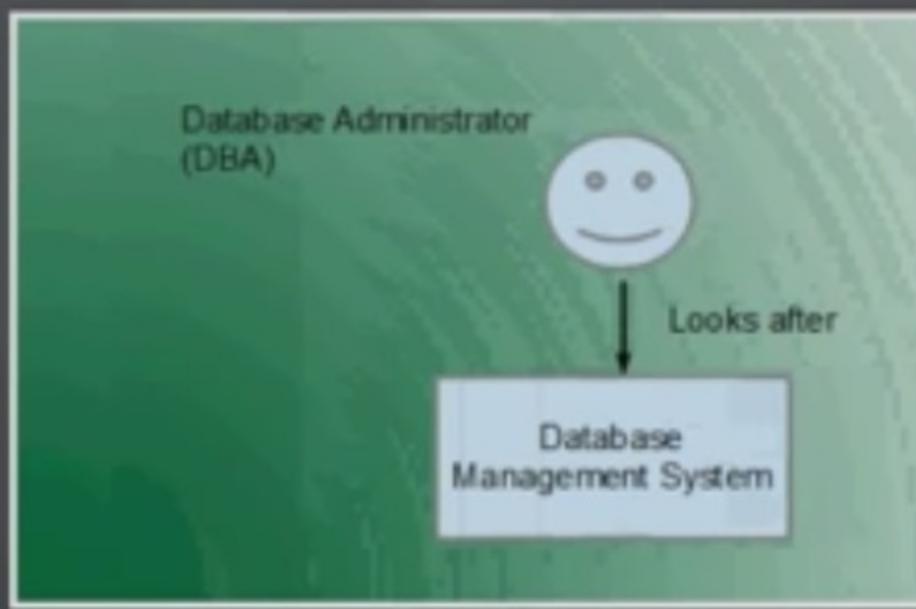
DATABASE MANAGEMENT SYSTEM

ERP software connects to a database software at the back end.
The data is managed in the database.



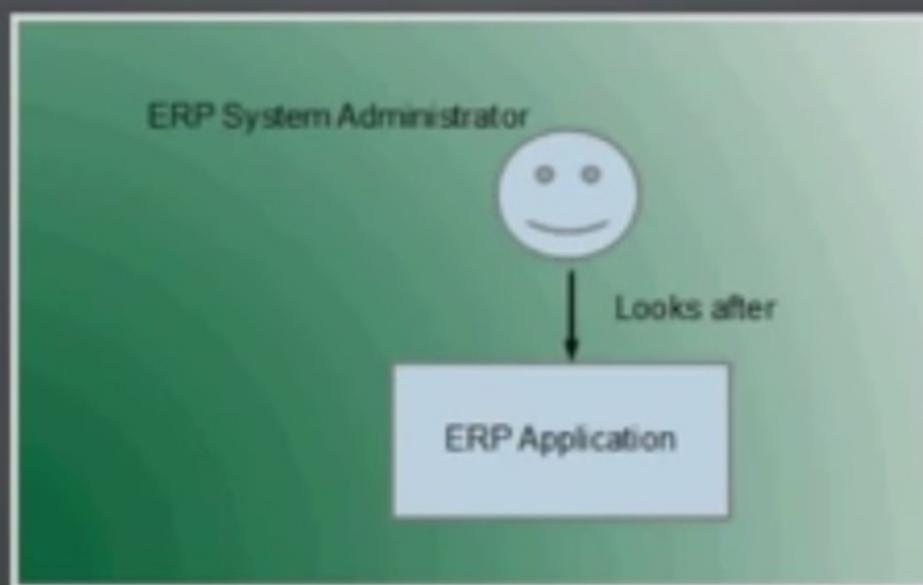
DATABASE ADMINISTRATOR

Database Administrator also known as DBA is the person who looks after the health of the database. He also performs installation of ERP software.



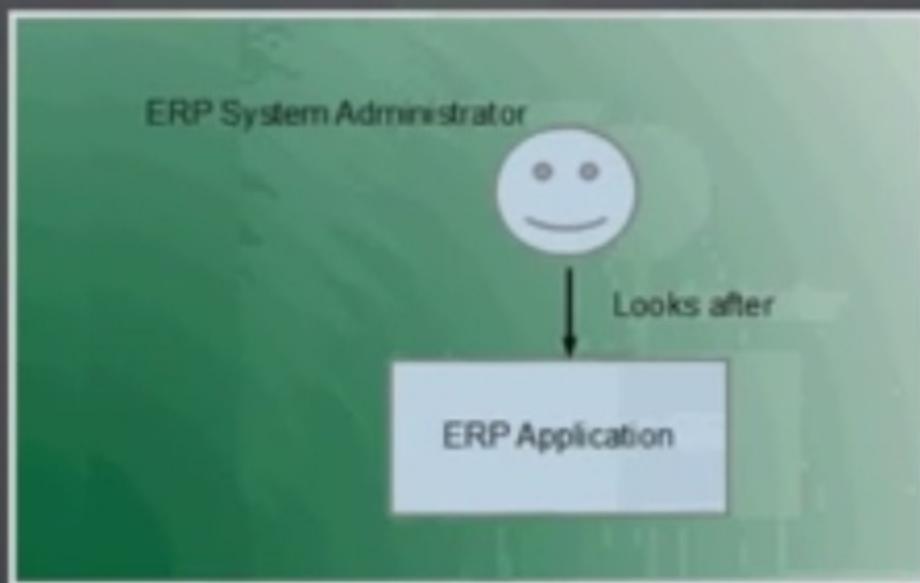
ERP SYSTEM ADMINISTRATOR

ERP System Administrator is the person who looks after the health of the ERP software.



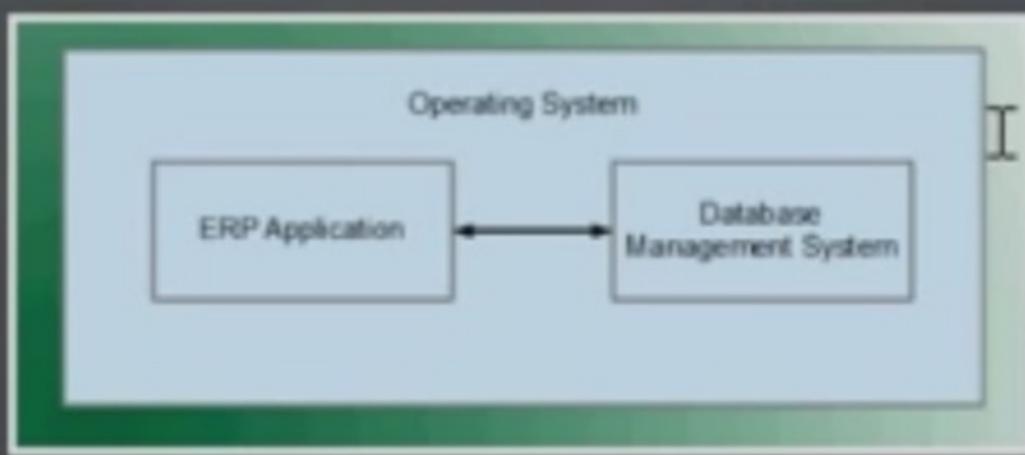
ERP SYSTEM ADMINISTRATOR

ERP System Administrator is the person who looks after the health of the ERP software.



