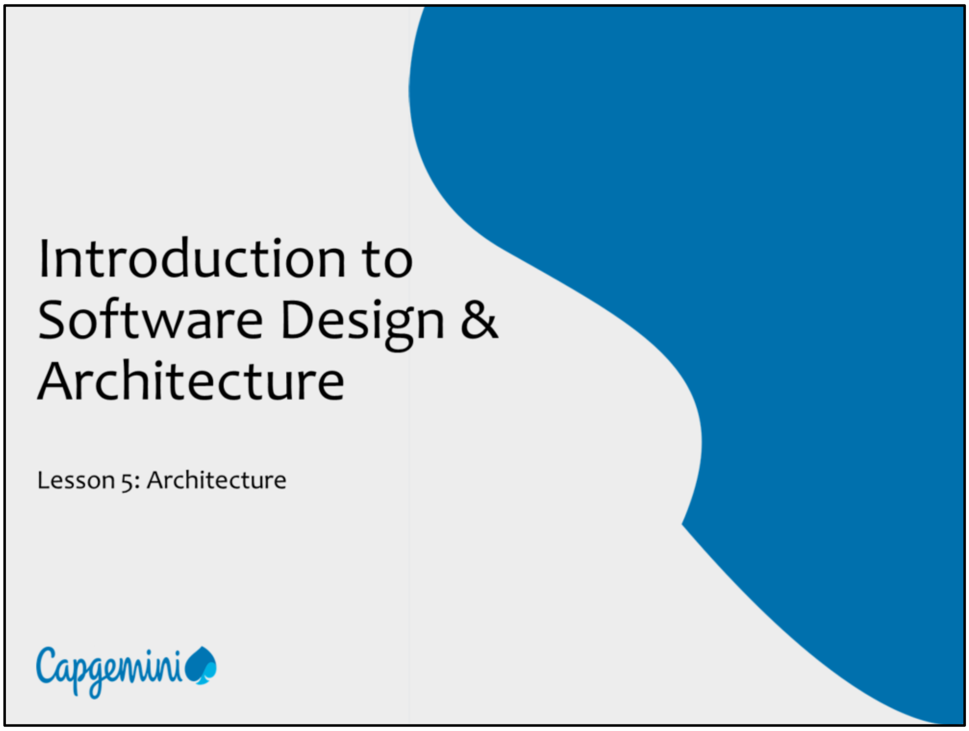
Instructor Notes:

Add instructor notes here.



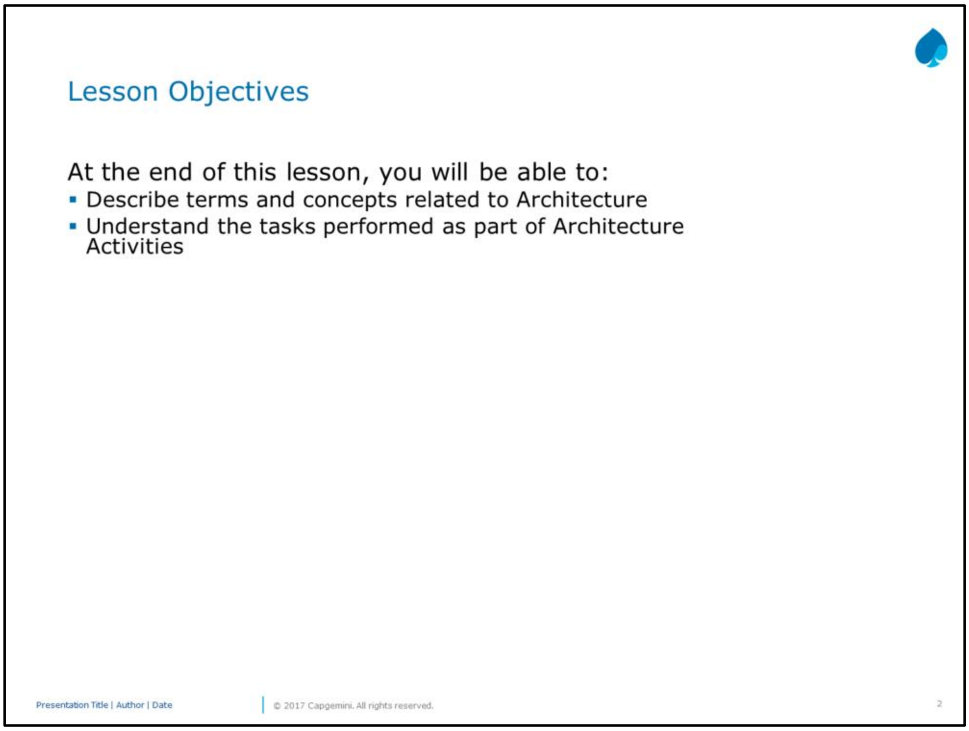
Introduction to Software Design & Architecture

Lesson 5: Architecture

Capgemini

Instructor Notes:

Having set the context for Architecture & Design discipline, we now look at the first set of activities. Emphasize that since this is a course on design, we are only touching upon the architecture bit and would not really be getting into the details.



Lesson Objectives

At the end of this lesson, you will be able to:

- Describe terms and concepts related to Architecture
- Understand the tasks performed as part of Architecture Activities

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

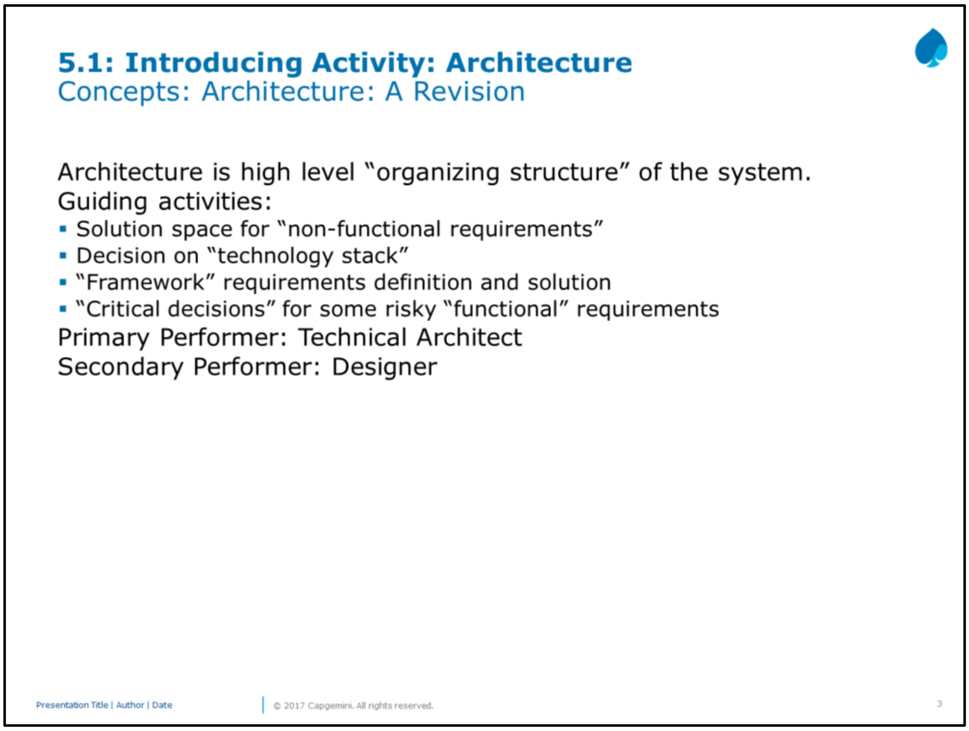
2

Lesson Objectives:

- In this lesson, we shall look at terms, concepts, and tasks related to Architecture.
- Do note that **Architecture** is a deep subject in itself. Hence in this course on Design, we are restricting our study of Architecture to:
  - See certain architectural patterns
  - See how to model the same in UML

Instructor Notes:

Encourage participants to recall what we have already seen in an earlier section.



5.1: Introducing Activity: Architecture
Concepts: Architecture: A Revision

Architecture is high level "organizing structure" of the system.
Guiding activities:
- Solution space for "non-functional requirements"
- Decision on "technology stack"
- "Framework" requirements definition and solution
- "Critical decisions" for some risky "functional" requirements

Primary Performer: Technical Architect
Secondary Performer: Designer

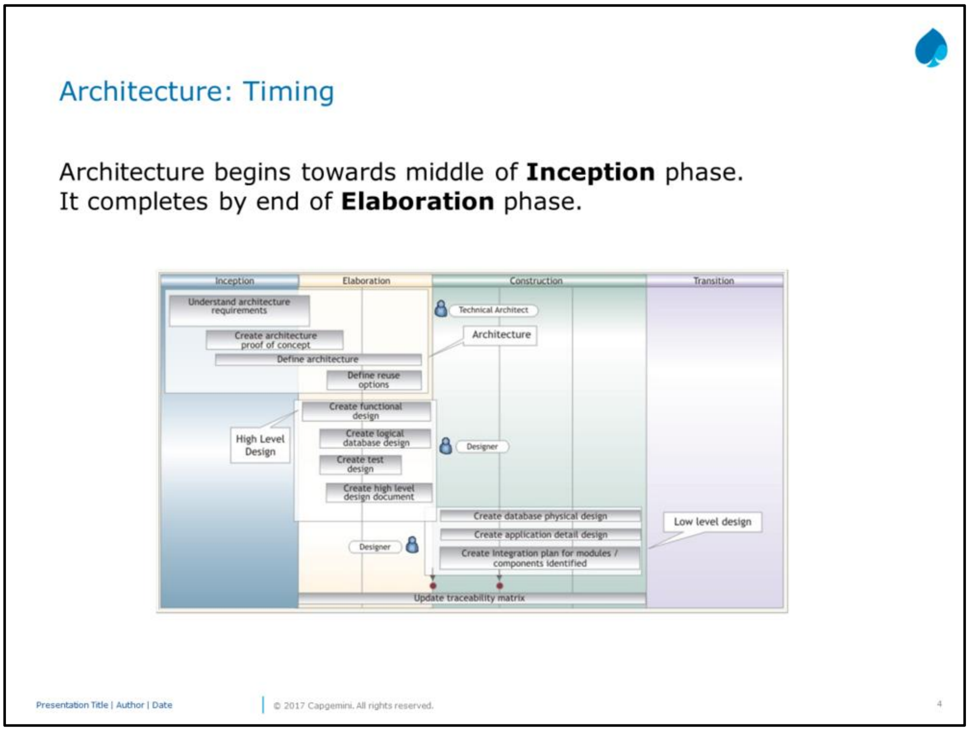
Presentation Title | Author | Date | © 2017 Capgemini. All rights reserved. 3

<u>Concepts: Architecture: Revision</u>:

- So far, we have already seen that Architecture gives the skeletal framework for the application. It is the solution space for Non-Functional Requirements and also looks at risky Functional Requirements. Decision on the technology stack is taken as part of the architecture activities.
- The Technical Architect is the role responsible for these activities.

Instructor Notes:

While RM activities are still in progress in the Inception phase, the Architectural activities are initiated towards mid of Inception. Architecture is baselined by end of elaboration.



Architecture: Timing

Architecture begins towards middle of **Inception** phase.
It completes by end of **Elaboration** phase.

