



RH Consulting, Inc.

Getting the most of your IT investment

Architecting and Developing Modern Systems

As of September 6, 2009



Session Objectives

- Key Architectural Concepts
 - Service Oriented Architecture (SOA)
 - Event Oriented Architecture (EDA)
 - Enterprise Service Bus (ESB)
- Tools and Programming Languages
- Developing and Architecture Plan

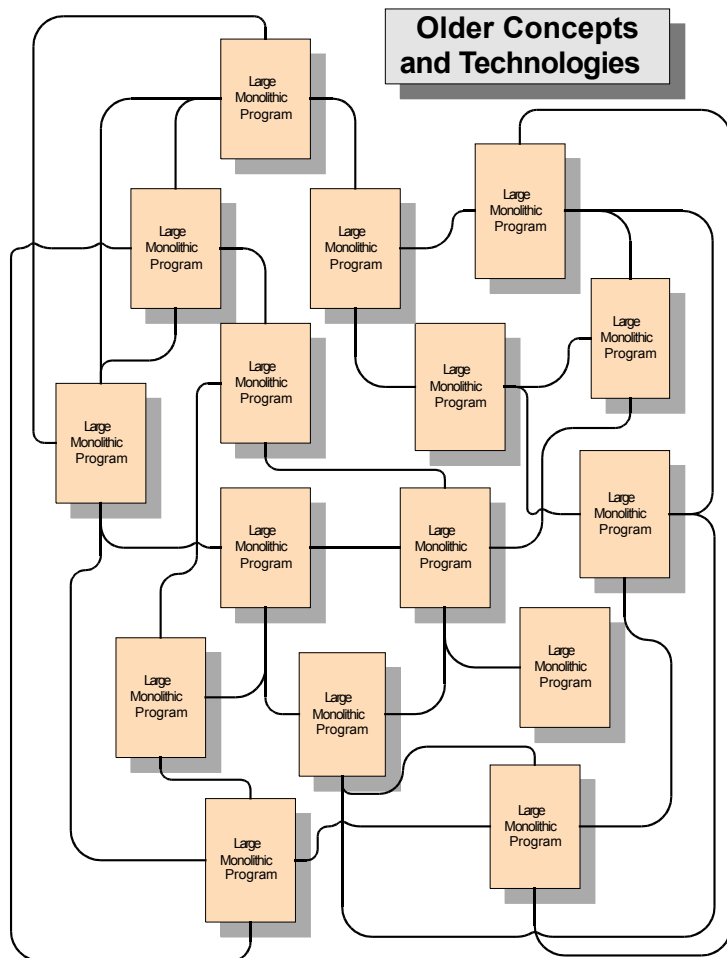




Key Architectural Concepts

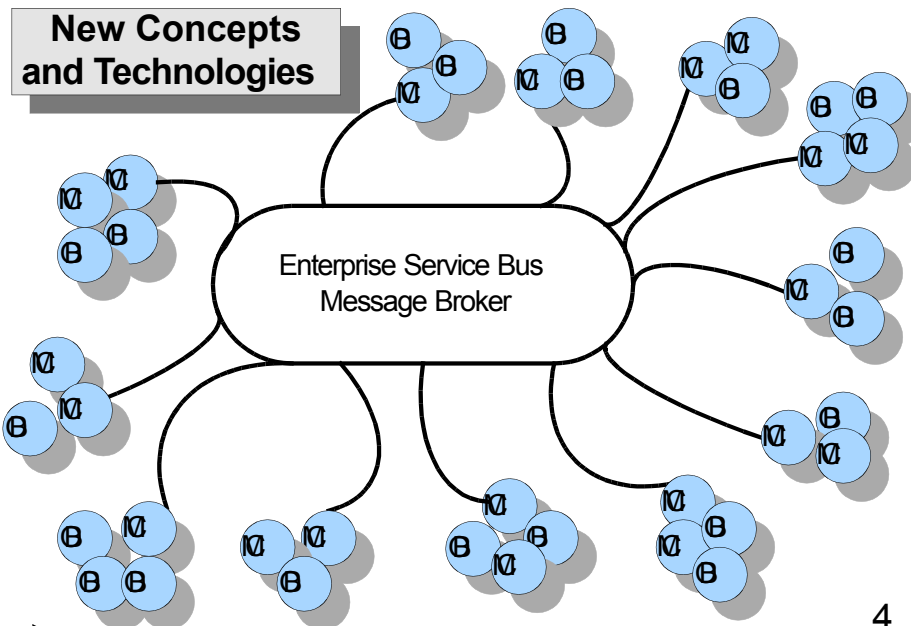


Key Concepts – Old vs. New Architecture



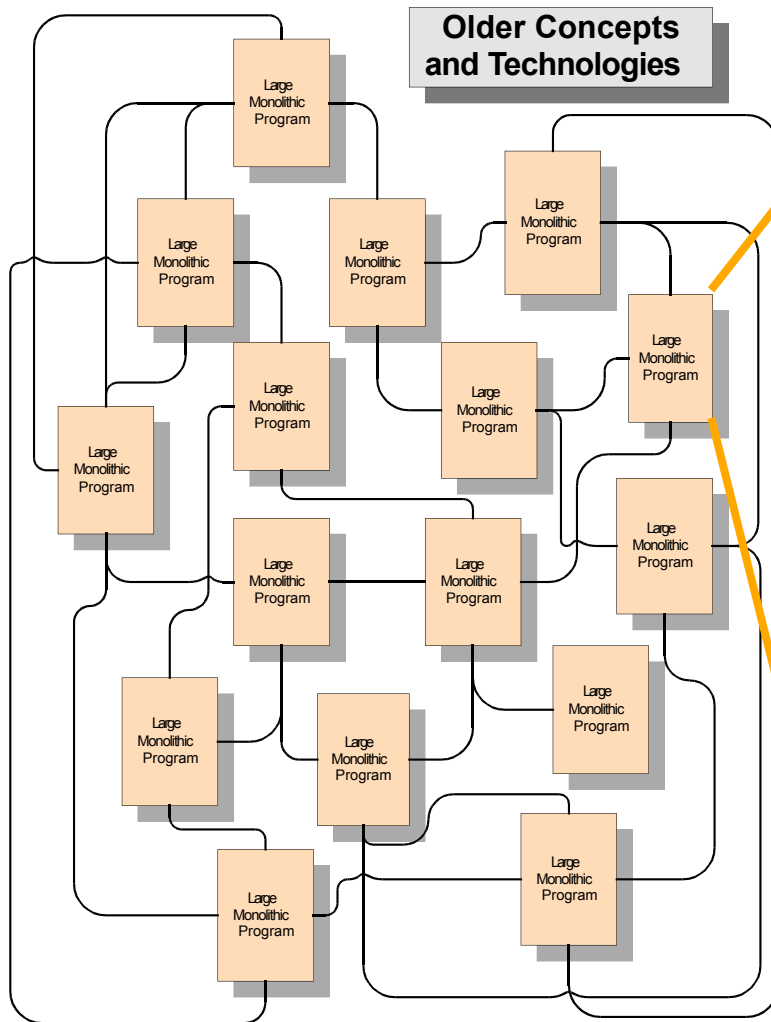
From – Large monolithic (Webster's: something made of a single large block) programs, that are tightly coupled, use a wide variety of technologies, and have a high degree of complexity.

To – Small single task components with loosely connected endpoints, standard set of technologies, and a lower level of complexity.



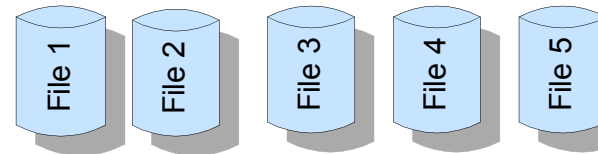


Key Concepts – Old Architecture

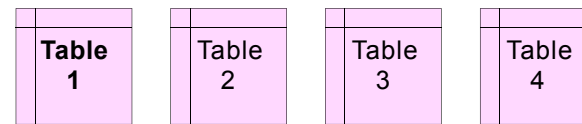


Monolithic Pgm CA 2240

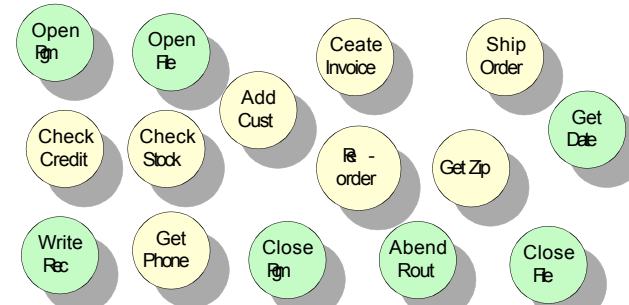
File Section



Working Storage Section



Procedure Devision



Static or dynamic linkage (a major advantage in its time)

Key Concepts

Old Architecture - Characteristics and Challenges



Characteristics

- Programs tend to be large, complex and are developed independently
- Business rules tend to be embedded in the program logic
- Logic is often duplicated in multiple programs
- Applications tend to be platform specific
- Program and vendor interfaces are often proprietary
- There is heavy dependence on older languages such as Cobol and Assembler
- There is heavy dependence on 3270 display screens
- There is still heavy use of SNA for transmitting files
- There is still heavy use of VSAM for data files
- Development tends to be waterfall in nature
- Client, middleware, and back-end logic is often disbursed and repeated
- Applications tend to be designed, built and maintained as silos with little componentization
- Controls are manual depending on emails, lacking automated checks

Key Concepts

Old Architecture - Characteristics and Challenges



Challenges

- Developers must be experienced in many disciplines and specialize in a specific enterprise functional layers
- Current systems and approach is very dependent upon SMEs
- Maintenance is difficult and time consuming
- Changes have to be made in multiple places