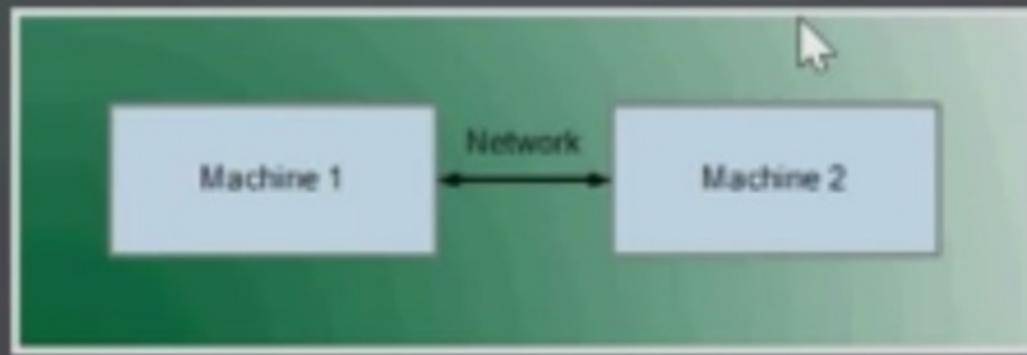
OPERATING SYSTEM

Both database and ERP software runs on an operating system like Linux, Unix, or Windows.



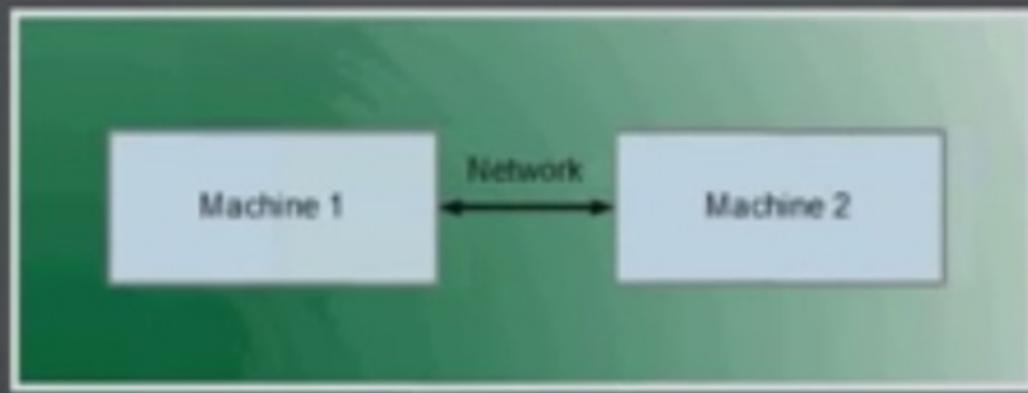
OPERATING SYSTEM ADMINISTRATOR

Operating System Administrator is the person who looks after the health of the operating system.



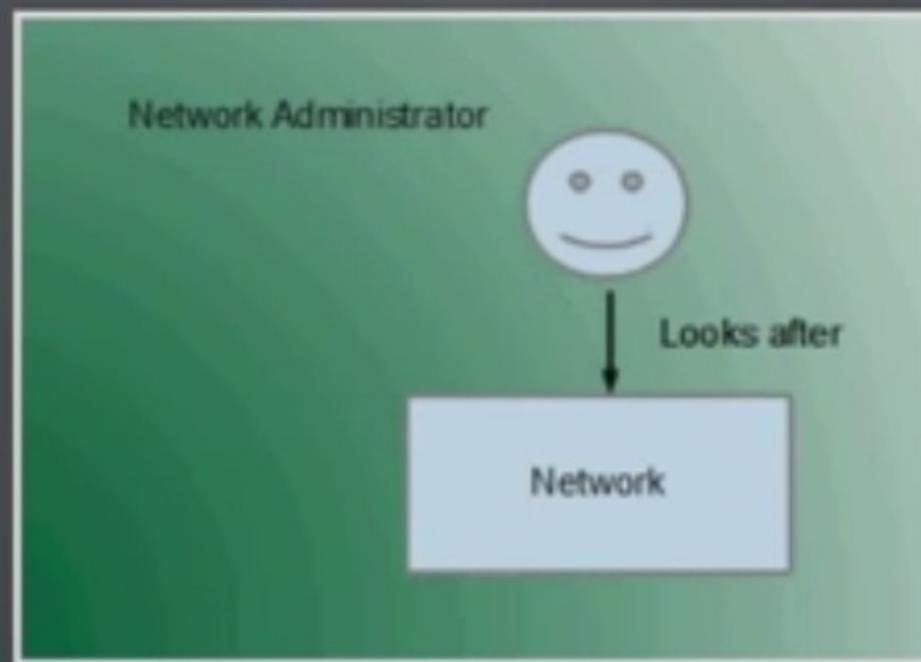
NETWORK

Network connects all the machines together in a system.



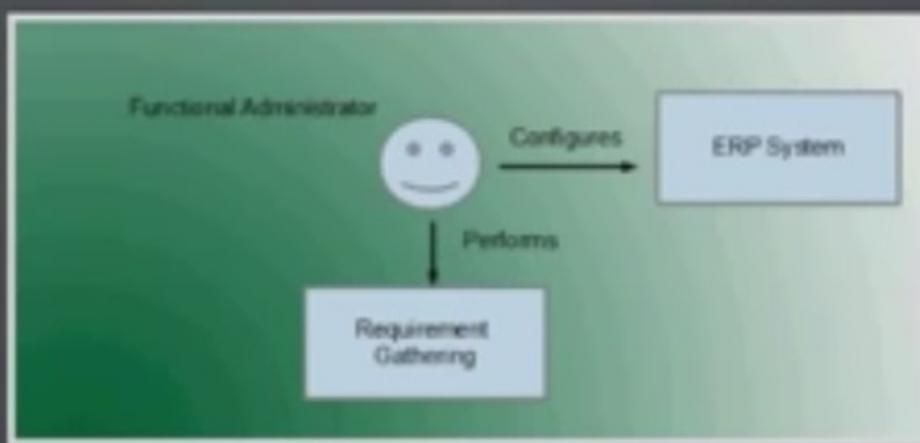
NETWORK ADMINISTRATOR

Network Administrator is the person who looks after the health of the network connecting all the computers together.