Inception
Elaboration
Construction
Transition
Understand architecture requirements
Create architecture proof of concept
Define architecture
Define reuse options
Technical Architect
Architecture
Create functional design
High Level Design
Create logical database design
Create test design
Create high level design document
Designer
Create database physical design
Create application detail design
Create integration plan for modules / components identified
Designer
Low level design
Update traceability matrix

Presentation Title | Author | Date

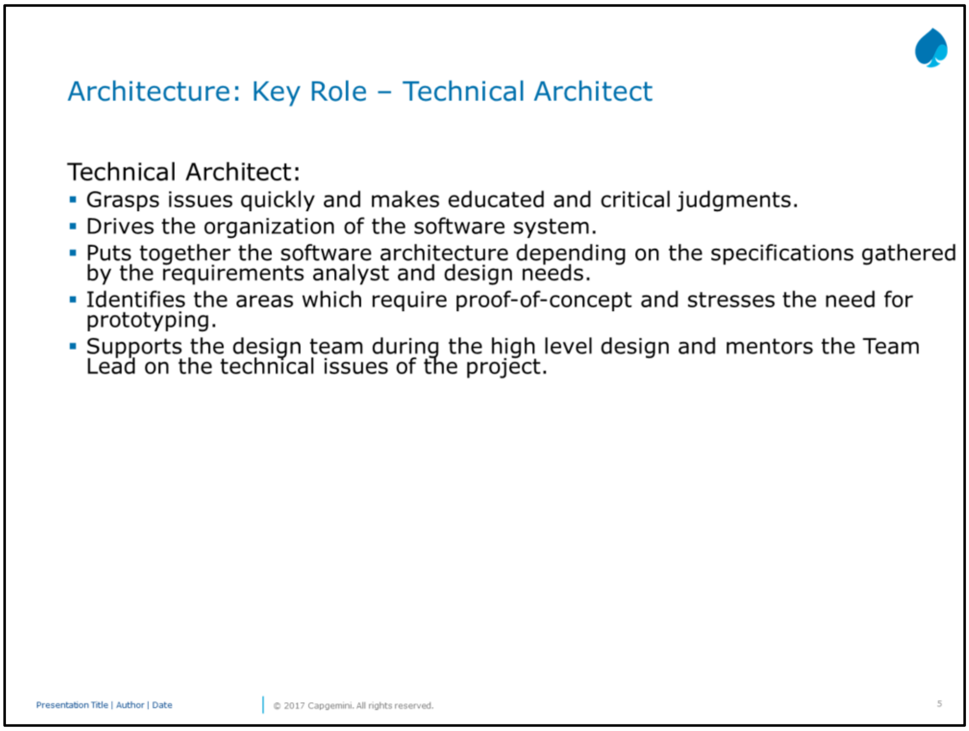
4

Architecture: Timing:

- Activities related to **architecture** begin towards mid of the **Inception** phase.
- As discussed earlier, architecture is considered as one of the highest risks, and is hence baselined by the end of **Elaboration** phase.

Instructor Notes:

Width and depth of technical expertise is a must for the role of Technical Architect. There would often be contradictory demands on the architecture of the application and the Architect has to balance these conflicting requirements and make informed decisions about the architecture. Besides, the Architect should also be able to effectively communicate technical aspects to the client and development team so the role also demands good communication skills.



Architecture: Key Role – Technical Architect

Technical Architect:

- Grasps issues quickly and makes educated and critical judgments.
- Drives the organization of the software system.
- Puts together the software architecture depending on the specifications gathered by the requirements analyst and design needs.
- Identifies the areas which require proof-of-concept and stresses the need for prototyping.
- Supports the design team during the high level design and mentors the Team Lead on the technical issues of the project.

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

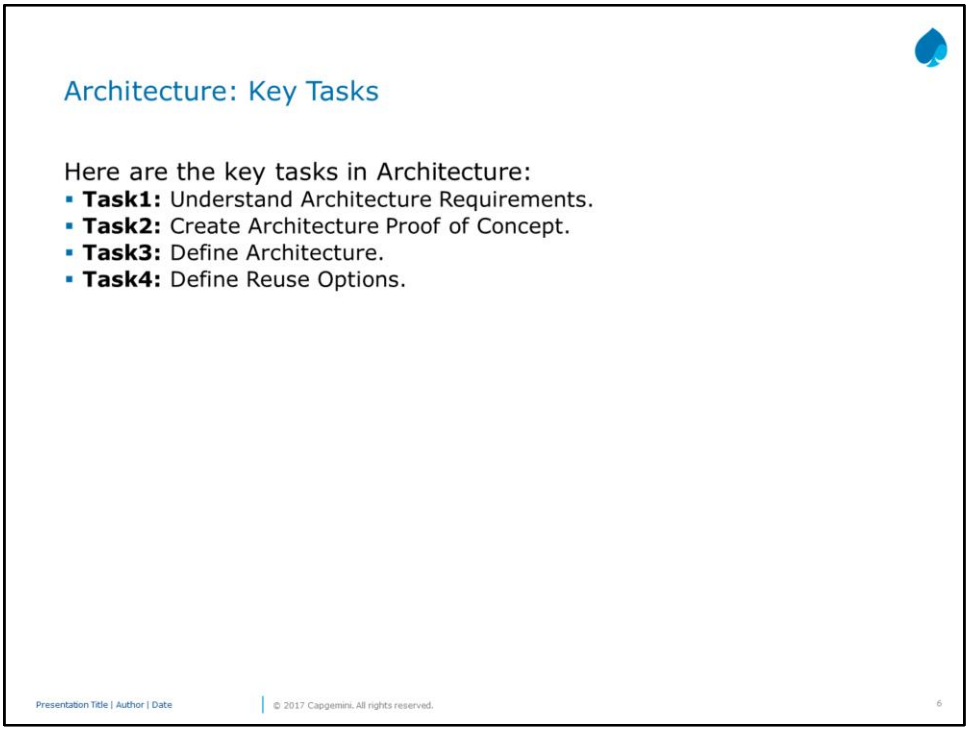
5

<u>Architecture: Key Role – Technical Architect</u>:

- The competencies and expectations from the Technical Architect are outlined here.
- The Technical Architect is not only responsible for coming up with the architecture of the application, but also providing technical inputs and guiding the team during the design activities.

Instructor Notes:

Here are the key tasks which will get discussed in detail in the later sections.



Architecture: Key Tasks

Here are the key tasks in Architecture:

- **Task1:** Understand Architecture Requirements.
- **Task2:** Create Architecture Proof of Concept.
- **Task3:** Define Architecture.
- **Task4:** Define Reuse Options.

Presentation Title | Author | Date

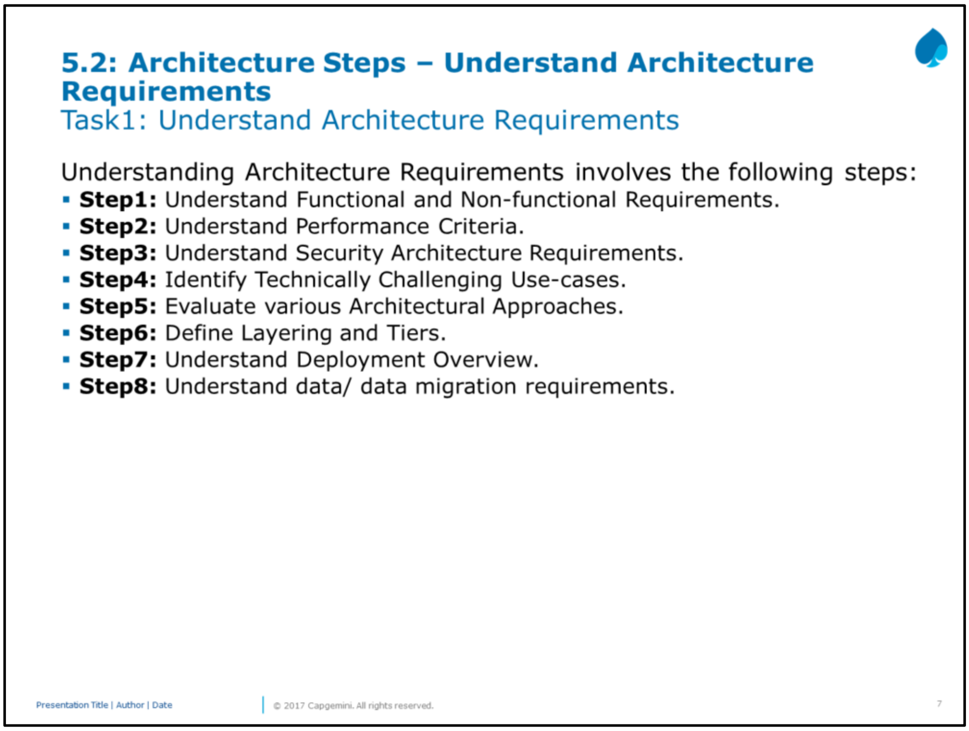
© 2017 Capgemini. All rights reserved.

6

Architecture: Key Tasks:
The various tasks of Architecture are listed in the above slide. We shall now look at the key steps that get done in each of these tasks. Let us first discuss the steps for the task “Understand Architecture Requirements”.

Instructor Notes:

Here are the steps of the first task viz. Understand Architecture Requirements.



5.2: Architecture Steps – Understand Architecture Requirements

Task1: Understand Architecture Requirements

Understanding Architecture Requirements involves the following steps:

- **Step1:** Understand Functional and Non-functional Requirements.
- **Step2:** Understand Performance Criteria.
- **Step3:** Understand Security Architecture Requirements.
- **Step4:** Identify Technically Challenging Use-cases.
- **Step5:** Evaluate various Architectural Approaches.
- **Step6:** Define Layering and Tiers.
- **Step7:** Understand Deployment Overview.
- **Step8:** Understand data/ data migration requirements.

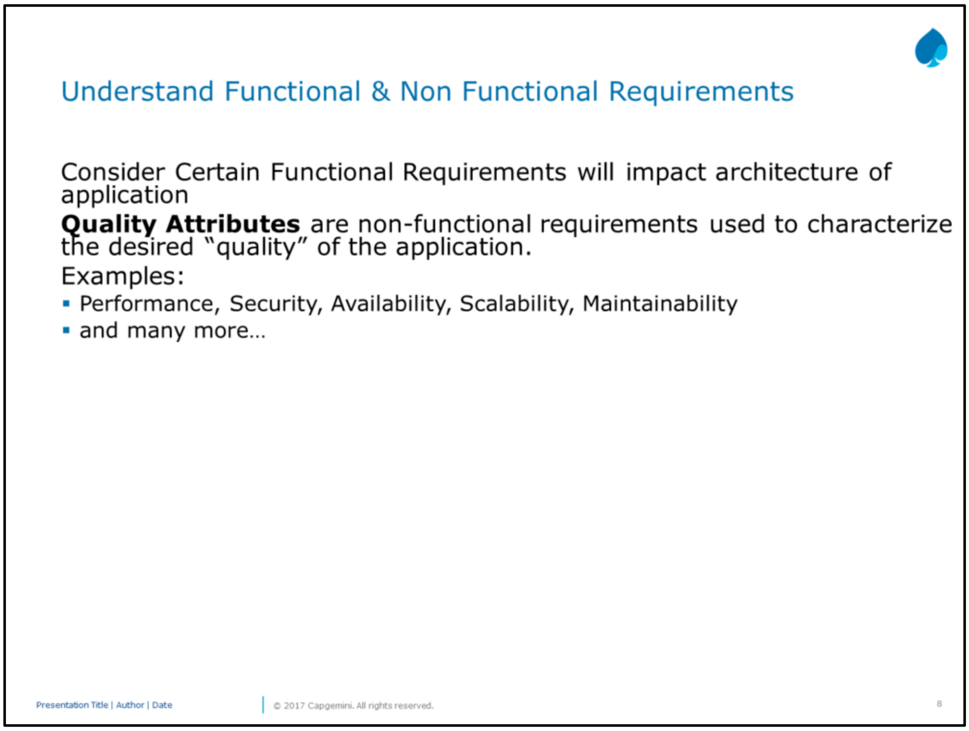
Presentation Title | Author | Date

7

Task1: Understand Architecture Requirements:
Steps of the first task, that is, Understand Architecture Requirements, are listed in the above slide. We shall look into each of these steps in the slides that follow.

Instructor Notes:

Quality attributes are sometimes named "ilities“ since many of the quality attributes share this common suffix!



Understand Functional & Non Functional Requirements

Consider Certain Functional Requirements will impact architecture of application

Quality Attributes are non-functional requirements used to characterize the desired “quality” of the application.