- Requires a significant amount of time to bring products to market
- Changes and or modifications generally require every layer to be tested
- Governance tends to be manual and labor intensive



Key Concepts - Terminology

Enterprise – Having an entire company perspective, rather than a view of just a specific application, project, business area, business function, platform, data-store, etc.

Service – noun – 1) A task that is performed (making of coffee), 2) the component that performs the task (the coffee maker). Gartner prefers to call the component a “Service Implementation”, but it is common to call it a “Service”. Sometimes these services take on the role of a “consumer” and sometimes the take on the role of a “provider”.

- **Consumer** – is a service that requests work to be performed by another service.
- **Provider** – is the service that performs the work that is requested. Note: when a component that was a provider calls a third component, it is viewed as a consumer to that third component.

Messages and Messaging

- A message is the information passed from one service to another
- Messaging is the process of sending messages between the consumer and the provider

Distributed Systems – multiple program applications, that can run on different computers connected by a network. In a simple case the modules may run on the same computer. This includes applications that have components or services that run on the mainframe.

Key Concepts - Terminology



Middleware – Runtime system software that provides infrastructure support functions for the business services. It removes the burden of tasks like security, protocols, and locating data from the service developer. It helps to ensure programs and databases running on different platforms work together in an efficient manner. Middleware can be sub-divided into data management, communications, and platform middleware. It can be either purchased or in house developed.

Common Modules – Common functions placed in sharable objects. These functions would otherwise have to be handled by the business function programs. Developers are now relieved of that duty and specialists in a common area provide these services. Common modules are classified as “Core”, “Enabling”, and “Business”. Examples of these modules may be date validation or standard transaction logging. Common modules can run in a middleware environment.