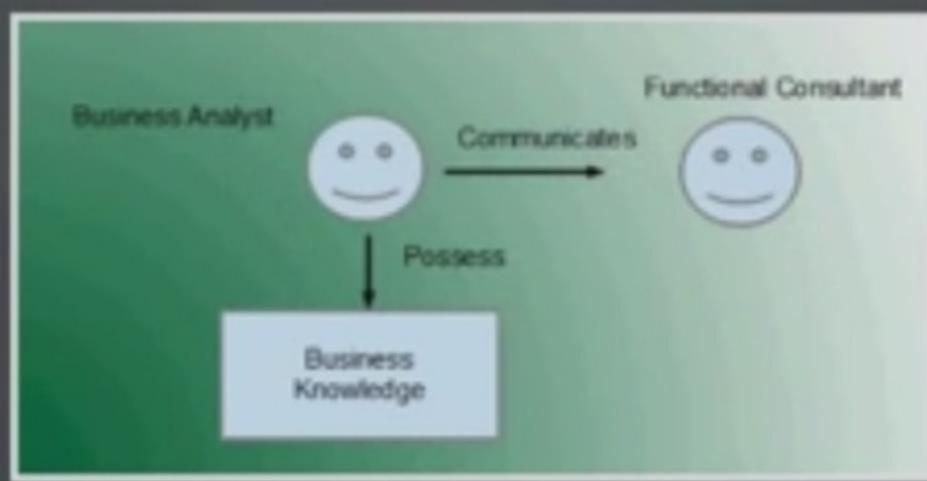
FUNCTIONAL CONSULTANT

Functional Consultant gathers the business requirements and performs ERP setup accordingly.



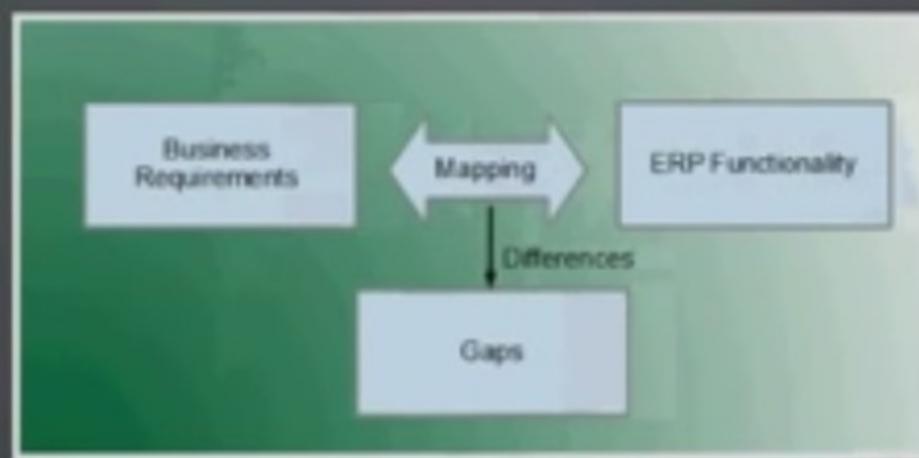
BUSINESS ANALYST

Business Analyst is the person who is an expert of business knowledge. He is in touch with the business users and verifies that the requirements clear to functional consultants.



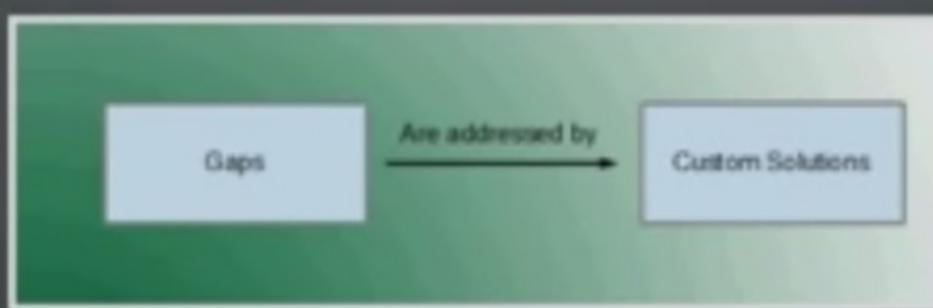
FUNCTIONAL GAPS

In almost all the cases some business requirements are so unique that the ERP system has no built-in functionality to handle those unique cases. The term "Gap" is used for such business requirements.



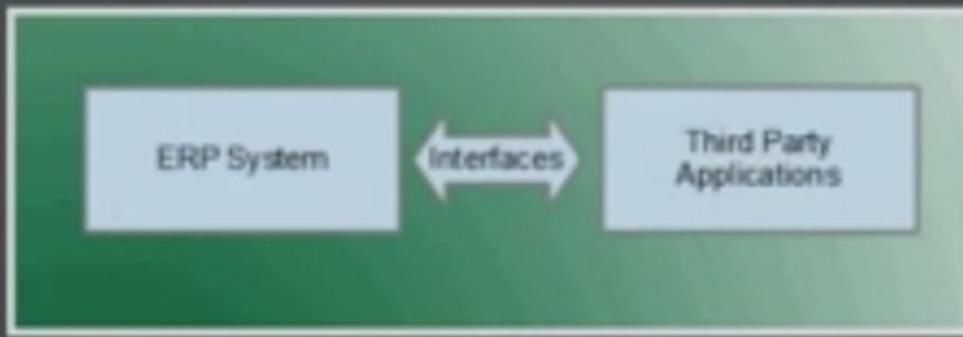
CUSTOMIZATIONS

This is where technical consultants come in. They modify the software by going under the hood and add the missing functionality e.g. new reports are created in the system that were needed by the business. This step is called customization or extension.



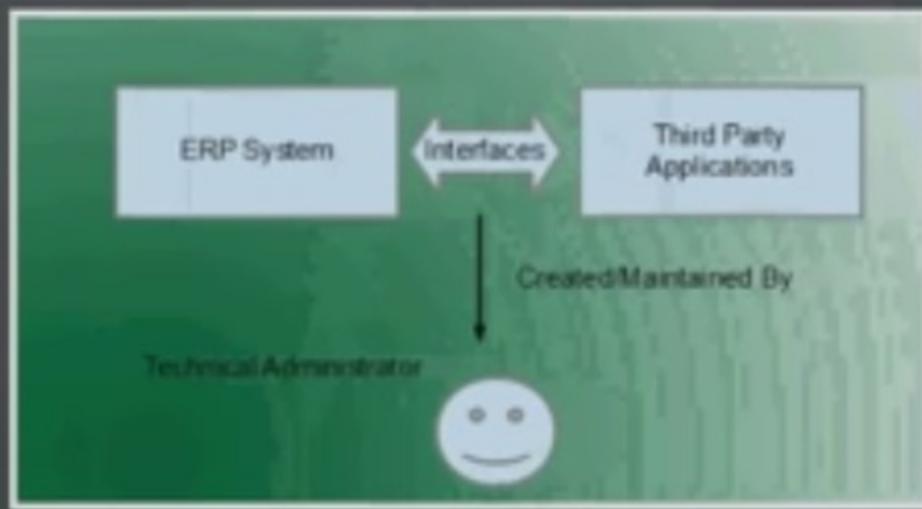
COMMUNICATION AMONG APPLICATIONS

In most cases ERP system talks to other third party system running within the same company or in an external company e.g. suppliers and customers.