Examples:

- Performance, Security, Availability, Scalability, Maintainability
- and many more…

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

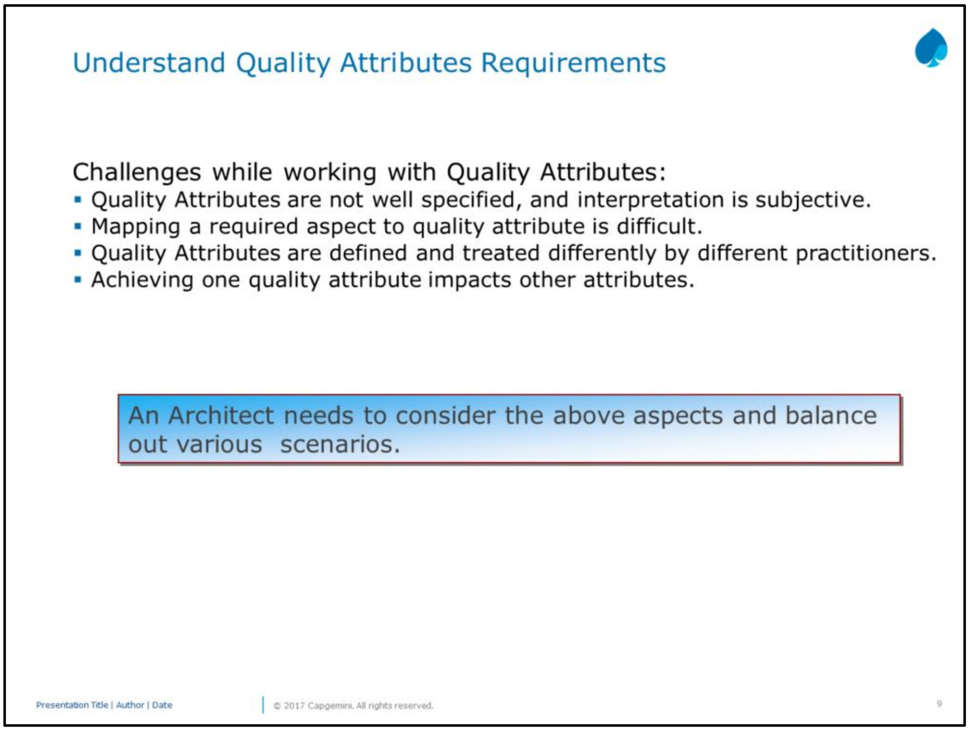
8

Task1: Understand Architecture Requirements:
Step1: Understand Quality Attributes Requirements:

- Certain functionality requirements will impact architecture for eg. need to archive, storing log of transactions etc.
- Other than the business functionality, the application needs to exhibit certain desired qualities, which are captured as quality attributes of the system.
- Unfortunately, the functionality often takes the front seat or only seat in the software design lifecycle, and many a times not enough attention is paid to the quality attributes. Most of the times, redesign of the system occurs not because the application is functionally deficient but because it is slow, not secure, difficult to use and maintain, and so on.
- Architecture is the first stage where quality requirements can be addressed. The required qualities have to be designed and evaluated at the architecture level. Quality requirements cannot be restricted only at the architecture level though. They need to be followed through in the subsequent steps too. Else the quality can deteriorate.
- For example, consider the criterion of **Performance**.
  - ➢ During the Architecture stage, the point of consideration can be how shared resources are allocated or what is the amount of communication necessary between components.
  - ➢ During the post-architecture stage, performance can be impacted by the choice of algorithms or how the algorithm gets implemented in the code.

Instructor Notes:

An Architect’s role is key in ensuring that the conflicting requirements are taken care of while designing the architecture of the system.



Understand Quality Attributes Requirements

Challenges while working with Quality Attributes:

- Quality Attributes are not well specified, and interpretation is subjective.
- Mapping a required aspect to quality attribute is difficult.
- Quality Attributes are defined and treated differently by different practitioners.
- Achieving one quality attribute impacts other attributes.

An Architect needs to consider the above aspects and balance out various scenarios.

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

9

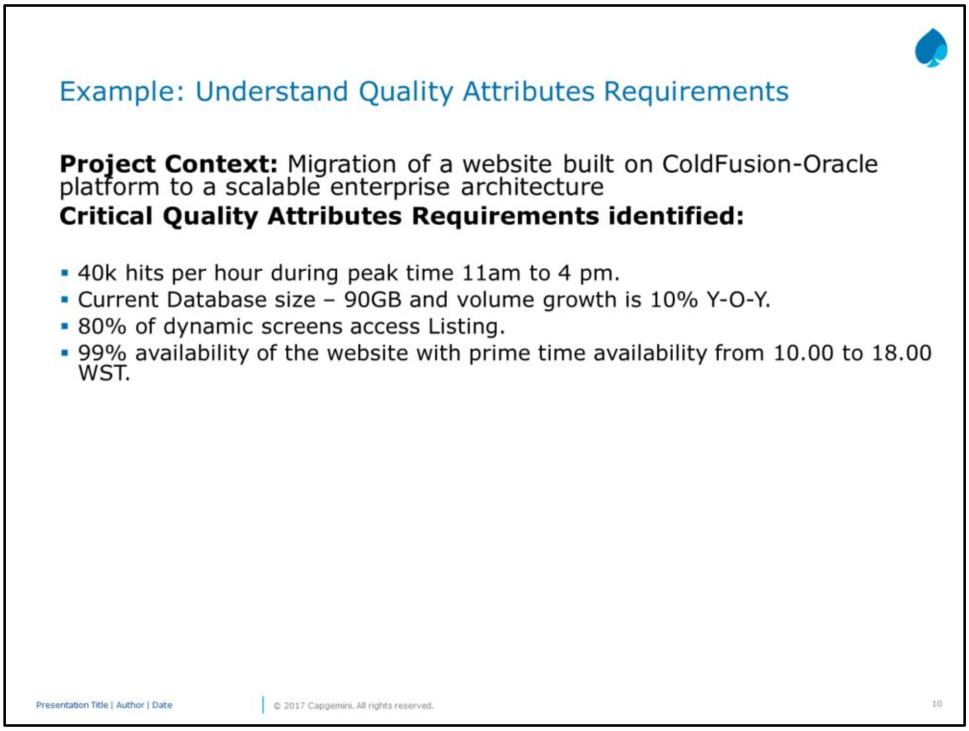
Task1: Understand Architecture Requirements:
Step1: Understand Quality Attributes Requirements (contd.):
Working with quality attributes may not be straight forward though.

1. Many a times, the requirement is vaguely stated, leading to multiple interpretations. For example, “the system should be *modifiable*” is a vague statement because every system is modifiable to some extent. When the requirement is “the system should provide high performance”, it is not clear whether it refers to data retrieval, user response, or algorithmic processing.
2. Sometimes, the attribute concerns are overlapping and may map to multiple quality attributes. For example, System failure can be an aspect of Availability, Usability, or Security.
3. There is not much standardization on the taxonomy and definitions among the researchers. So one may come across the same aspect being mentioned and handled differently by different groups.
4. However, the most important challenge is the “tension” that exists between quality attributes. Quality attributes cannot be achieved in isolation, and achievement of one quality attribute can have positive or negative effects on other quality attributes. For example, consider the criterion of performance. Achieving Security, Availability, Portability may all negatively impact performance.

It is therefore necessary for the architect to consider these conflicting scenarios and balance them out while arriving at an architectural solution.

Instructor Notes:

Example of non-functional requirements from a Project involving Website migration.



Example: Understand Quality Attributes Requirements

Project Context: Migration of a website built on ColdFusion-Oracle platform to a scalable enterprise architecture

Critical Quality Attributes Requirements identified:

- 40k hits per hour during peak time 11am to 4 pm.
- Current Database size – 90GB and volume growth is 10% Y-O-Y.
- 80% of dynamic screens access Listing.
- 99% availability of the website with prime time availability from 10.00 to 18.00 WST.

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

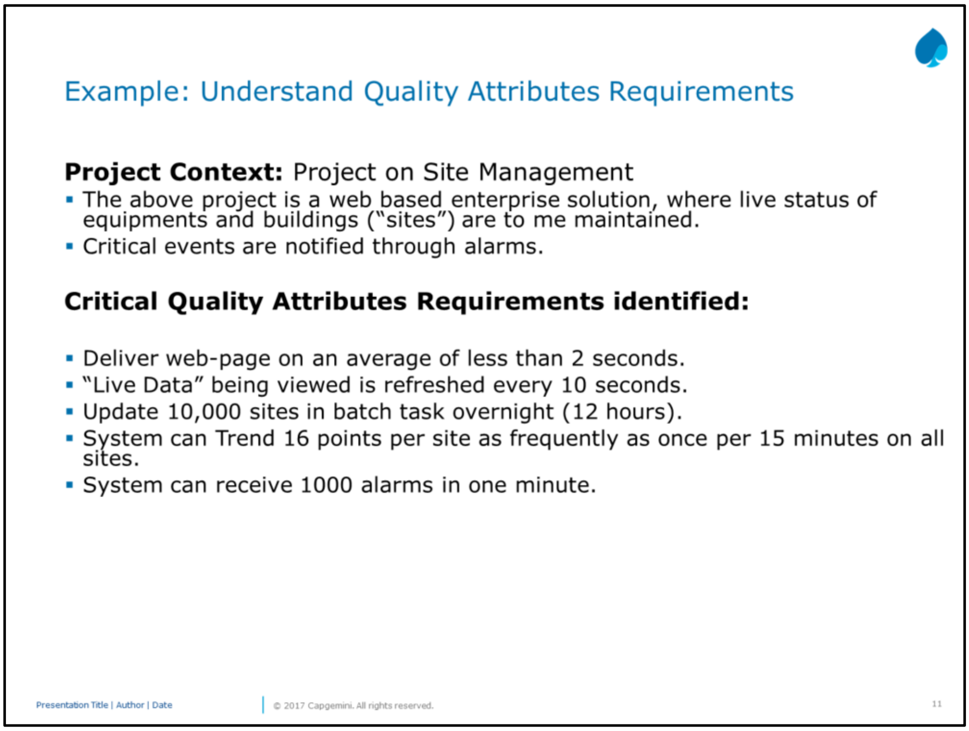
10

Note:

In the above project, some of the non-functional requirements identified are listed in the slide.

Instructor Notes:

Example of non-functional requirements from a Project on Site Management.



Example: Understand Quality Attributes Requirements

Project Context: Project on Site Management

- The above project is a web based enterprise solution, where live status of equipments and buildings (“sites”) are to me maintained.
- Critical events are notified through alarms.

Critical Quality Attributes Requirements identified:

- Deliver web-page on an average of less than 2 seconds.
- “Live Data” being viewed is refreshed every 10 seconds.
- Update 10,000 sites in batch task overnight (12 hours).
- System can Trend 16 points per site as frequently as once per 15 minutes on all sites.
- System can receive 1000 alarms in one minute.

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

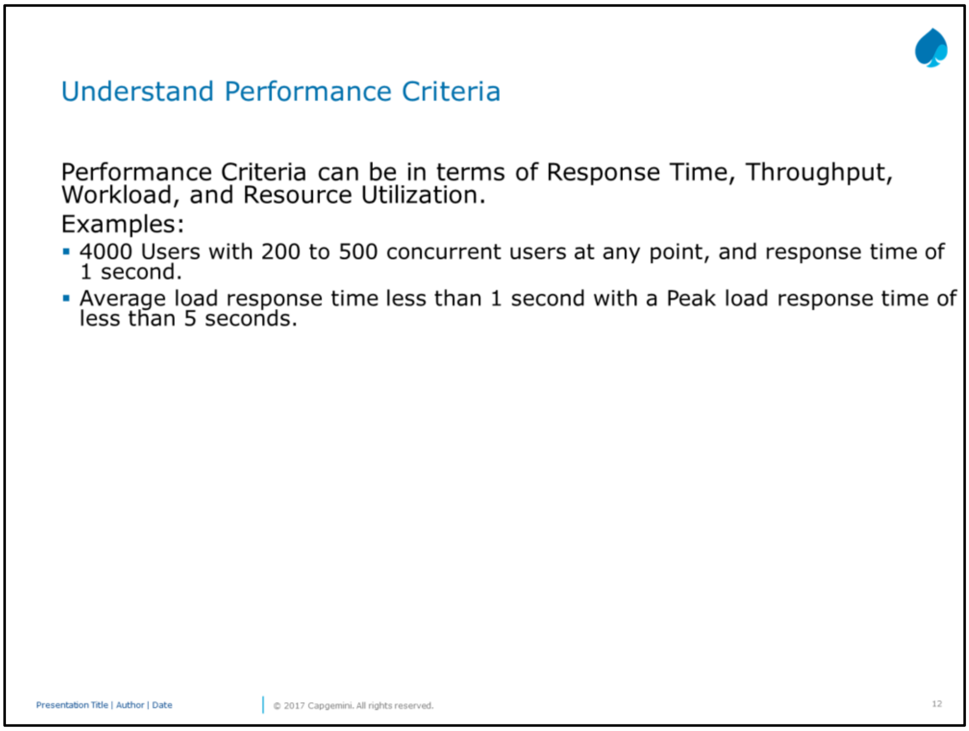
11

Note:
The above project is a web based enterprise solution, where live status of equipments and buildings (“sites”) are to me maintained. Critical events are notified through alarms. Note some of the non-functional requirements for this application.