Frameworks – The definition of how common functional areas, that are needed by most applications (e.g. application security, transaction management, data access) should be handled across the enterprise. Each framework provides a description of and specifications for the related group of common functions. It includes the standard, guidelines, guiding principles and/or sample code..

Key Concepts - Terminology

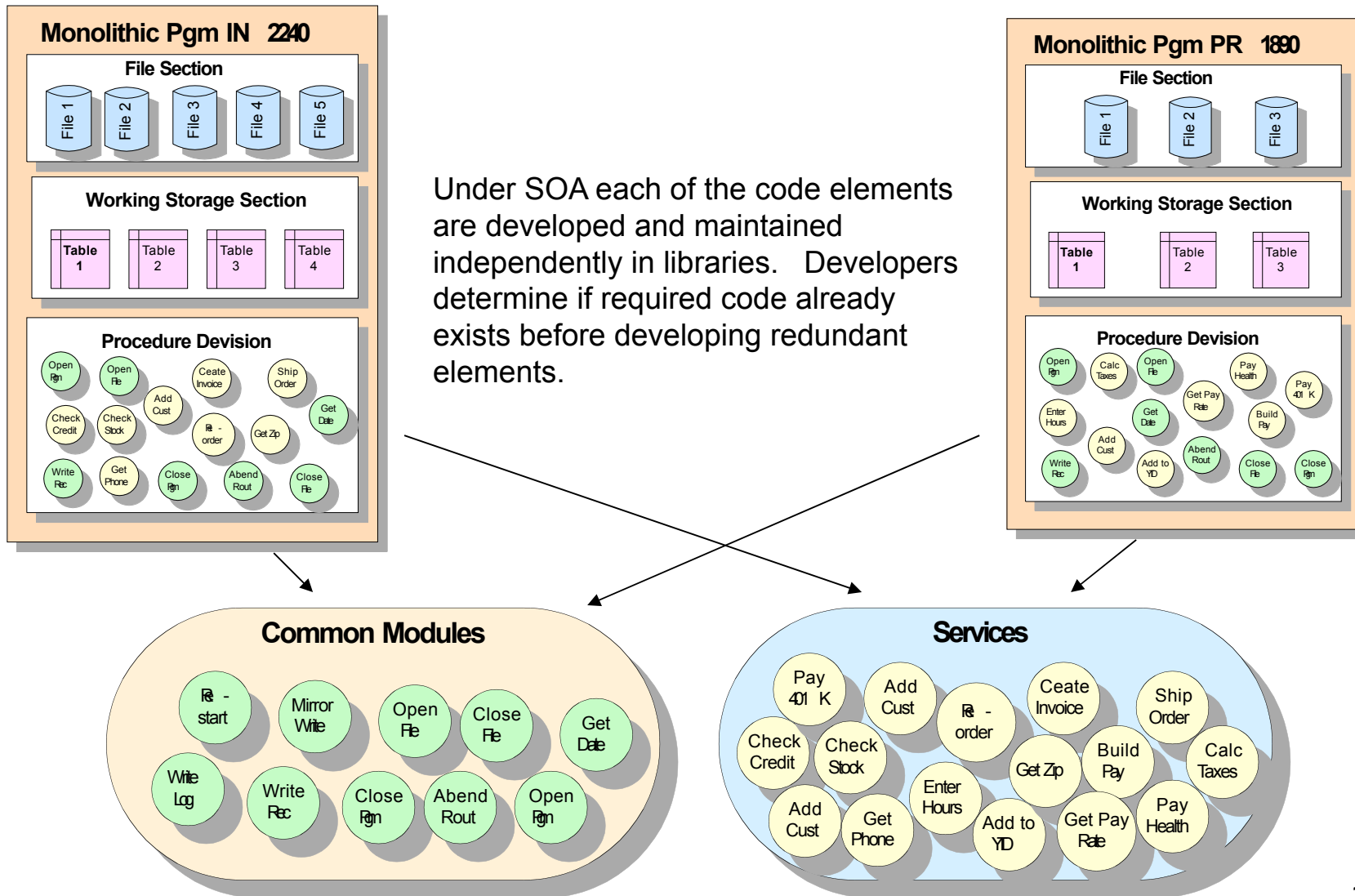


Architectural Styles – SOA and EDA are concepts or approaches for architecting and designing distributed systems. They are complementary and compatible concepts.

- **Service Oriented Architecture (SOA)** – the process of designing applications so that the services perform exactly one task, have one defined set of inputs and outputs, can be invoked by disparate (Webster's: separate, not alike, distinct or different in kind) clients, and have implementation metadata so that the identity and location can be dynamically discovered at runtime.
- **Event Driven Architecture (EDA)** – The process of designing services so that they are event triggered. Under EDA, services support many to many connections, the flow of control is determined by the recipient, supports dynamic parallel asynchronous flow through the network of modules and can react to new, external input that may arrive at unpredictable times.

Web Services – A concept or suite of specifications for defining services for “network” computing. It evolved into defining the development of software components using Simple Object Access Protocol (Soap), Web Services Description Language (WSDL) and/or Universal Description Discovery and Integration (UDDI).