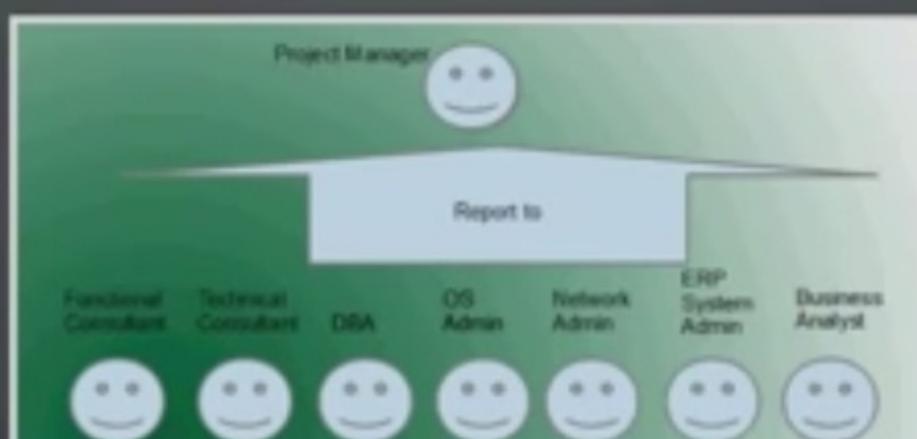
INTERFACES

Technical consultants assist in writing programs that help communicate information back and forth between the ERP software and the third party software either within the company or outside it. Programs that aid communication between two software is called "interface".



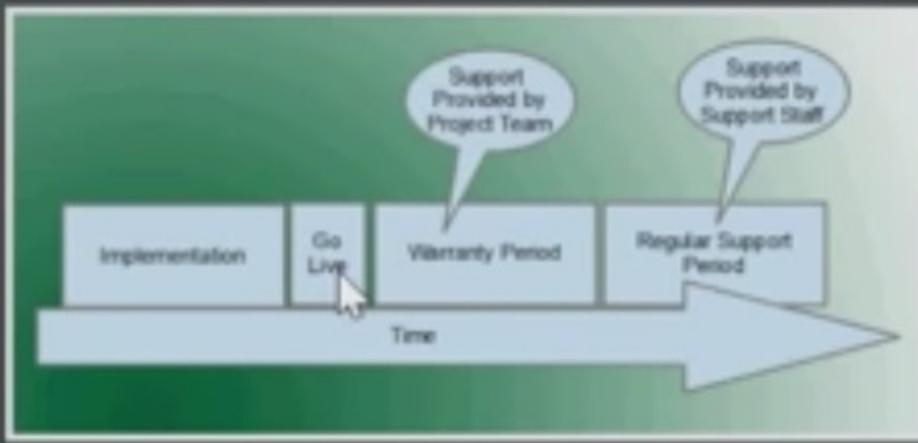
PROJECT MANAGER

Project manager is the person who is managing the project. All other team members report to the project manager during the project. Most members also report to their regular bosses as well. For example, a database administrators will be reporting to the manager or director of the IT department as well as to the project manager during the life of the project.



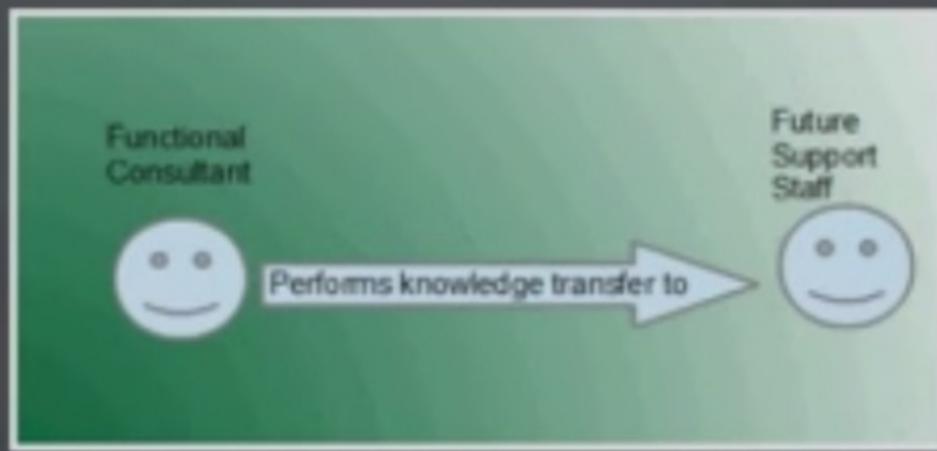
WARRANTY AND SUPPORT PERIODS

After the go-live, warranty period begins. Any problems that come up will be handled by the implementation team. After that the support role will be transferred to another team, usually permanent staffs or a third party company specialized in support.



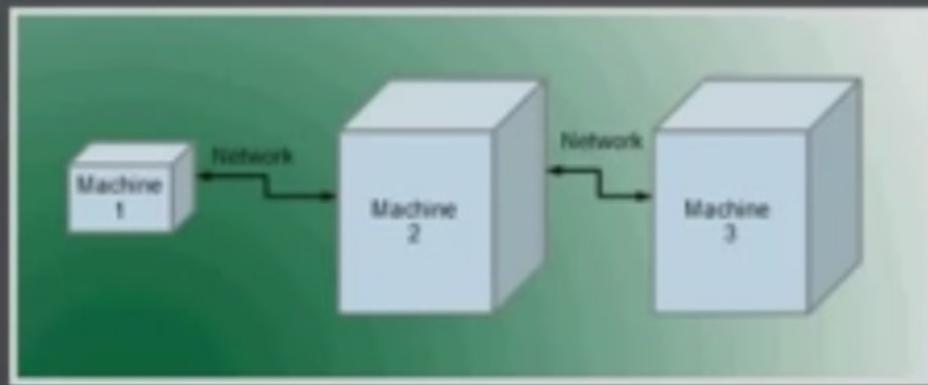
TRAINING

Consultants would provide training to the new person or team who will be providing support.