Instructor Notes:

Qzen provides detailed guidance on Performance workflow.

It is important for the performance objectives to be realistic, understandable and measurable.



Understand Performance Criteria

Performance Criteria can be in terms of Response Time, Throughput, Workload, and Resource Utilization.

Examples:

- 4000 Users with 200 to 500 concurrent users at any point, and response time of 1 second.
- Average load response time less than 1 second with a Peak load response time of less than 5 seconds.

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

12

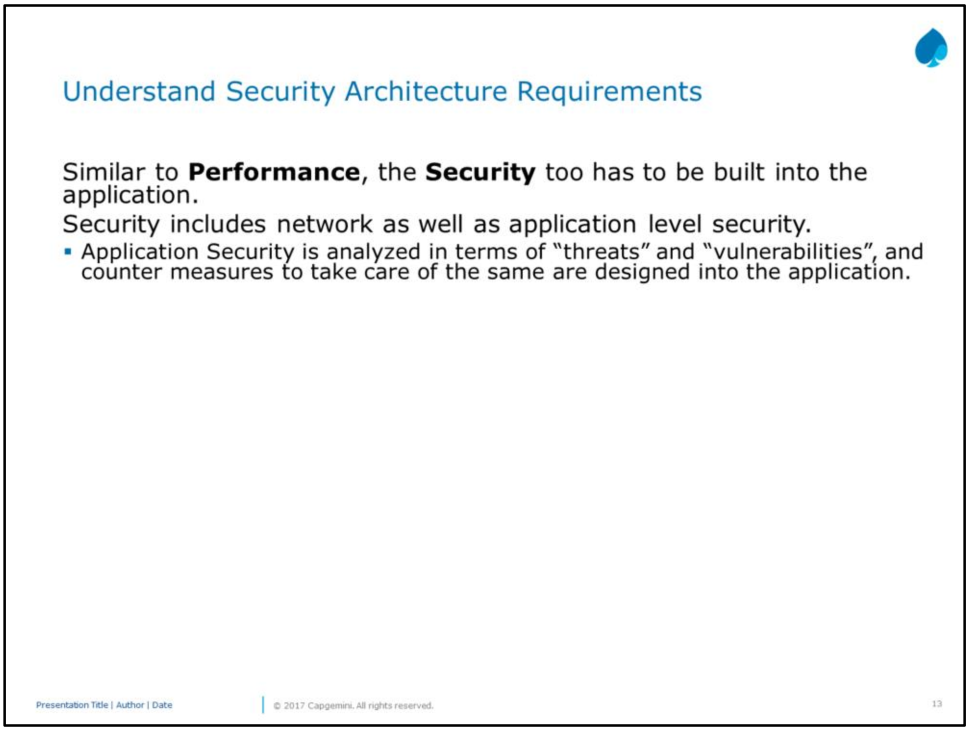
<u>Task1: Understand Architecture Requirements</u>:
<u>Step2: Understand Performance Criteria</u>:

- Performance for an application cannot be an after thought. It has to be looked into pro-actively during the application design.
- Performance Objectives are usually measured in terms of response time, throughput, workload, and resource utilization.
    1. Response time is the time taken by the system to respond to a request.
    2. Throughput is the number of requests that can be serviced by the system in a given unit time. It varies depending on the load.
    3. Workload includes total number of users along with number of active concurrent users and transaction volumes.
    4. Resource utilization is a measure of how much server and network resources are consumed by the application. Resources include CPU, Memory, Disk I/O, and Network I/O.

Instructor Notes:

It is estimated that 75% of the threats are at application level today. OWASP is a leading organization which studies and provides recommendations on security aspects.

Refer to Qzen guidelines on Security for more details.



Understand Security Architecture Requirements

Similar to **Performance**, the **Security** too has to be built into the application.

Security includes network as well as application level security.

- Application Security is analyzed in terms of “threats” and “vulnerabilities”, and counter measures to take care of the same are designed into the application.

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

13

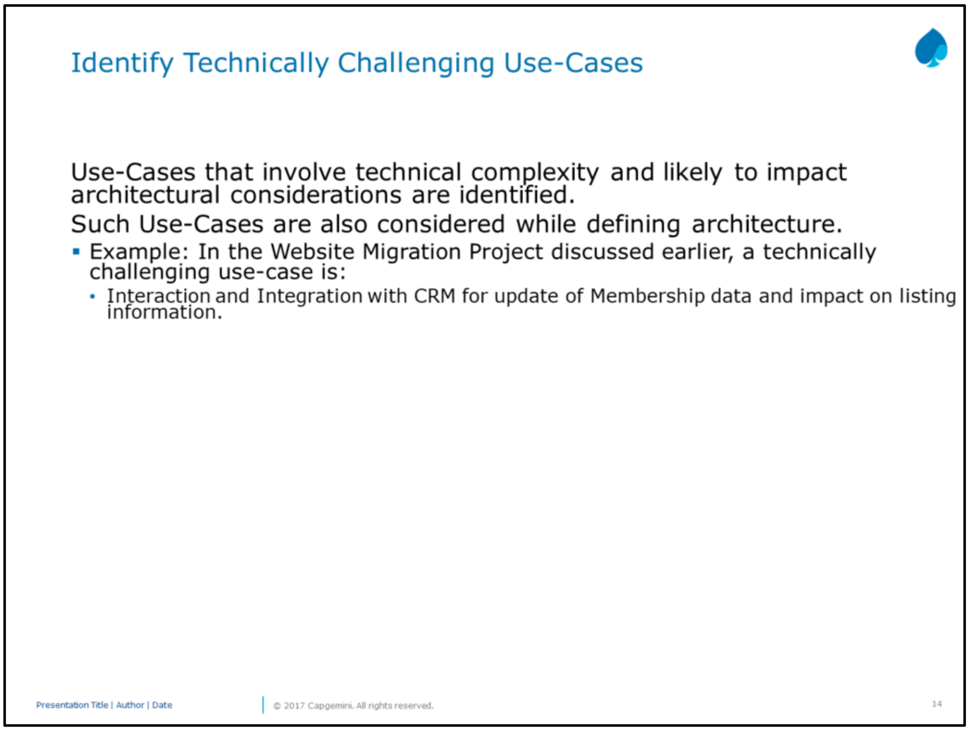
Task1: Understand Architecture Requirements:

Step3: Understand Security Architecture Requirements:

- **Security** too needs a systematic approach which spans across the development stages. Earlier, the focus was more on network level security, and solutions such as Firewalls managed by the System Administrators provided the required solutions. However, now a days apart from the network level, the security at application level too has become a prime concern. A typical valid request that safely passes the network security mechanism can itself contain malicious code which can attack at the application level. Hence providing security at the application level also becomes very important.
- A **threat** is a potential occurrence that can harm the application.
- A **vulnerability** is a weakness in the system that can make the threat possible. For example, if the application does not have input validation, then it is a vulnerability that can be exploited by the user to provide malicious input to the system. Security solutions at application level look at threats and vulnerabilities, and design counter measures that help detect and handle the attacks.

Instructor Notes:

Use-Cases critical to system functionality or technically risky are also taken into account in the initial iterations. Typically these use-cases are such that they cover the “breadth” of the application and hence have a bearing on the application architecture.



Identify Technically Challenging Use-Cases

Use-Cases that involve technical complexity and likely to impact architectural considerations are identified.

Such Use-Cases are also considered while defining architecture.

- Example: In the Website Migration Project discussed earlier, a technically challenging use-case is:
  - Interaction and Integration with CRM for update of Membership data and impact on listing information.

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

14

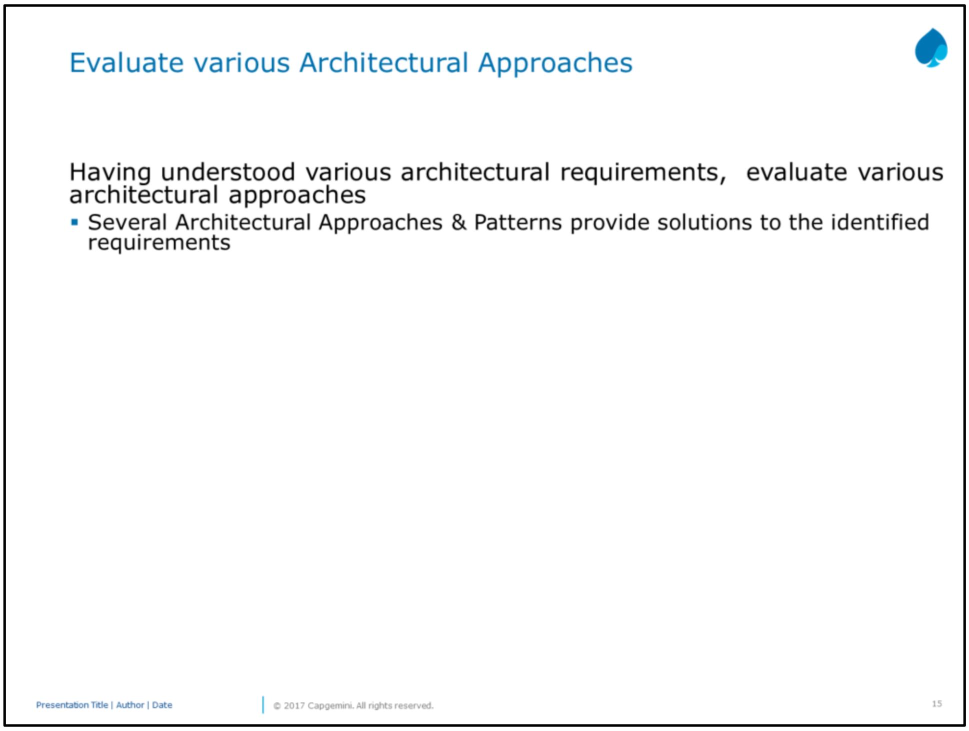
<u>Task1: Understand Architecture Requirements</u>:

<u>Step4: Identify Technically Challenging Use-Cases</u>:

- Though Architecture is essentially the solution space for non-functional requirements, critical functional requirements too are considered from the perspective of minimizing the risks. Use-cases that are technologically challenging and are likely to have an impact on the architectural aspects are considered.
- In the example mentioned in the slide, the data is owned by CRM and not by the application being developed. However, the listing information is maintained within the application. This implies two things.
  - ➢ First, the application needs to connect with CRM.
  - ➢ Second, appropriate mapping to listing information is needed.

Instructor Notes:

Some of the architectural approaches are explained on subsequent slides



Evaluate various Architectural Approaches

Having understood various architectural requirements, evaluate various architectural approaches

- Several Architectural Approaches & Patterns provide solutions to the identified requirements

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

15

<u>Task1: Understand Architecture Requirements</u>:
<u>Step5: Evaluate various Architectural Approaches</u>:

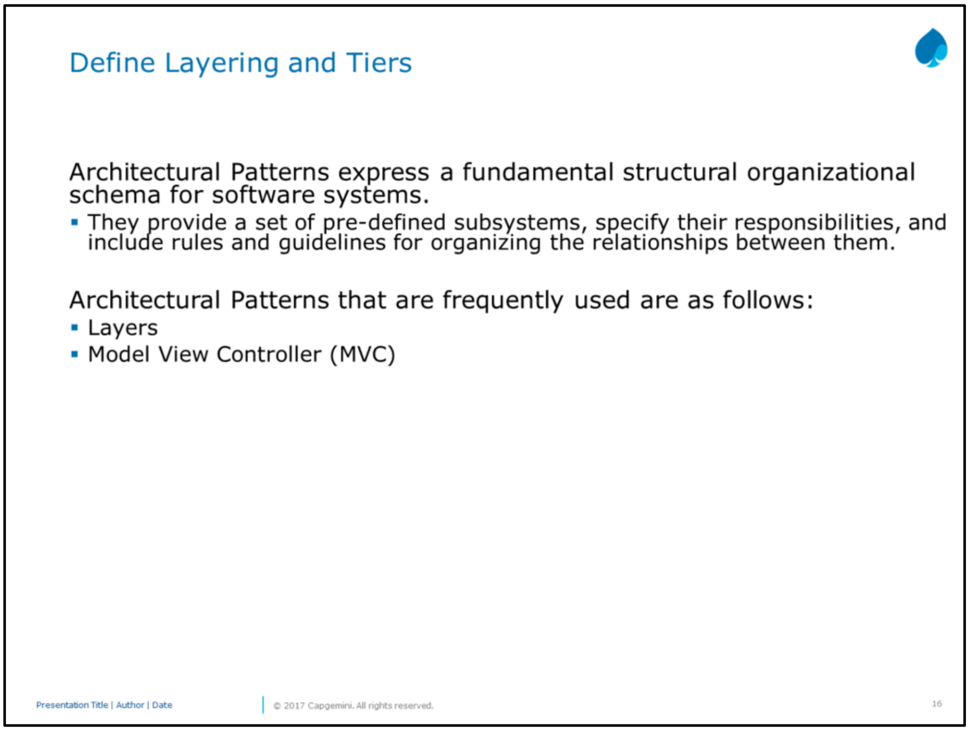
- Several Architectural Approaches & Patterns exist. An architect has to evaluate the ones that best suit the requirements of the project. Some of these patterns are outlined in upcoming slides.