Key Concepts – Service Oriented Architecture

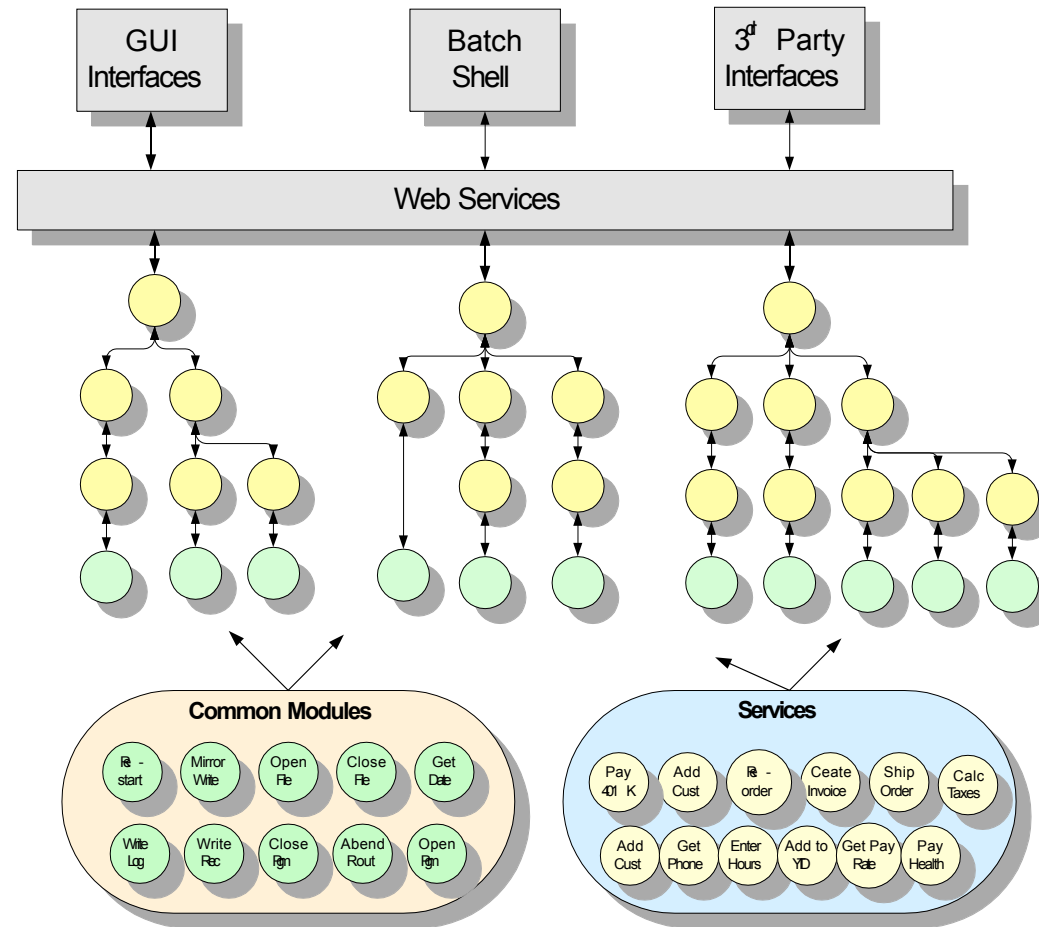


Key Concepts – Service Oriented Architecture



The business functionality is developed by linking pre-existing services and common modules together. Services and common modules are shared.

Developers must be aware of services and common modules that already exist and must not build redundant ones.

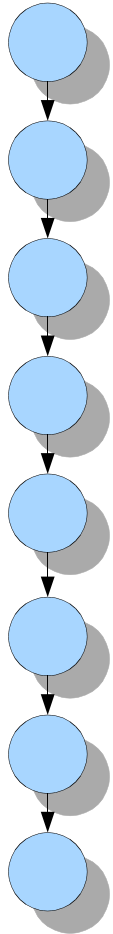


Key benefits 1) Performance – once the work is triggered, multiple threads of activities can run at the same time 2) Ease of development – services and common modules do not need to understand the inter-workings of other services or common modules.

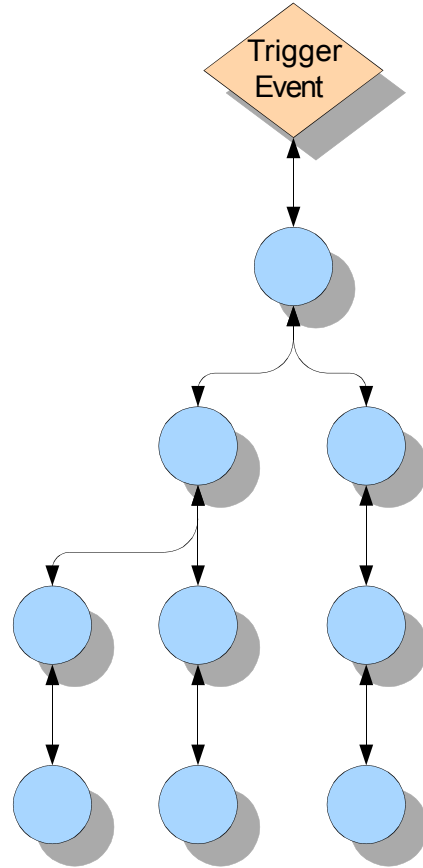
Key Concepts – Service Oriented Architecture