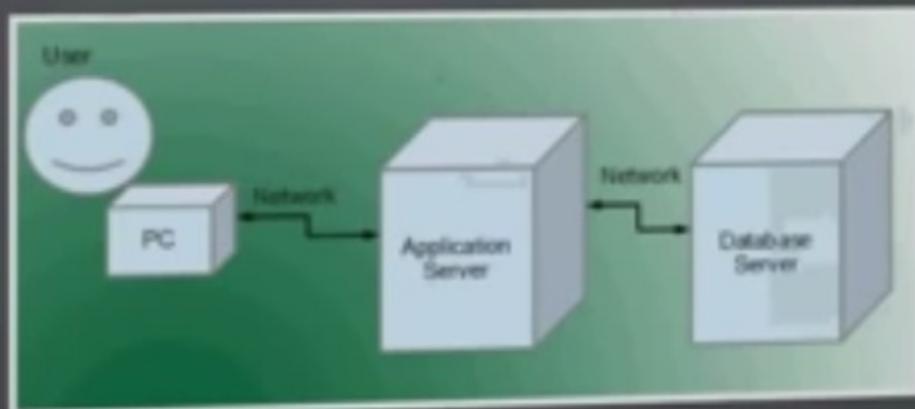
MULTIPLE SERVERS

Usually there are few computers, called servers, that are connected and work together to produce workable environment for ERP systems.



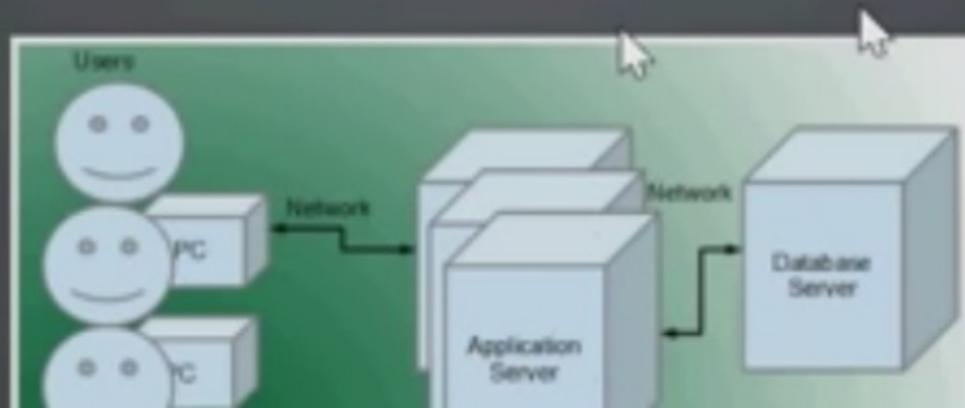
THREE TIER ENVIRONMENT

One computer hosts the application (the actual ERP software) and another one hosts the database. Users connect to the computer running ERP System through their browsers. This is called a three tier environment.



LOAD BALANCING

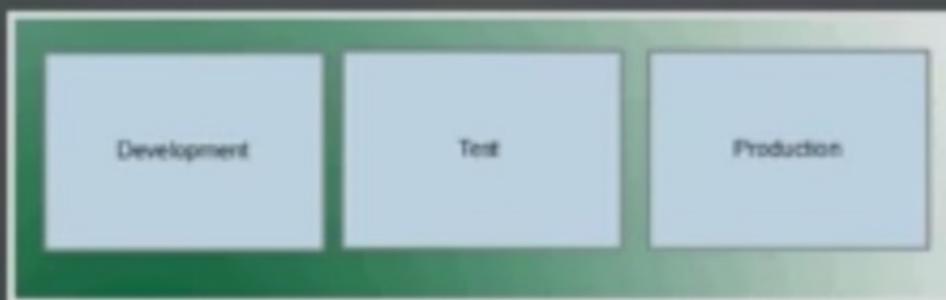
In an environment where numerous users connect, there could be even multiple application servers installed, communicating with the same database server. This way the load is shared among various machines. The term used for this setup is "Load Balancing".



ENVIRONMENTS

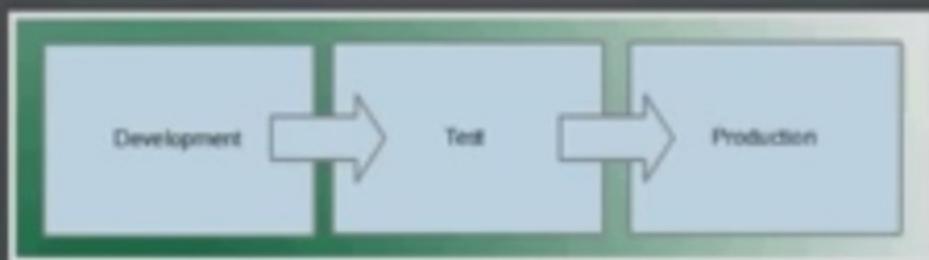
Companies running ERP systems usually keep three sets of environments:

- Development
- Test
- Production



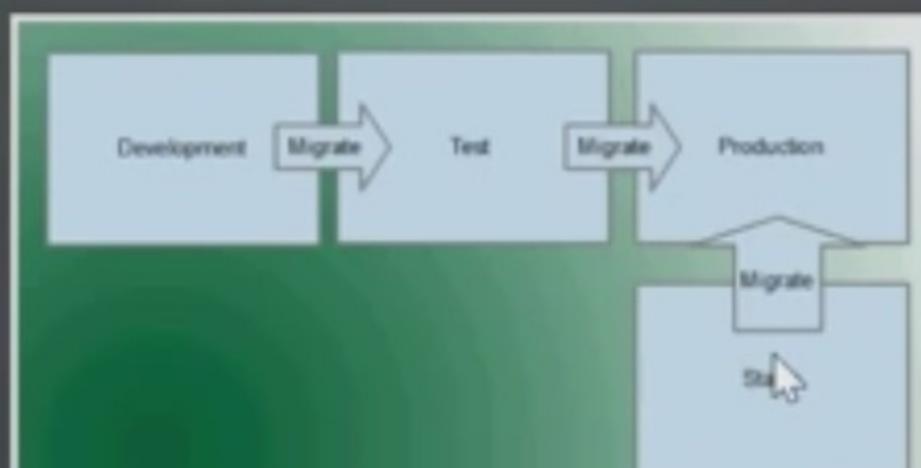
TRANSPORTING SETUP

The configuration is performed in the development environment first by the team. The setup is copied over to the Test environment where users perform their testing. Once users are happy the setup is copied to the production (Go-live), where the system is actually used to manage day to day operations of the company. The term used for copying configuration from one environment to the other is "Transport".