Instructor Notes:

Since the best practices are already captured as architectural patterns, use of an appropriate architectural pattern helps in efficient designing of the architecture of the system.

Pipes and Blackboards are some of the other architectural patterns.

Qzen recommends Layers and hence we look at that in detail.



Define Layering and Tiers

Architectural Patterns express a fundamental structural organizational schema for software systems.

- They provide a set of pre-defined subsystems, specify their responsibilities, and include rules and guidelines for organizing the relationships between them.

Architectural Patterns that are frequently used are as follows:

- Layers
- Model View Controller (MVC)

Presentation Title | Author | Date

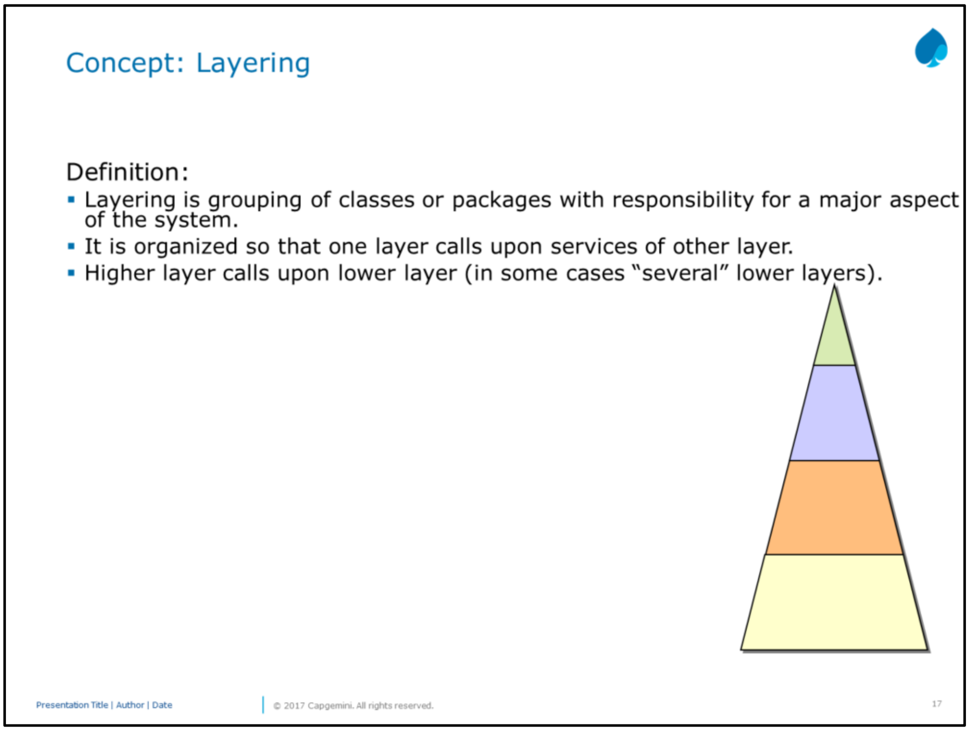
16

<u>Task1: Understand Architecture Requirements</u>:
<u>Step6: Define Layering and Tiers</u>:

- Architecture deals with structural building blocks of the system and the communications across these building blocks. Best Practices in terms of architectural styles are captured through architectural patterns.
- “**Architectural patterns** define the system characteristics, performance characteristics and the process and distribution architectures” (Book: Pattern-Oriented Software Architecture-A System of Patterns; Buschmann et al.)
- Each pattern focuses on solving certain unique set of problems. Frequently used architectural patterns are as follows:
  - **Layers:** The layers pattern is where an application is decomposed into different levels of abstraction. Layering can be responsibility-based or reuse-based.
  - **Model-View-Controller:** The MVC pattern is where an application is divided into three partitions. They are:
    - **Model**, that is the business rules and underlying data
    - **View**, that is how information is displayed to the user
    - **Controllers**, that is the process the user input
- We shall focus on Layers as an Architectural Pattern.

Instructor Notes:

Basic concept is separation of concerns. Each Layer focuses on something specific.



Concept: Layering

Definition:

- Layering is grouping of classes or packages with responsibility for a major aspect of the system.
- It is organized so that one layer calls upon services of other layer.
- Higher layer calls upon lower layer (in some cases “several” lower layers).

Presentation Title | Author | Date

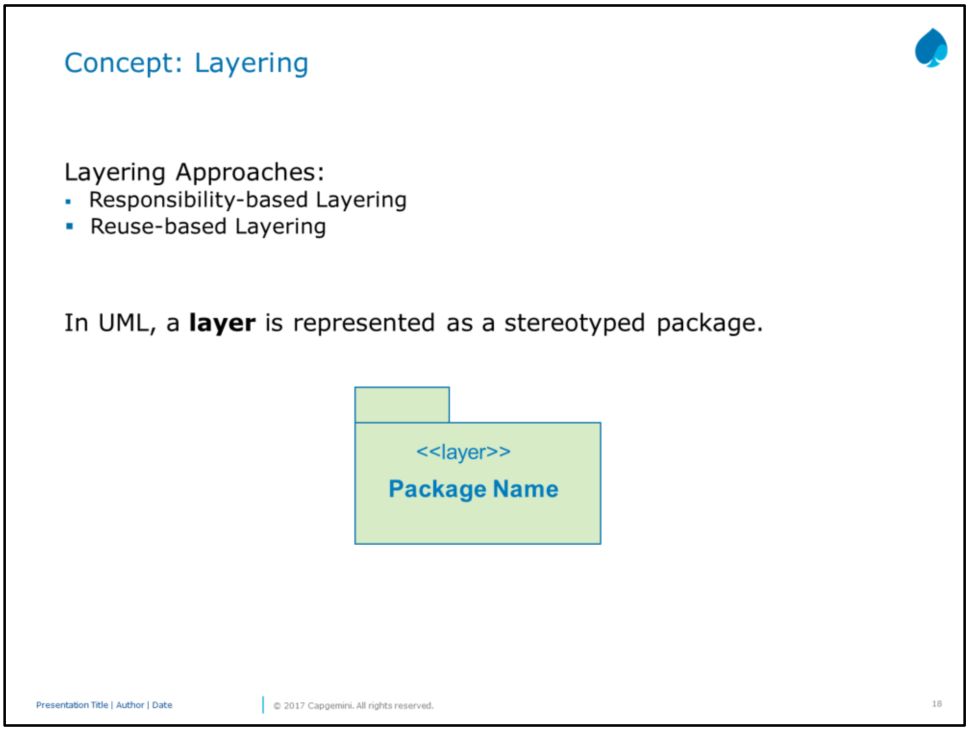
17

<u>Concept: Layering</u>:

- Layers are architectural patterns used in large systems that need a high-level of decomposition. Layers are used in systems that needs to handle issues at different levels of abstraction. For example, hardware control issues, common services issues, domain-specific issues, etc.
- Layering is an ideal solution for systems with the following requirements:
  - Parts of the system need to be replaced.
  - Changes in components should not ripple.
  - Similar responsibilities should be grouped together.
  - Size of complex-components may need to be decomposed.
- Large systems having the need for decomposition can be structured to form groups of components that are arranged in layers one above the other. Upper layers should always use services of layers directly below them only, and not the other way round. This should be mandatory. No layer should be skipped, unless intermediate layers only contain pass-through components.

Instructor Notes:

Focus on the UML notation.



Concept: Layering

Layering Approaches:

- Responsibility-based Layering
- Reuse-based Layering

In UML, a **layer** is represented as a stereotyped package.

<<layer>>
Package Name

Presentation Title | Author | Date

18

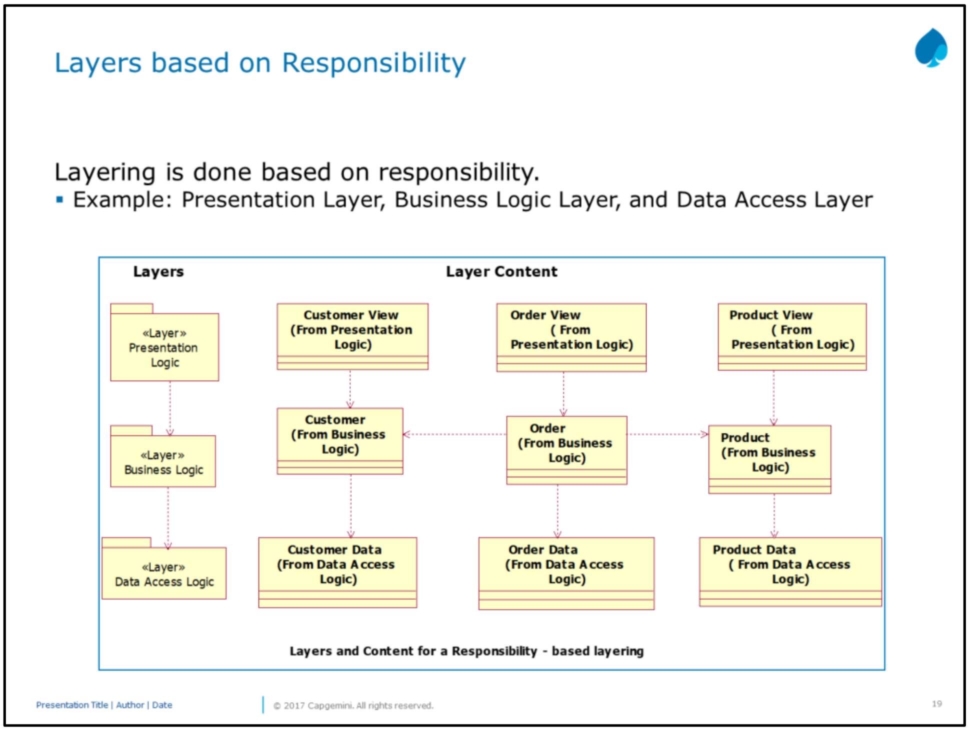
<u>Concept: Layering</u>:

- Two often used approaches for layering are:
  - ➢ Responsibility-based layering
  - ➢ Reuse-based layering
- These are discussed in detail later in the lesson.
- A Layer is nothing but a way of grouping elements at the same level of abstraction. Hence the UML construct Package, which is a general purpose mechanism used to organize elements into groups, is used. It is stereotyped as a Layer.

Instructor Notes:

Participants are usually familiar with responsibility based layering.

Note that names and number of layers could be different based on requirements of the applications.



Layers based on Responsibility

Layering is done based on responsibility.