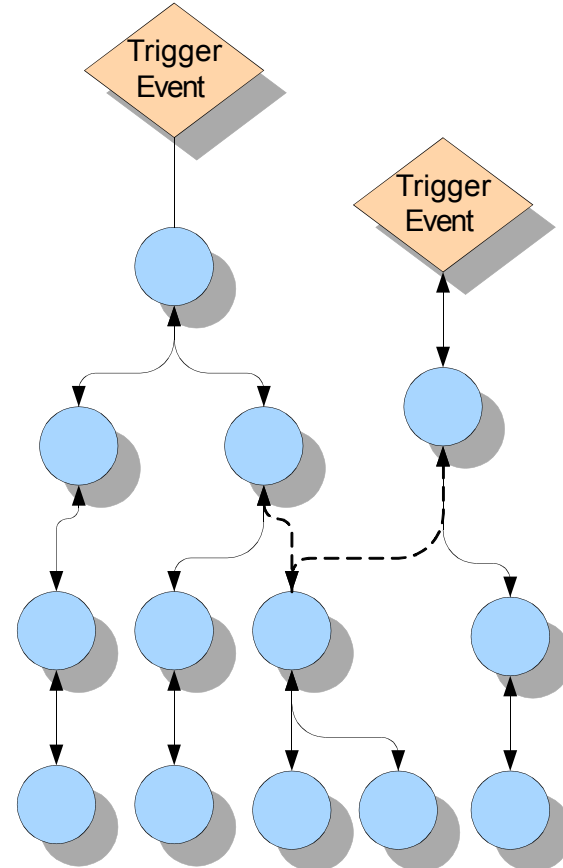
Batch Processing



Service Oriented Architecture



Event Driven Architecture



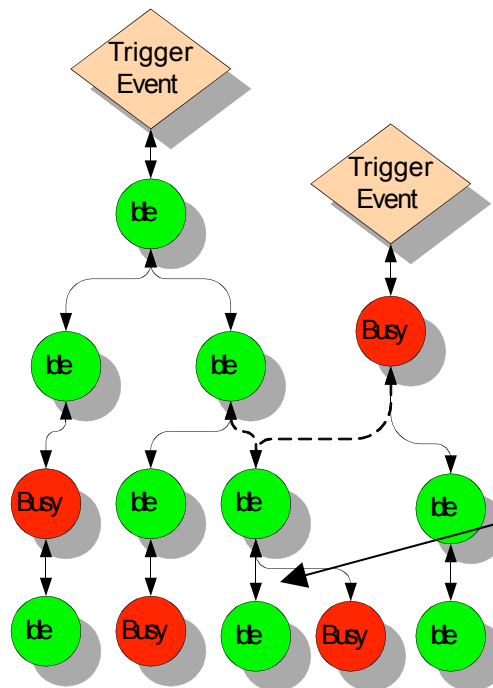
Key Concepts – Service Oriented Architecture



Inter-connectivity (SOA) Synchronous or A-Synchronous

The consumer (sender) knows that an event has occurred and establishes a persistent connection with the provider (receiver).

Once the provider has completed the work, the results are returned to the consumer and the connection is severed.



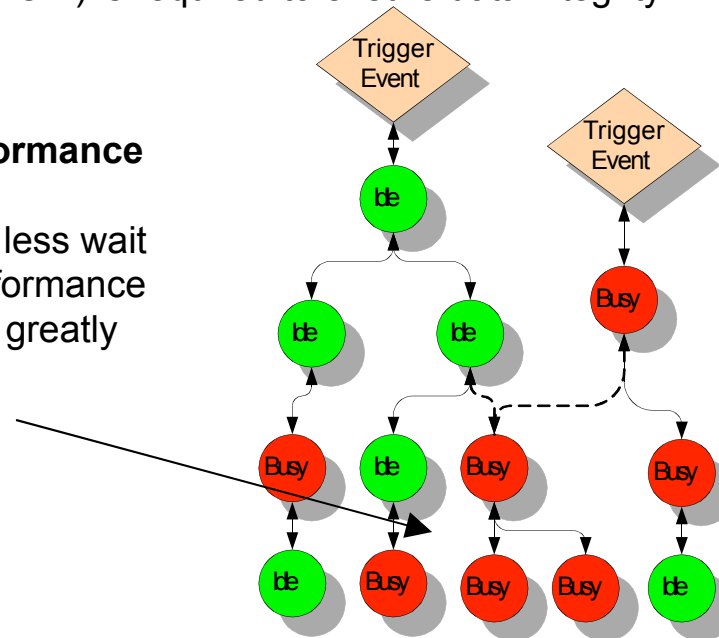
Wait Time and Performance

Under EDA, there is less wait time and overall performance of the systems are greatly improved.

Inter-connectivity (EDA) Requires A-Synchronous

The consumer (sender) knows that an event has occurred and contacts the provider (receiver). Once the provider receives the request, the connection is severed.

Once the work is complete, a connection is made back to the consumer and the results are returned. Special software such as Message Oriented Middleware (MOM) is required to ensure data integrity.