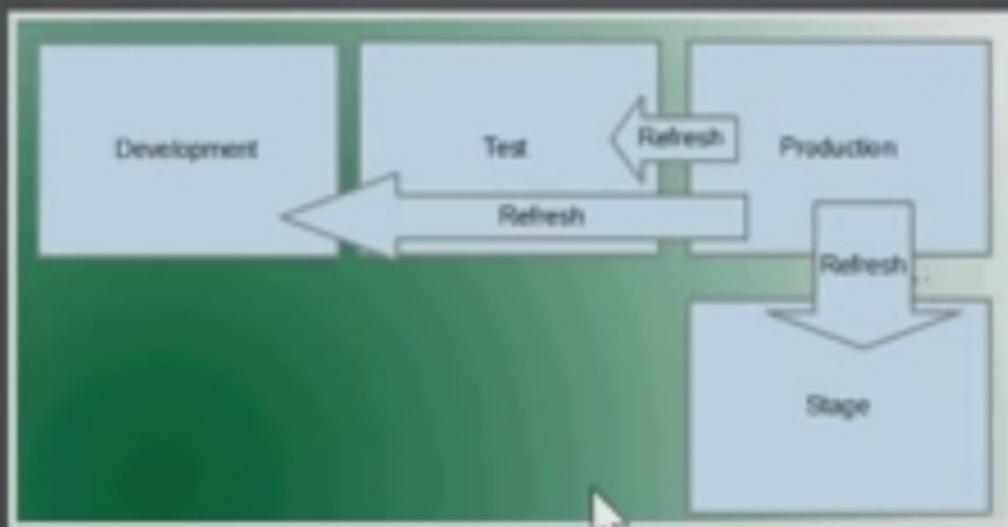
STAGE ENVIRONMENT

In some companies few other environments are also maintained
e.g. "Stage" where troubleshooting is performed, and bug
fixes/patches are applied and tested first before moving them to
production.



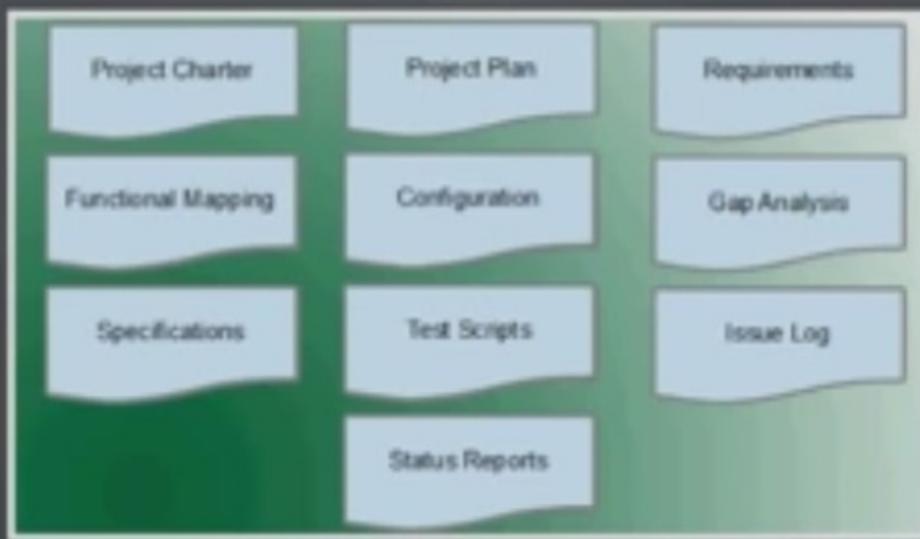
REFRESH

All environments are periodically refreshed with recent production data.



PROJECT DOCUMENTATIONS

Documents are created through the project and are stored in the central place for the project.



EXAMPLES OF DOCUMENTS

Some examples are:

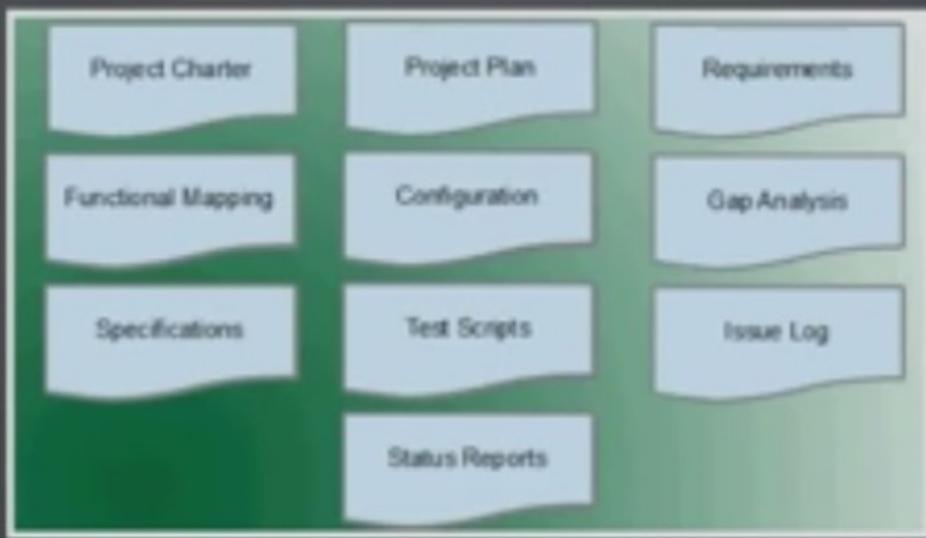
- Project Charter by Project Manager
- Project Plan by Project Manager
- Requirement Gathering document by Functional Consultant



EXAMPLES OF DOCUMENTS

... Continued

- Functional Mapping by Functional Consultant
- Configuration document by Functional Consultant
- Gap Analysis document by Functional Consultant



EXAMPLES OF DOCUMENTS

... Continued

- Functional/Technical Specifications by Technical Consultant
- Test Scripts by Functional Consultant/Business Analyst
- Issue log by Project Manager
- Period Status Reports by all members



PROJECT CHARTER

Document: Project Charter by Project Manager

Contents: What is involved in the project from a high level perspective, which business needs are behind this project, what benefits it will bring, who is sponsoring the project, what are the risks, what are dependencies and assumptions etc.

Project Charter

PROJECT PLAN

Document: Project Plan by Project Manager

Contents: Tasks that will take place during the project.