- Example: Presentation Layer, Business Logic Layer, and Data Access Layer

Layers
Layer Content
«Layer» Presentation Logic
«Layer» Business Logic
«Layer» Data Access Logic
Customer View (From Presentation Logic)
Order View (From Presentation Logic)
Product View (From Presentation Logic)
Customer (From Business Logic)
Order (From Business Logic)
Product (From Business Logic)
Customer Data (From Data Access Logic)
Order Data (From Data Access Logic)
Product Data (From Data Access Logic)

Layers and Content for a Responsibility - based layering

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

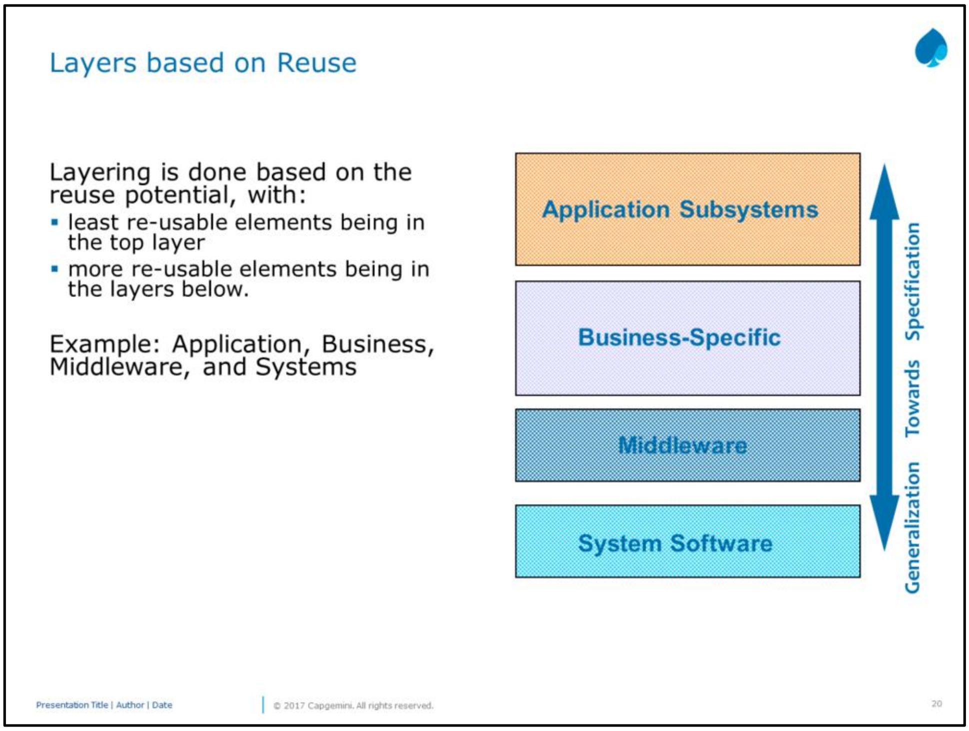
19

<u>Layers based on Responsibility</u>:

- One of the commonly used layering approach is to decompose the application based on responsibilities such as Presentation, Business Logic, Data Access, and so on.
  - ➢ Elements pertaining to the Presentation aspects such as all Forms belong to the Presentation Layer.
  - ➢ Elements pertaining to storing and processing of business date belong to the Business Layer, and so on.

Instructor Notes:

Participants may not usually be very familiar with this approach and hence this needs more explanation.



Layers based on Reuse

Layering is done based on the reuse potential, with:

- least re-usable elements being in the top layer
- more re-usable elements being in the layers below.

Example: Application, Business, Middleware, and Systems

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

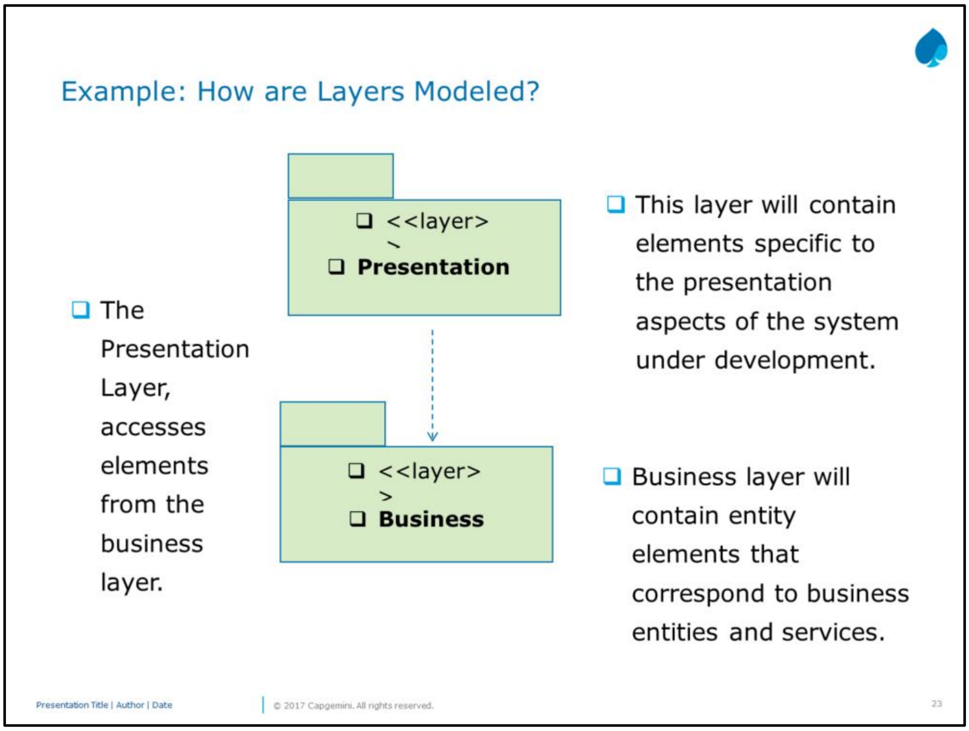
20

<u>Layers based on Reuse</u>:

- Another approach towards decomposing is on the basis of re-use potential. For example, for the layering shown here in the above slide:
  - **Application layer** contains specific services and software specific to the system under development. This is usually a single layer and is the topmost layer. This is specific to the application and hence least re-usable.
  - **Business-specific layer** contains a number of reusable subsystems that are commonly used in similar domains. Hence it is re-usable across the applications of the same domain.
  - **Middleware layer** contains GUI-builders, interfaces to database management systems, platform-independent services for distinguished object-computing in heterogeneous environments and so on. Hence these are more reusable as they are no longer restricted to a domain.
  - **System Software layer** contains the software on which the system runs, for example, operating systems, interfaces to specific hardware devices, and device drivers. They are most reusable since they are core functionalities.

Instructor Notes:

How will MVC be modeled? We still use packages for Model, View and Controller. The stereotype is not needed and packages themselves can be named as Model, View and Controller. Dependencies between the three will need to be modeled based on the MVC architectural pattern.



Example: How are Layers Modeled?

❑ <<layer>
>
❑ **Presentation**

❑ <<layer>
>
❑ **Business**

❑ The Presentation Layer, accesses elements from the business layer.

❑ This layer will contain elements specific to the presentation aspects of the system under development.

❑ Business layer will contain entity elements that correspond to business entities and services.

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

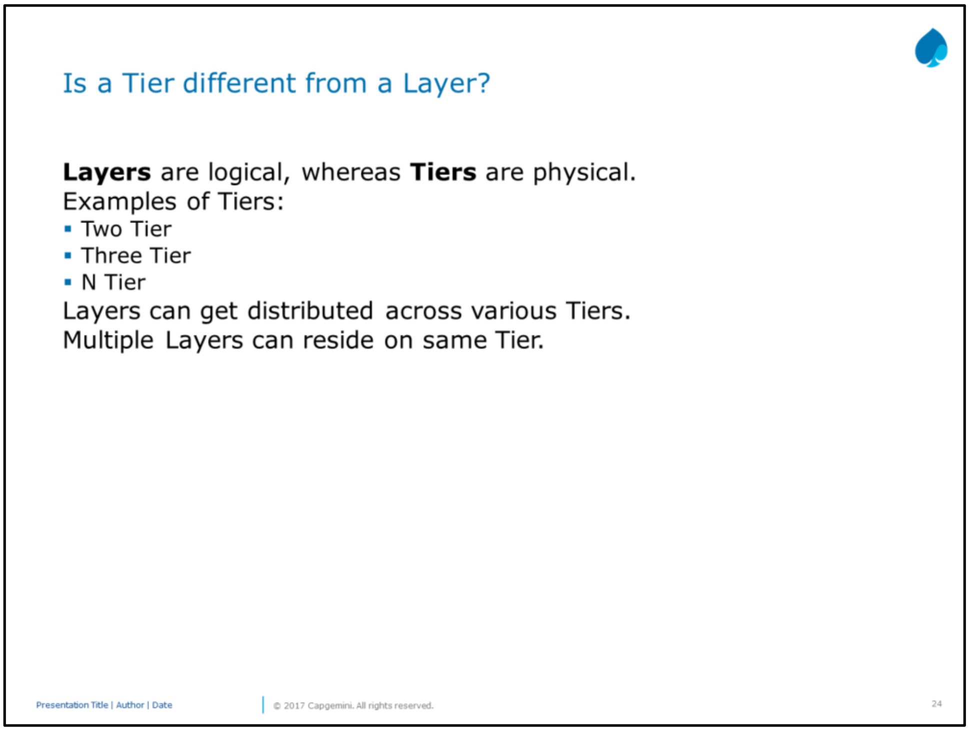
23

<u>Example: How are Layers Modeled</u>?

- In a UML project, Layers can be modeled as shown in the above slide.
  A **dependency relationship** across the layers indicates that the layer above uses the elements of the layer below.
- In the example in the slide:
  - ➢ The top most layer contains elements and software that is specific to presentation.
  - ➢ Key abstractions and services common to several systems are encapsulated in the business layer and layers below. These elements are accessible to the presentation layer.

Instructor Notes:

Literatures tend to use these two terms interchangeably. From a design perspective, we are more in the realm of logical behaviour and tend to go with layers.



Is a Tier different from a Layer?

Layers are logical, whereas **Tiers** are physical.
Examples of Tiers:

- Two Tier
- Three Tier
- N Tier

Layers can get distributed across various Tiers.
Multiple Layers can reside on same Tier.

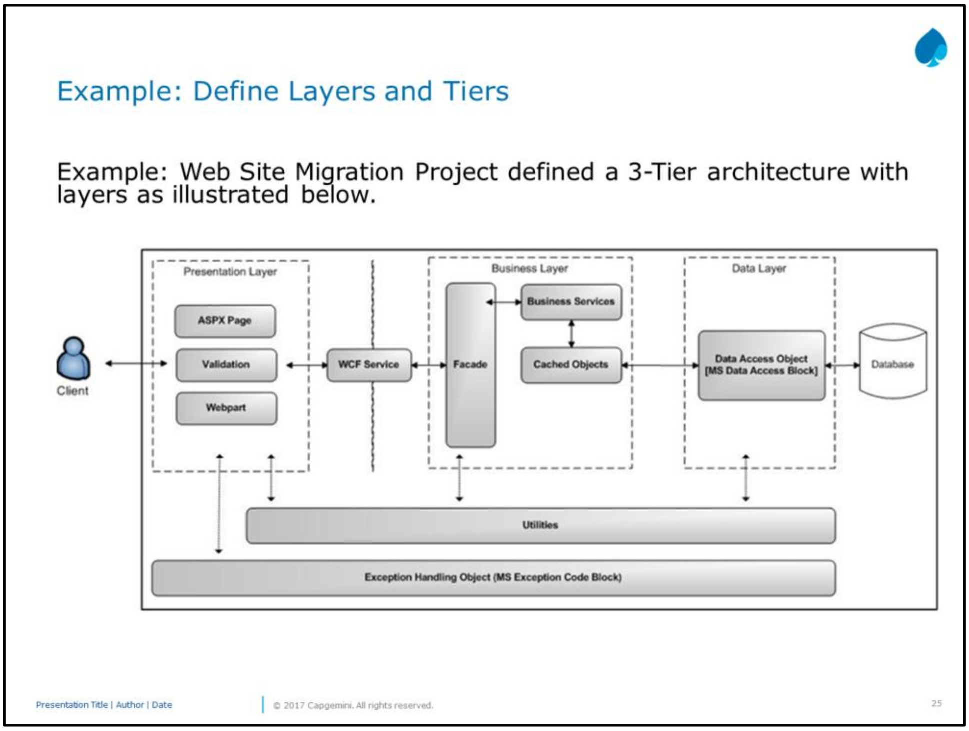
Presentation Title | Author | Date | © 2017 Capgemini. All rights reserved. 24

<u>Is a Tier different from a Layer</u>?