Key Concepts – Service Oriented Architecture

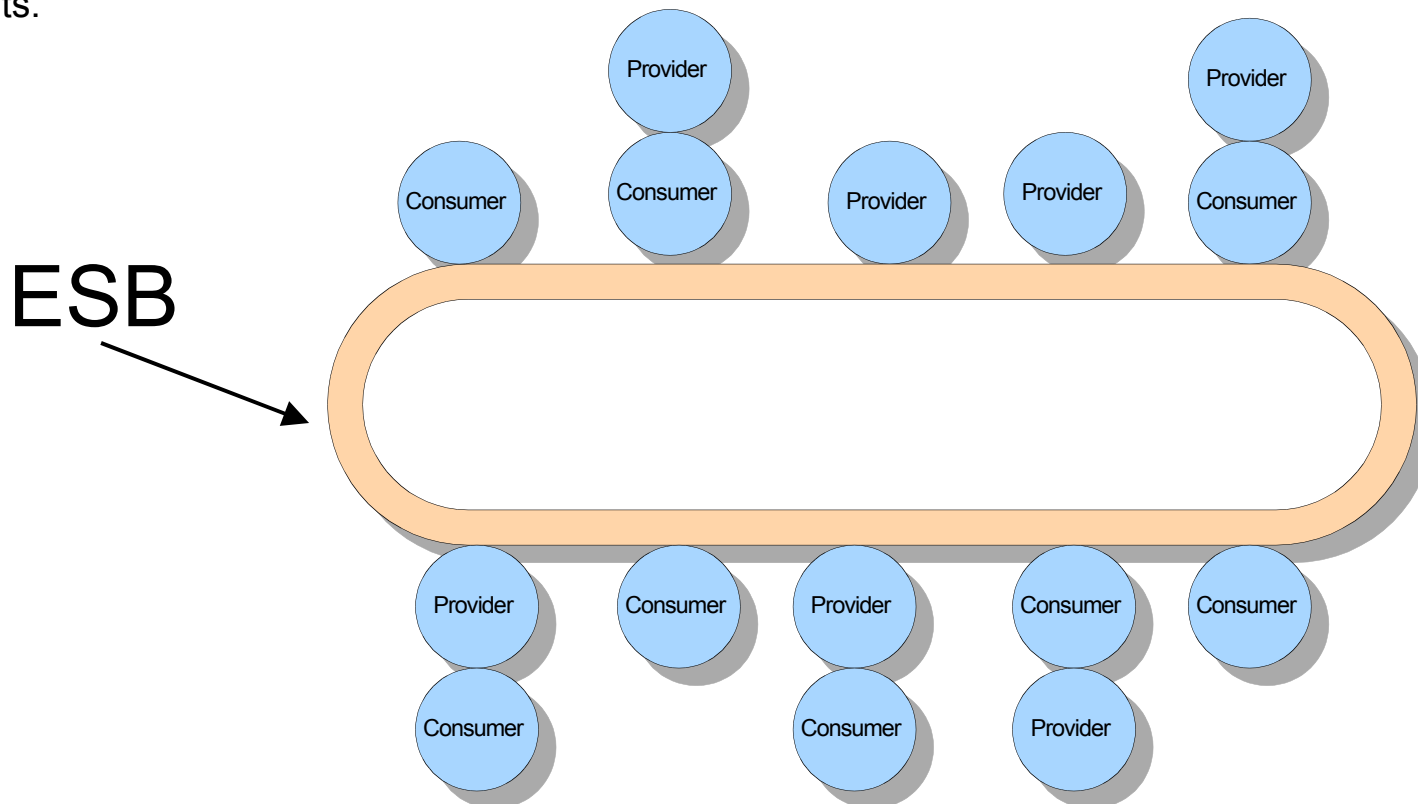


<u>Batch Processing</u>	<u>Service Oriented</u>	<u>Event Driven</u>
<ul style="list-style-type: none">• Are based on monolithic programs• Triggered by time and day or another job• Processes sequentially one step at a time	<ul style="list-style-type: none">• Are typically object oriented• Are triggered by an event or client• Can trigger multiple tasks to occur at the same time• Has one to one connections• Has flow routing that is directed by the consumer• Employs a linear path of execution through a hierarchy of modules• Is closed to new unforeseen input once the flow is started.	<ul style="list-style-type: none">• Are typically object oriented• Are triggered by an event or client• Can trigger multiple tasks to occur at the same time• Supports many to many connections• Has a flow of control that is determined by the recipient based on the message itself• Supports dynamic, parallel asynchronous flows through a network of modules• Can react to new external input that may arrive at unpredictable times.

Key Concepts – Enterprise Service Bus



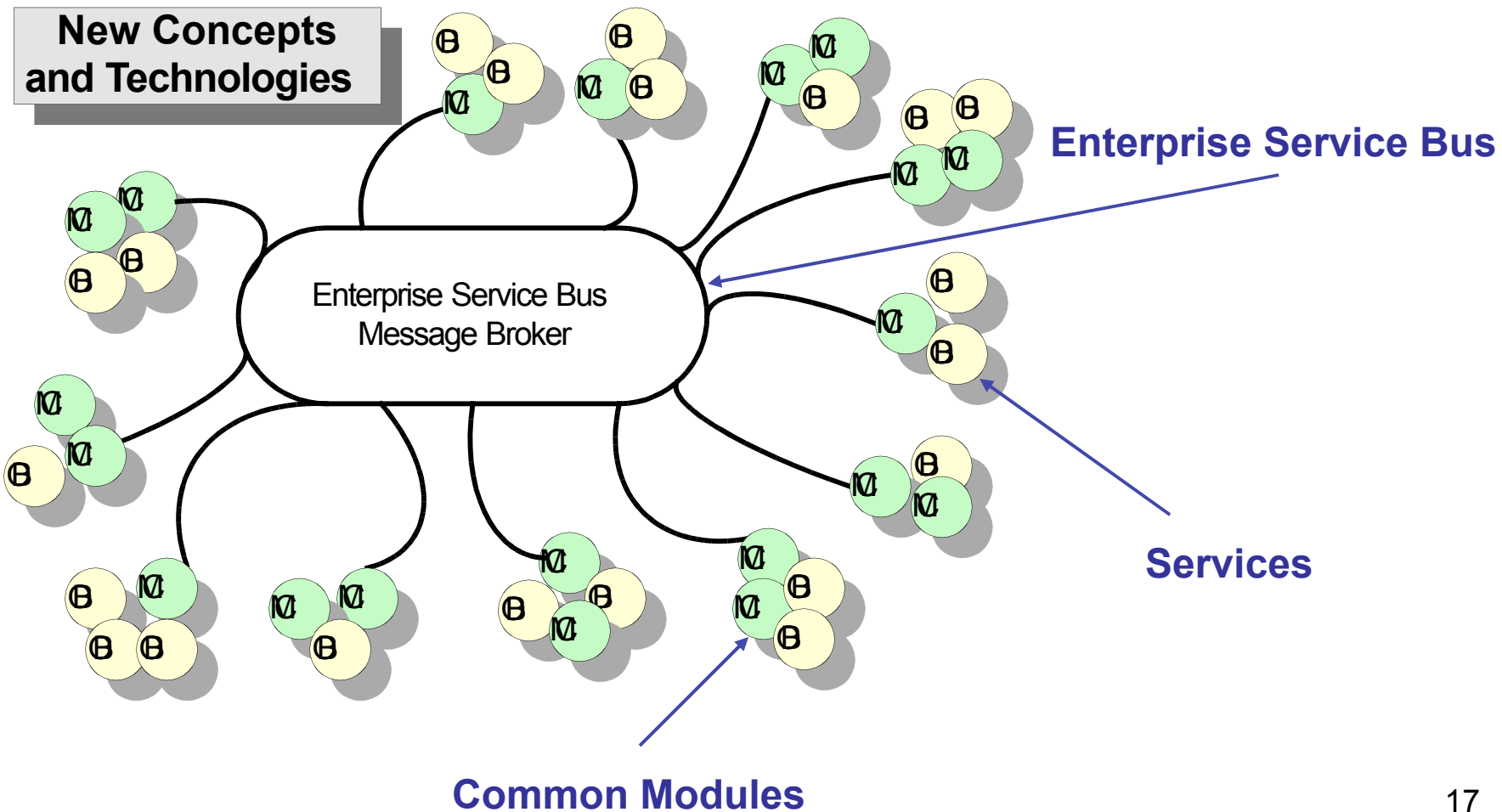
Enterprise Service Bus (ESB) - The concept and technologies that enable SOA and EDA to work. It is a concept or collection of standards under which applications (even of different eras) can exchange data as long as they exchange data through software that meets the requirements of ESB. Numerous communications interface products (e.g. Websphere MQ, Web Services, RMI,) support ESB requirements.