Project Plan

REQUIREMENT GATHERING

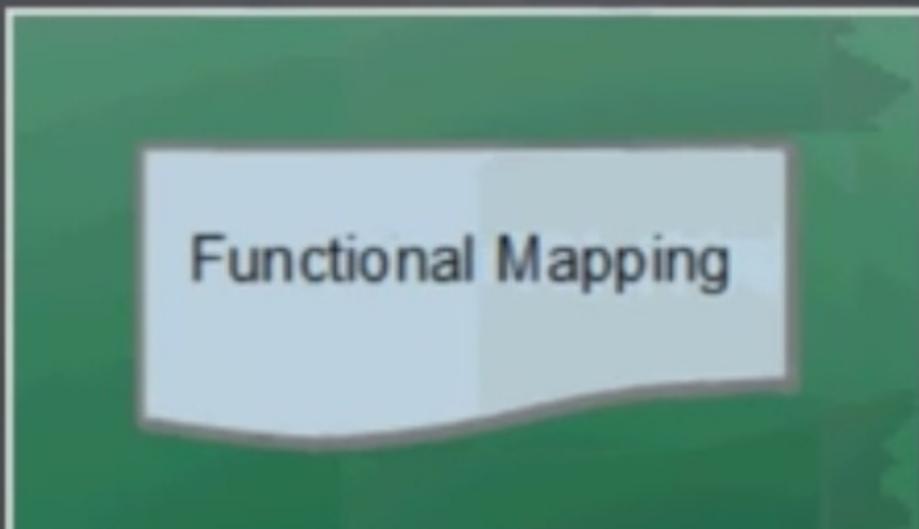
Document: Requirement Gathering document by Functional Consultant

Contents: How the new system should be implemented.

Requirements

FUNCTIONAL MAPPING

Document: Functional Mapping by Functional Consultant
Contents: Mapping of requirements to the ERP software features; Which features should be enabled in ERP.



Functional Mapping

SYSTEM CONFIGURATION

Document: Configuration document by Functional Consultant

Contents: How the new system will be configured so that the required features are available.

Configuration

GAP ANALYSIS

Document: Gap Analysis document by Functional Consultant
Contents: Which requirements are not matched to any of the available features in ERP.

Gap Analysis

SPECIFICATIONS

Document: Functional/Technnical Specifications by Technical
Consultant

Contents: How custom developed features will work.



Specifications

TEST SCRIPTS

Document: Test Scripts by Functional Consultant/Business Analyst

Contents: What steps users will perform during testing. Test Results are also captured by the Business in the Test Scripts document.

Test Scripts

ISSUE LOG

Document: Issue log by Project Manager

Contents: What problems came up and how they were handled

Issue Log

STATUS REPORTS

Document: Period Status Reports by All

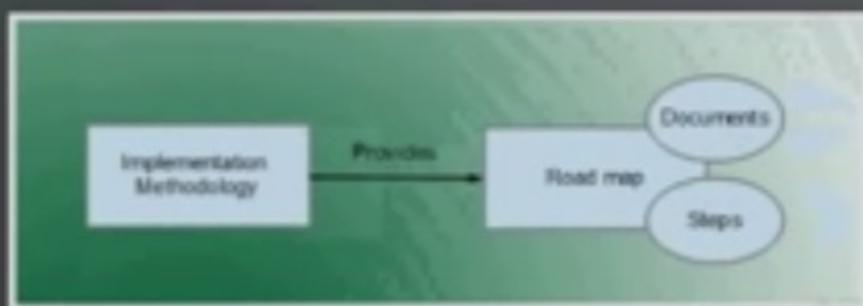
Contents: What was completed last week and what is due next week

Status Reports

IMPLEMENTATION METHODOLOGY

The project teams usually follows an implementation methodology which provides a roadmap through the project. A methodology dictates:

- Which documents will be created at which point
- How the documents will look like (Templates are provided)
- What will be the sequence of configuration tasks.



STATUS REPORTS

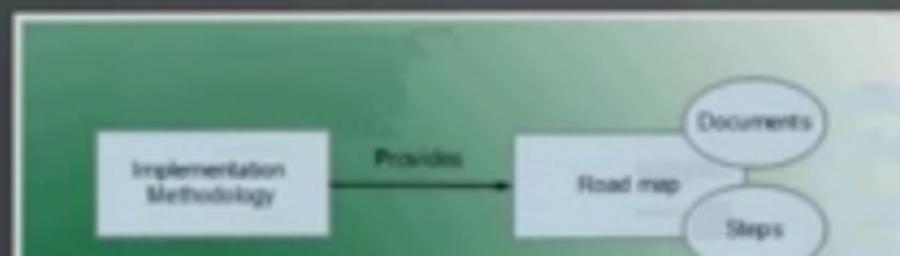
Document: Period Status Reports by All
Contents: What was completed last week and what is due next week

Status Reports

IMPLEMENTATION METHODOLOGY

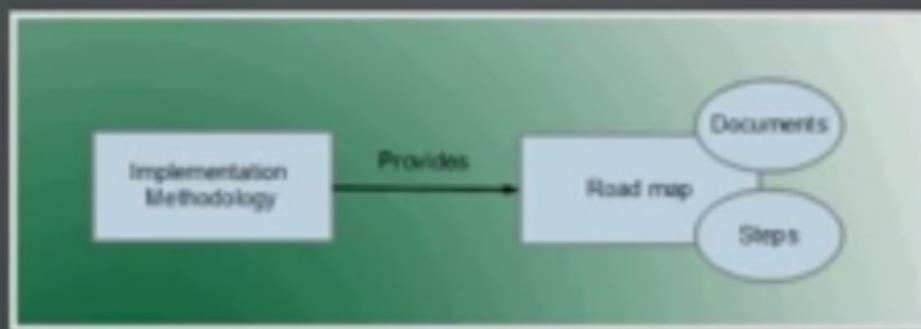
The project teams usually follows an implementation methodology which provides a roadmap through the project. A methodology dictates:

- Which documents will be created at which point
- How the documents will look like (Templates are provided)
- What will be the sequence of configuration tasks.



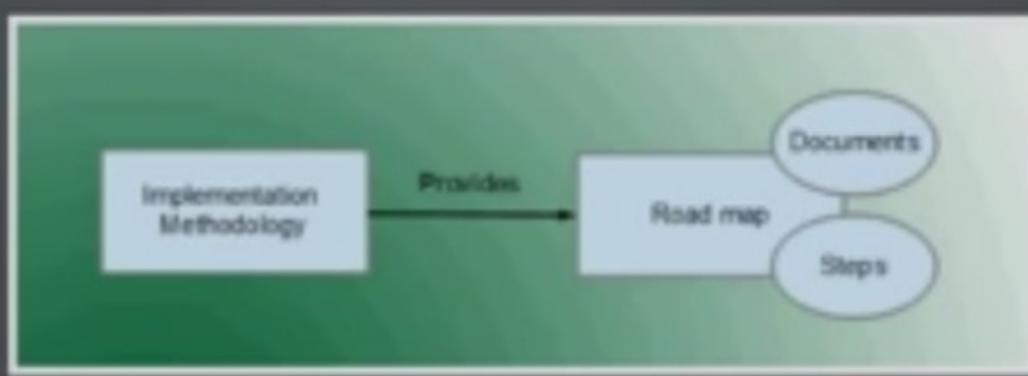
IMPLEMENTATION METHODOLOGY AND PROJECT PLAN

The steps are incorporated in desired sequence, as dictated by the methodology, in the overall project plan managed by project manager.