- **Layers** and **Tiers** tend to get used interchangeably. However, layering typically refers to logical separation of code, whereas tiers refers to code separation on network or process boundaries.
- We have come across different examples of Tiered architecture, such as Two-Tier as in a Client-Database separation, Three-Tier as in a Client-Web Server-Database separation, or N-Tier as in Client-Web Server-Application Server-Database distribution.
- Since a tier is a physical distribution pattern, there can possibly be one or more layers allocated to same tier or a layer spanning multiple tiers.

Instructor Notes:

Example from a Project on Web Site Migration



Example: Define Layers and Tiers

Example: Web Site Migration Project defined a 3-Tier architecture with layers as illustrated below.

Presentation Title | Author | Date

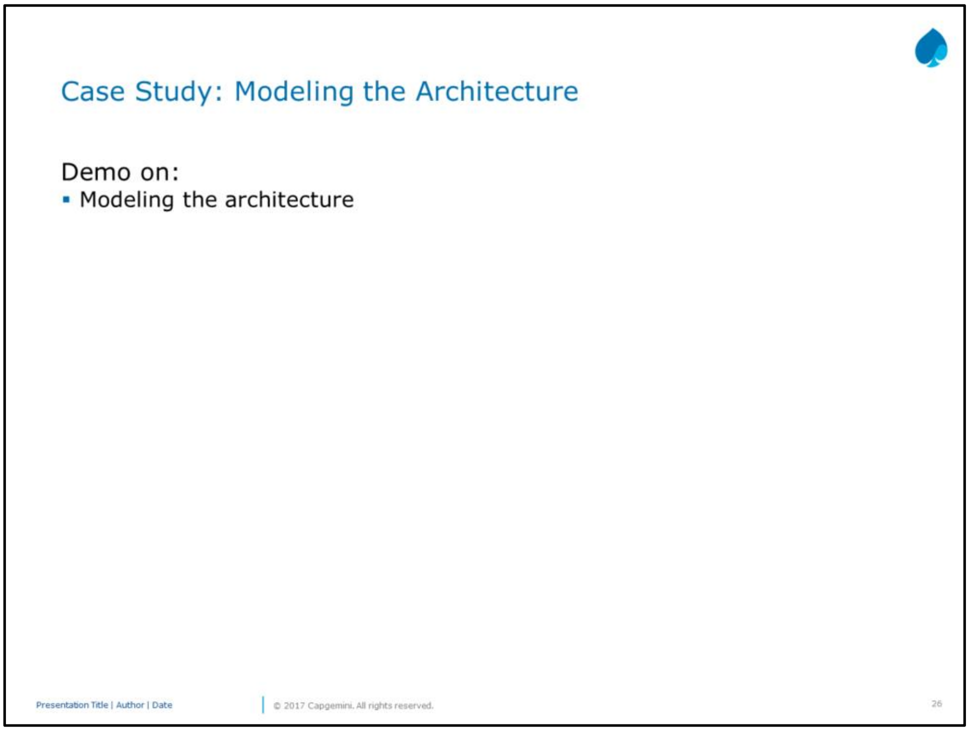
© 2017 Capgemini. All rights reserved.

25

<u>Example: Define Layers and Tiers</u>:
The slide shows an example of how tiers and layers are defined for a web site migration project.

Instructor Notes:

Add an Analysis Model to the project. Now include the architecture diagram for the project in this model.



Case Study: Modeling the Architecture

Demo on:

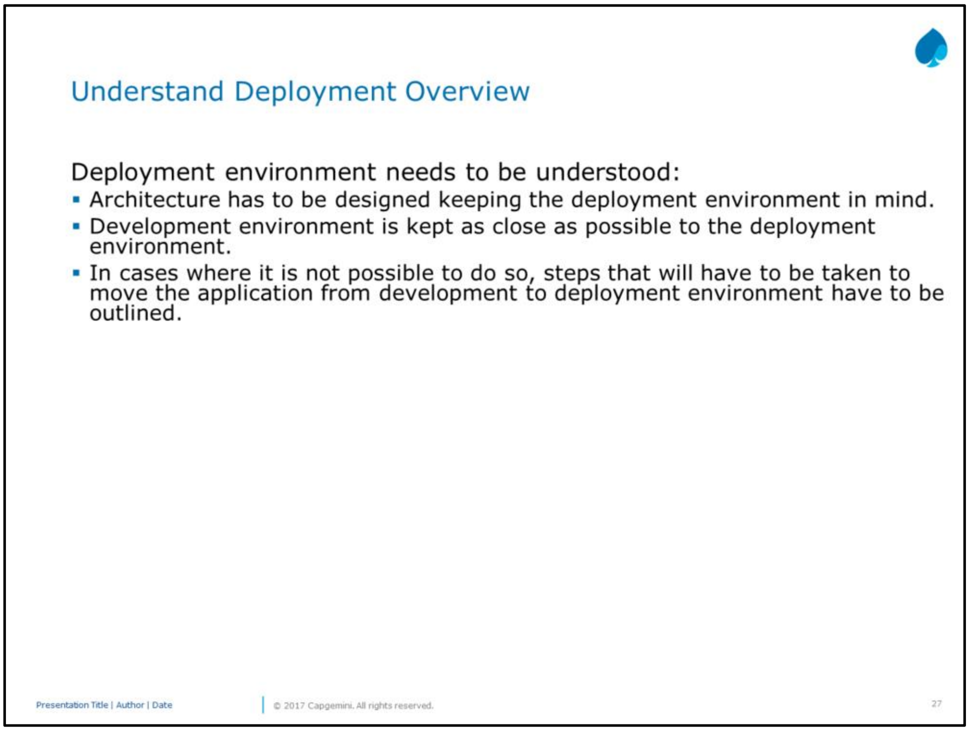
- Modeling the architecture

Presentation Title | Author | Date

26

Instructor Notes:

The IMS group will also come into picture when deployment architectures are discussed.



Understand Deployment Overview

Deployment environment needs to be understood:

- Architecture has to be designed keeping the deployment environment in mind.
- Development environment is kept as close as possible to the deployment environment.
- In cases where it is not possible to do so, steps that will have to be taken to move the application from development to deployment environment have to be outlined.

Presentation Title | Author | Date

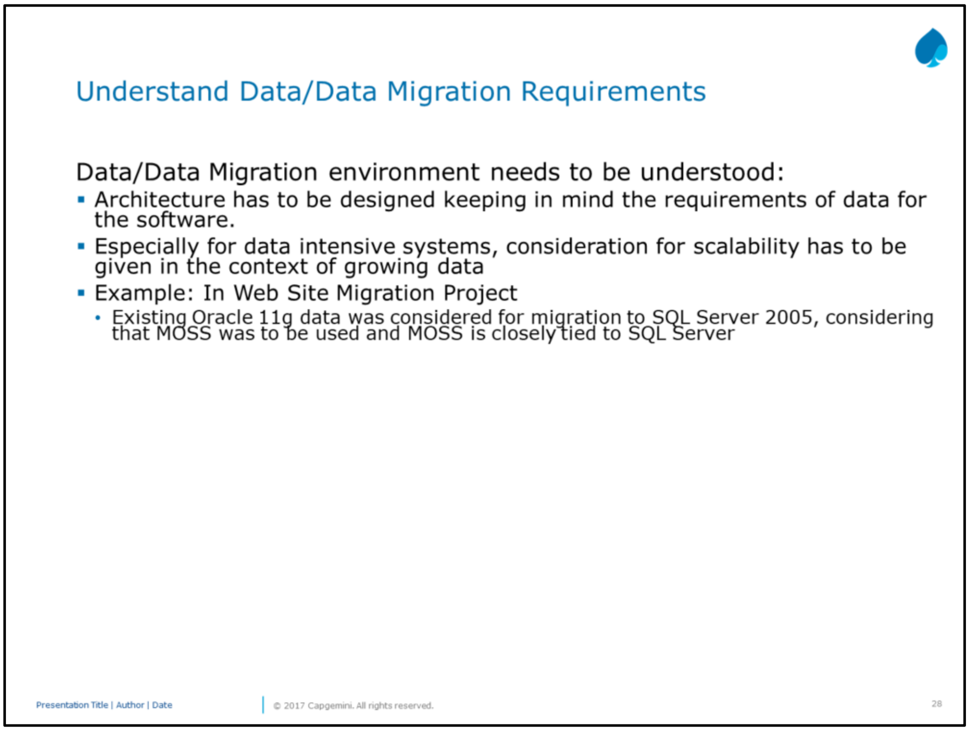
27

Task1: Understand Architecture Requirements:
Step7: Understand Deployment Overview:

- Deployment view is especially important for distributed systems. Deployment view focuses on the hardware environment needed to deploy the application. Design and development of the application packages and components need to be done keeping in mind the deployment architecture.
- Ideally, the **development environment** can be a replica of the **deployment environment**. However, when such a scenario is not possible, thought has to be given to how the application will be moved from development to deployment environment (for example, any changes to code, configuration settings, any best practices, and so on).

Instructor Notes:

The IMS group will also come into picture when deployment architectures are discussed.



Understand Data/Data Migration Requirements

Data/Data Migration environment needs to be understood:

- Architecture has to be designed keeping in mind the requirements of data for the software.
- Especially for data intensive systems, consideration for scalability has to be given in the context of growing data
- Example: In Web Site Migration Project
  - Existing Oracle 11g data was considered for migration to SQL Server 2005, considering that MOSS was to be used and MOSS is closely tied to SQL Server

Presentation Title | Author | Date

28

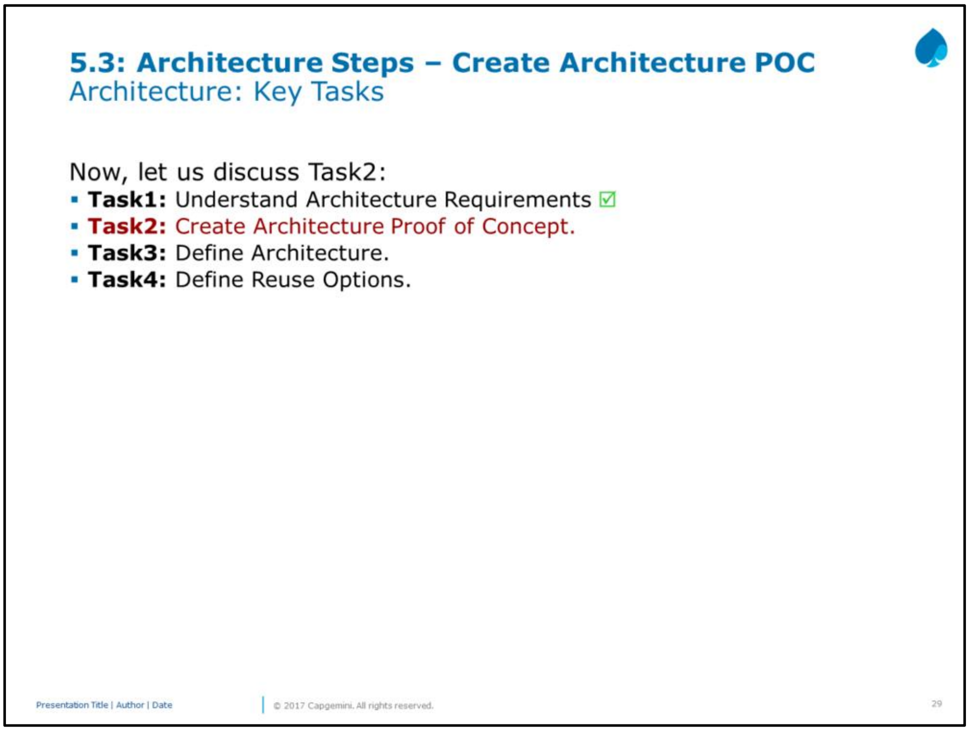
<u>Task1: Understand Architecture Requirements</u>:

<u>Step8: Understand Data/ Data Migration Requirements</u>:

- This will drive the data architecture of the system
- Special attention will have to be given to data intensive systems, and how application will handle the growing data for the system

Instructor Notes:

Participants usually know about POCs. Encourage discussion before moving to next slide.



5.3: Architecture Steps – Create Architecture POC

Architecture: Key Tasks

Now, let us discuss Task2:

- **Task1:** Understand Architecture Requirements ☑
- **Task2:** Create Architecture Proof of Concept.
- **Task3:** Define Architecture.
- **Task4:** Define Reuse Options.