Key Concepts – Future Architecture Revisited



To – Small single task components with loosely coupled endpoints, standard set of technologies, and lower level of complexity



Key Concepts

New Architecture - Characteristics and Challenges



Characteristics

- Programs are small and succinct
- Program and vendor interfaces are based on open technologies
- Java, C, and C++ are frequently used languages
- User screens are based on GUI
- Main files are maintained in current database structures (DB2, Informix)
- Data is exchanged using dynamic movement products such as Websphere MQ, Web Services, RMI, ODBC, ETL, etc.
- Common Routines are shared
- Applications can span multiple platforms
- Development tends to be more iterative
- Maintenance and support is based on specialization
- Design is based on SOA and EDA concepts

Challenges

- Developers must be aware of what objects and frameworks already exist and not generate duplicate objects.

Key Concepts – More Terminology

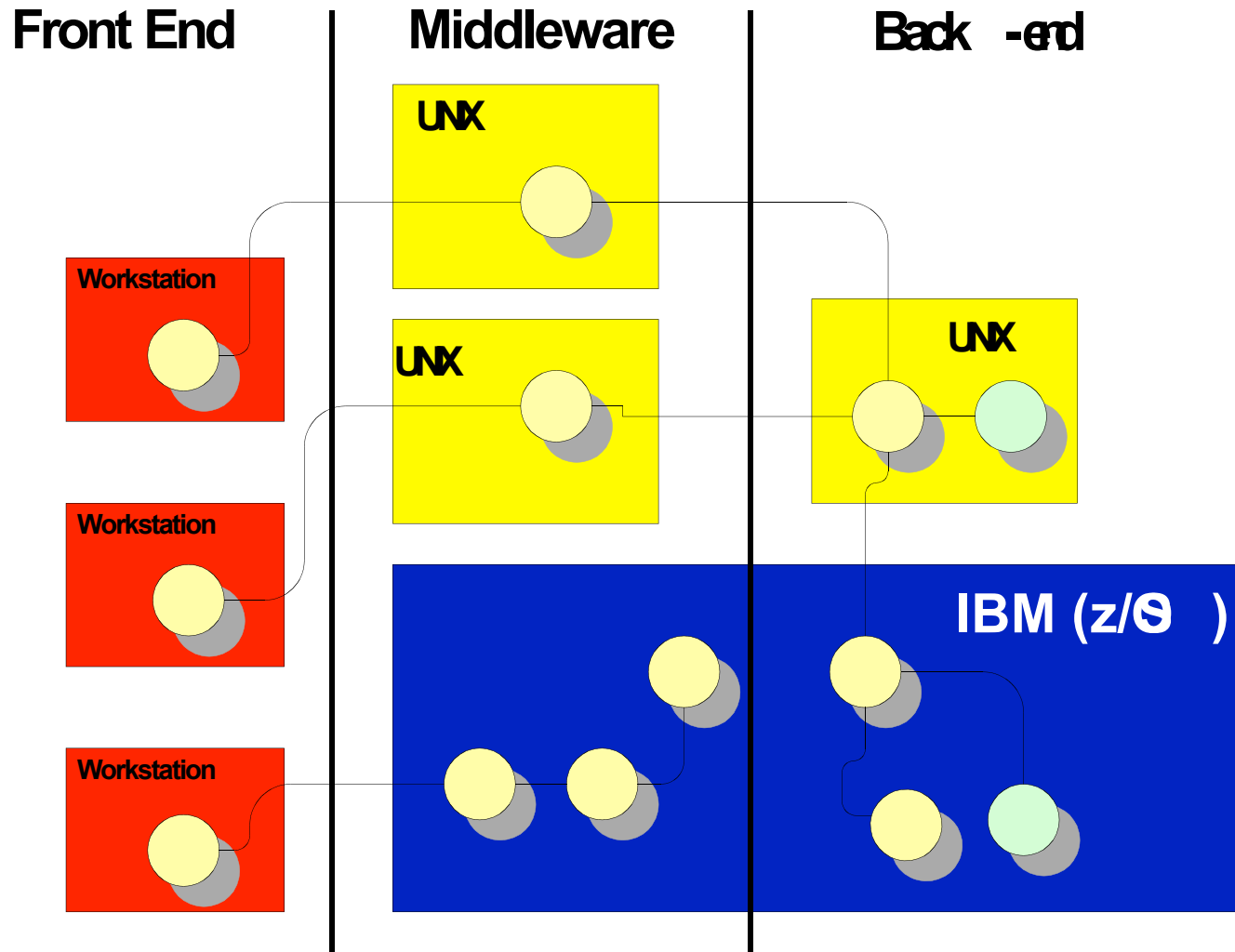


Open Systems – Systems that are constructed based on common protocols or standards (example IEEE). Sun or Apple tend to be more proprietary. Products are considered “Open” with time and momentum as their engineering is shared and other products and services provide interfaces. J2EE is based on open architecture.