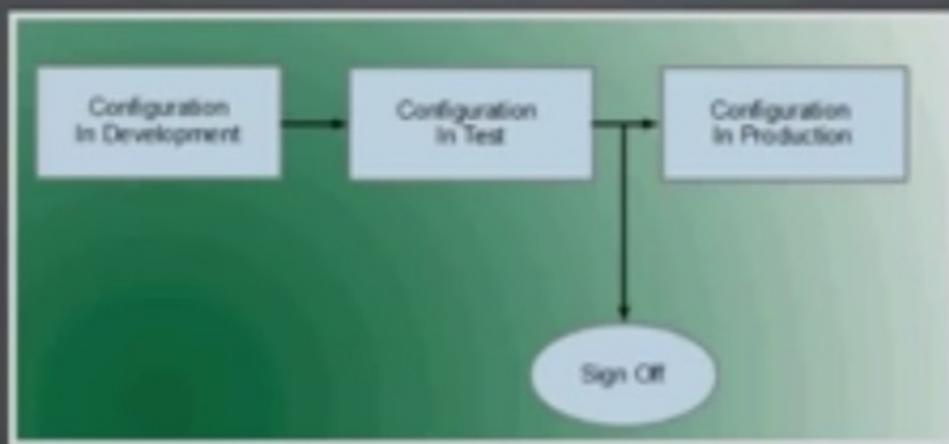
PROPRIETARY METHODOLOGIES

Large and reputed consultings firms usually have their own proprietary methodology that they follow.



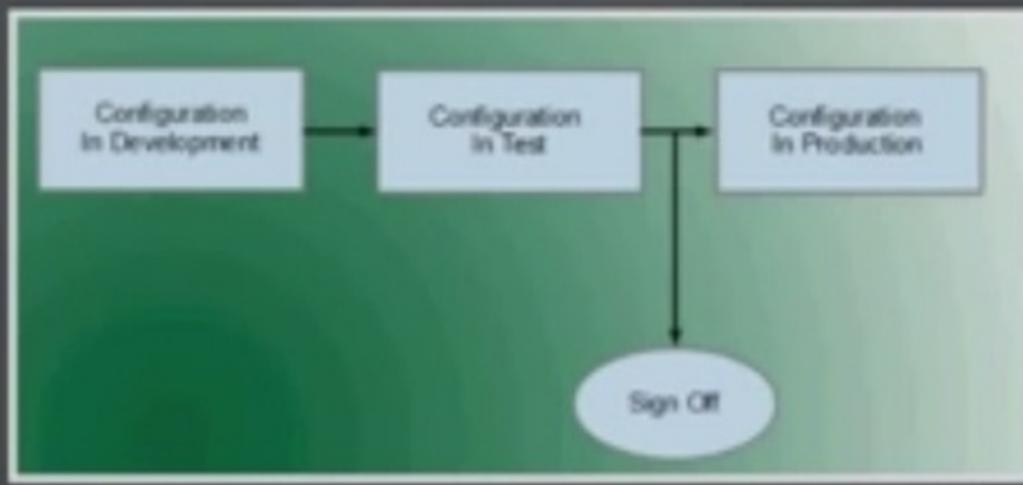
SIGN-OFF

Written sign-offs are required from the business at various stages e.g. before moving configuration from Test to Production.



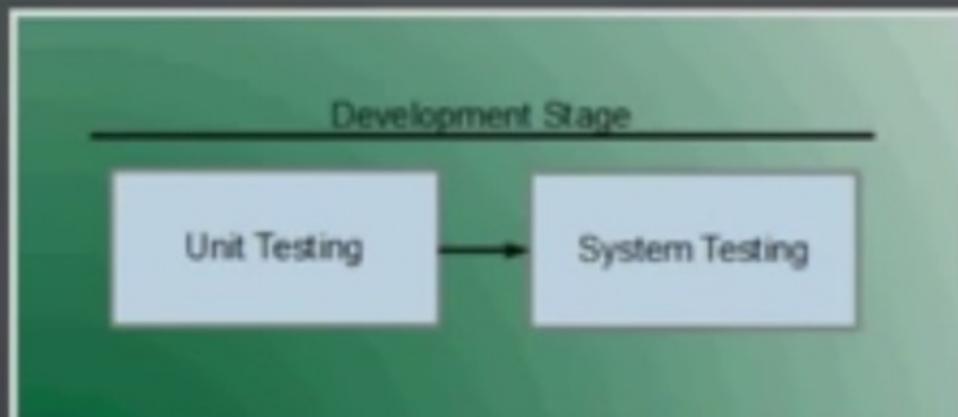
SIGN-OFF

Written sign-offs are required from the business at various stages e.g. before moving configuration from Test to Production



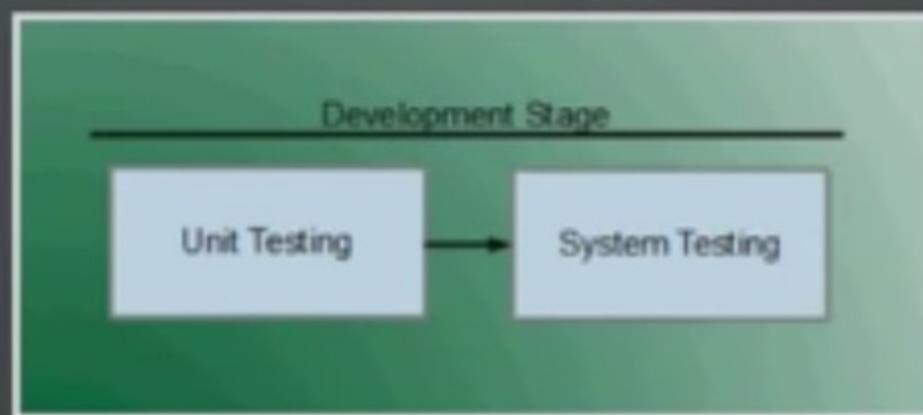
UNIT TESTING

Consultants perform unit testing and system testing during development stage. Unit testing refers to testing of one module (or unit) individually. The focus is on the functionality of the modules.



SYSTEM TESTING

System testing refers to the testing of the whole system (all modules together). The focus is on the integrity of the system as a whole e.g. if modules are communicating with each other properly. The testings are performed on development environment, however a separate test environment could be requested as well for this purpose.



USER ACCEPTANCE TESTING

User Acceptance Testing (UAT) is a mandatory testing. Here users drive the system. The consultants train and guide the users. The focus is to verify whatever was promised, is delivered or not. At the end of the testing users must provide a sign-off. All outstanding problems need to be fixed. The testing takes place in Test environment dedicated for this purpose.