Object Oriented – A concept when functionality and data of a software structure are combined into one object. This enables the object to be viewed as a piece of software that contains both state (data) and behavior (function or method) and can be invoked via a published interface. Under an object oriented concept, the logic that makes up a system is broken into individual succinct functions or units of work. These objects are shared and one object does not need to understand the interworkings of other objects. Developers must be aware of what objects already exist and develop new objects to cover new functions. Change it once and all areas that use it are impacted.

Multi-tier – Applications that have components that reside on multiple platforms. Having open systems that comprise of smaller single purpose functions, facilitates placing these components where it makes the most sense from a performance, security, availability, cost and maintenance perspective. Multi-tier is somewhat obsolete or implied. A more advanced and current concept is “Enterprise Layering”, where each layer specializes in a specific function and cross function mixing does not occur.

Key Concepts – Illustration of multi-tier applications





Key Concepts – Terminology (Where to Deploy)

Clients – Earlier we defined a client as the triggering component that requested a work from another component. Another definition of client is the workstation that the end user uses to perform their daily activities. These workstations might be referred to as either:

- Thin Clients – are applications that require very little data or logic to reside on the users workstation. Web applications are examples of extremely thin clients.
- Thick Clients (often referred to as Rich Clients) – are applications that require a lot of data or logic on the workstation.

Generally thin clients are preferable to rich clients, but some applications are more suited to rich clients.



Tools and Programming Languages

Tools and Programming Languages – Terminology



Java – A programming language developed by Sun. It provides most of the same features of other programming languages such as C++.

J2EE – Stands for Java 2 Enterprise Edition. It is a set of application development tools based on the Java programming language, for very large distributed applications. It contains all of the functionality of J2SE (Java 2 Standard Edition).

J@EE has become an open standard, meaning Sun has provided specifications on how to develop J2EE servers and applications. Most vendors including IBM, recognize the popularity of J2EE and are providing products based on J2EE Standards.

Java – Many vendors are now offering J2EE compatible application servers. For example: BEA, WebLogic, IBM WebSphere, and Sun's Java Enterprise System.

J2EE application servers provide background services that make it easier for developers to develop modern business applications using SOA and EDA design patterns.

Tools and Programming Languages

Emerging Technologies



Category	Current Technologies	Future Technologies
Platforms	Solaris, HP/U&X, z/OS, AIX, Windows	HP/UX, z/OS, AIX. Windows
Application Servers	SunOne, WebSphere	WebSphere
Transaction Management	CICS, TUXEDO, SunOne	WebSphere, DB2 Stored Procedures, CICS
Mainframe Languages	Cobol II, Java, OS/390, Assembler	Cobol II, Java
Middleware Languages	C++, Java, HTML	C++, Java, HTML
Client Languages	C++, Java, PowerBuilder, C#, VisualBasic, HTML, JavaScript	C#, Java, C++ VisualBasic, .Net, HTML, JavaScript
Distributed Software Migration	CMVC	Rational ClearCase/ ClearQuest



Developing an Architectural Plan



Fail to plan and you are planning to fail

We have discussed exciting new concepts that provide such potential benefits. But how do we achieve our goal? How do we develop a comprehensive and complete vision of our “Target Business Systems”? How do we ensure the people that are building the new systems understand that vision.



Creating and communicating the vision of your target systems

Developing an Architectural Plan - Terminology



Enterprise Architecture – At a broad perspective, Enterprise Architecture is a science or methodology for defining and planning how business systems should be constructed using the most effective concepts and technologies in order to best meet business objectives. It has become a specialization with the information technology field.

Artifact – A piece of digital information. An artifact may be any size, and may be composed of other artifacts. Examples of artifacts: a URI, an XML Document, a PNG image, a bit stream. Examples of artifacts are models and patterns.

Model – A preliminary representation or illustration of something: the plan from which the final project is to be constructed. Highest level models create the context for more detailed architecture work that highlights relationships within the enterprise and with business partners. High level models are used to aid in communications with the business executives and must draw on their knowledge and perspectives. Models could be described as logical, physical or emerging.

Developing an Architectural Plan - Terminology



Pattern – A form, template or model (or, more abstractly, a set of rules) which can be used to make or to generate things or parts of a thing, especially if the things that are generated have enough in common for the underlying pattern to be inferred or discerned, in which case the things are said to exhibit the pattern.

Architectural Patterns – a practical or logical construct that shows the interaction of key logical elements of functionality and the relationships of the components to carrying out core elements of system design.

Software or Design Patterns – are meant only to address recurring problems or to create repeatable or reusable functionality. Patterns may be logical or physical. A logical pattern is a collection of “Bricks”.

Bricks – Are the foundational building blocks of technology. Bricks help us organize and rationalize the various technologies in use in the enterprise. The term brick is used interchangeable with the term component.

Component - 1) a component is a software object meant to interact with other components, encapsulating certain functionality or a set of functionalities. A component has a clearly defined interface and conforms to a prescribed behavior common to all components with an architecture. 2) A component is an abstract unit of software instructions and internal state that provide a transformation of data via its interface. 3) A component is a unit of architecture with defined boundaries.

Developing an Architectural Plan



Layer – Used within enterprise architecture to methodologies to define the level of granularity with a vision of a target architecture. There is a common theme across the layer.

View – is used to provide a reference for the level and scope of the information that you are looking at or the model or pattern that you are developing.

Where to Deploy