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

29

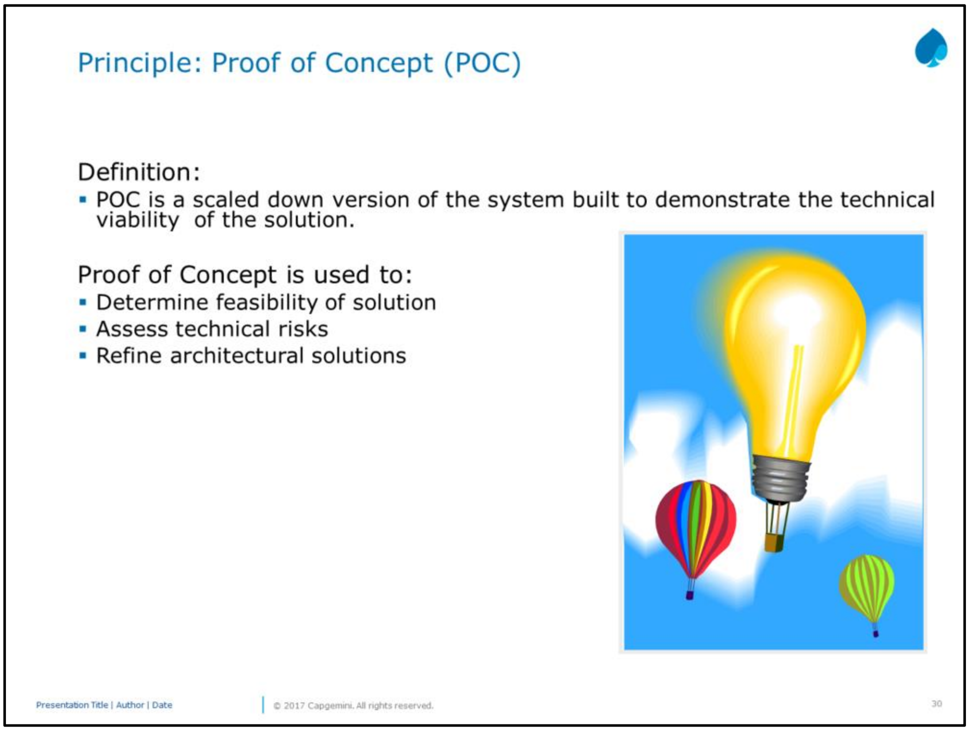
Architecture: Key Tasks:

- We have looked at the steps of the task “Understand Architecture Requirements”. Now, let us understand the steps of “Create Architecture Proof of Concept”.

Instructor Notes:

POCs are very useful and it is recommended to have one or more POCs before baselining a proposed architectural solution.

POC and Prototype tend to get used interchangeably. POC is usually to test a specific concept/idea whereas Prototype is usually used in a broader sense for the more complete artifact.



Principle: Proof of Concept (POC)

Definition:

- POC is a scaled down version of the system built to demonstrate the technical viability of the solution.

Proof of Concept is used to:

- Determine feasibility of solution
- Assess technical risks
- Refine architectural solutions

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

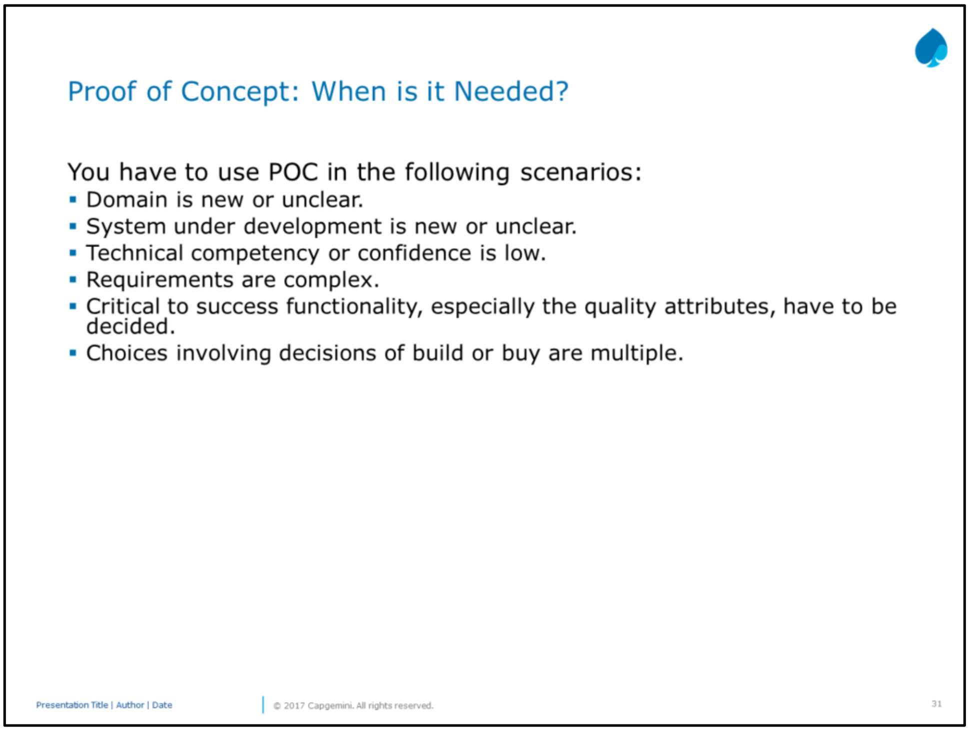
30

<u>Principle: Proof of Concept (POC)</u>:

- Many a times, it is necessary to demonstrate the feasibility of a proposed solution before actually going ahead with the solution. POCs help in realizing proposed ideas.
- One or more POCs can be created to validate different aspects of the proposed solution. At times, a POC can be a stepping stone to the development of a full fledged prototype.
- POC can take the form of either of the following:
  - ➢ A conceptual solution
  - ➢ A simulation of a solution
  - ➢ An executable prototype
- POCs are of course not restricted to software systems! We can see it in fields as diverse as Engineering, Film making, and Business. For example, POCs were created for the film *Finding Nemo* to explore various animation techniques.

Instructor Notes:

POCs need not be done when domain/system is known, risks are understood and requirements are well defined.



Proof of Concept: When is it Needed?

You have to use POC in the following scenarios:

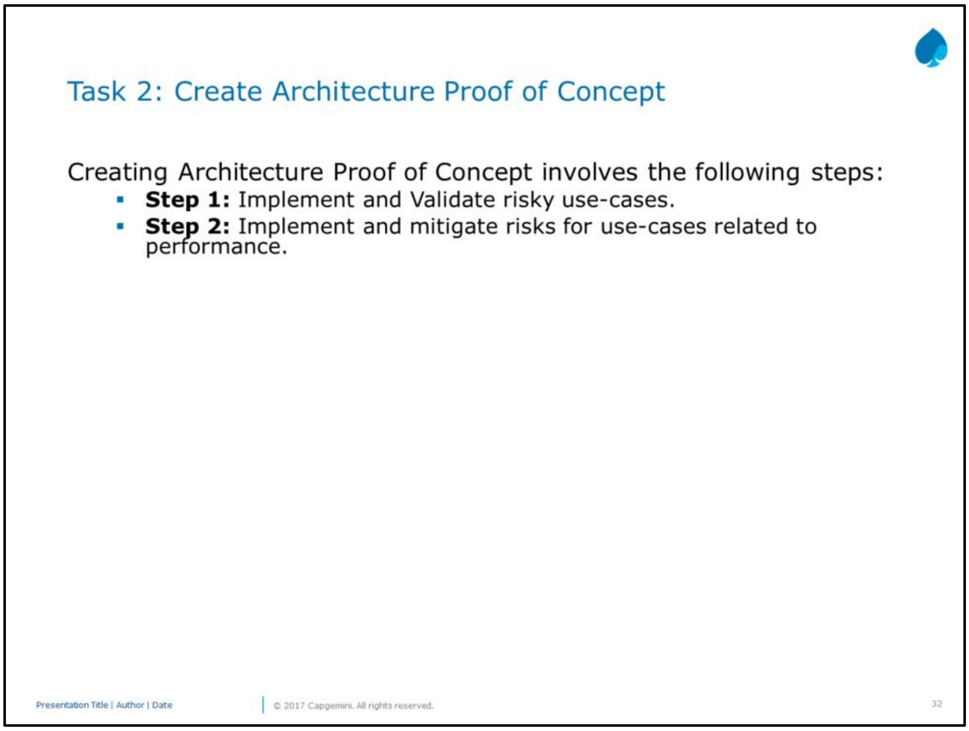
- Domain is new or unclear.
- System under development is new or unclear.
- Technical competency or confidence is low.
- Requirements are complex.
- Critical to success functionality, especially the quality attributes, have to be decided.
- Choices involving decisions of build or buy are multiple.

<u>Proof of Concept: When is it Needed</u>?

- If the domain is unfamiliar, POCs help in understanding and clarifying requirements.
- Similarly, if such a kind of application has not been built before even if it is a familiar domain, then POCs help.
- POCs are useful when implementation in a new technology or using a new methodology is required and the development team does not have that expertise.
- POCs help understand complex requirements and explore alternate solutions.
- POCs are especially useful to arrive at possible solutions for quality attributes.
- Many a times, a choice has to be made between building or buying a component. Here too, a POC helps.

Instructor Notes:

Here are the steps of the second task viz. Create Architecture Proof of Concept.



Task 2: Create Architecture Proof of Concept

Creating Architecture Proof of Concept involves the following steps:

- **Step 1:** Implement and Validate risky use-cases.
- **Step 2:** Implement and mitigate risks for use-cases related to performance.

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

32

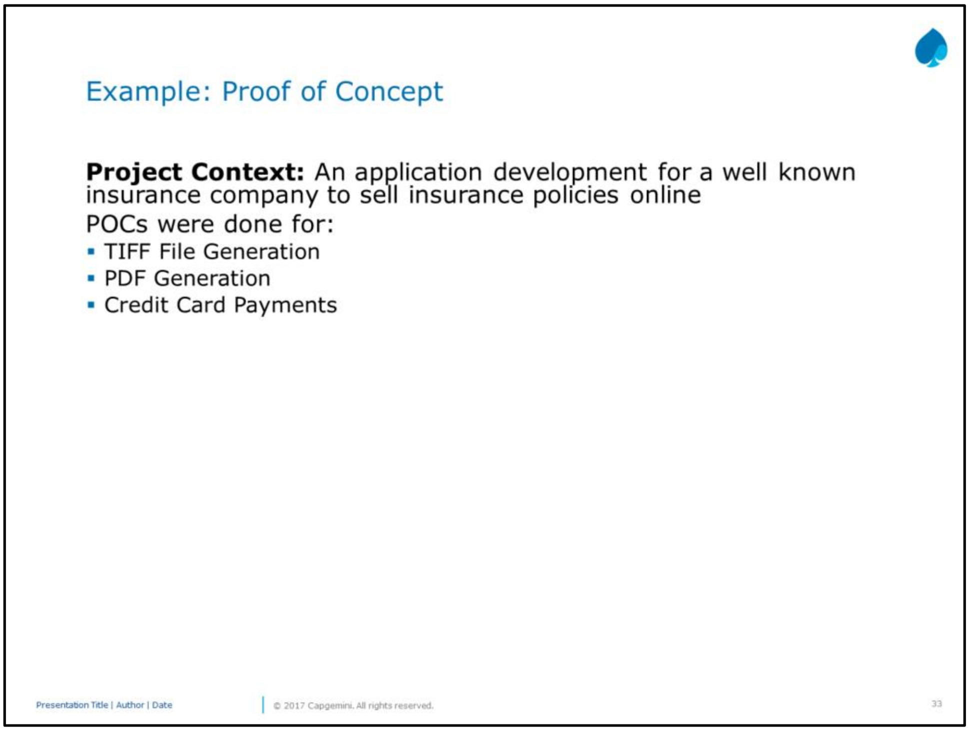
Task2: Create Architecture Proof of Concept:

Steps: Proof of Concept:

- The steps involve identifying the right use-cases for creating the POCs based on proposed ideas.
- POCs are particularly useful to arrive at solutions for the non-functional requirements such as Performance and Security.

Instructor Notes:

This is a Project example and benefit that the team got by using POCs.



Example: Proof of Concept

Project Context: An application development for a well known insurance company to sell insurance policies online

POCs were done for:

- TIFF File Generation
- PDF Generation
- Credit Card Payments

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

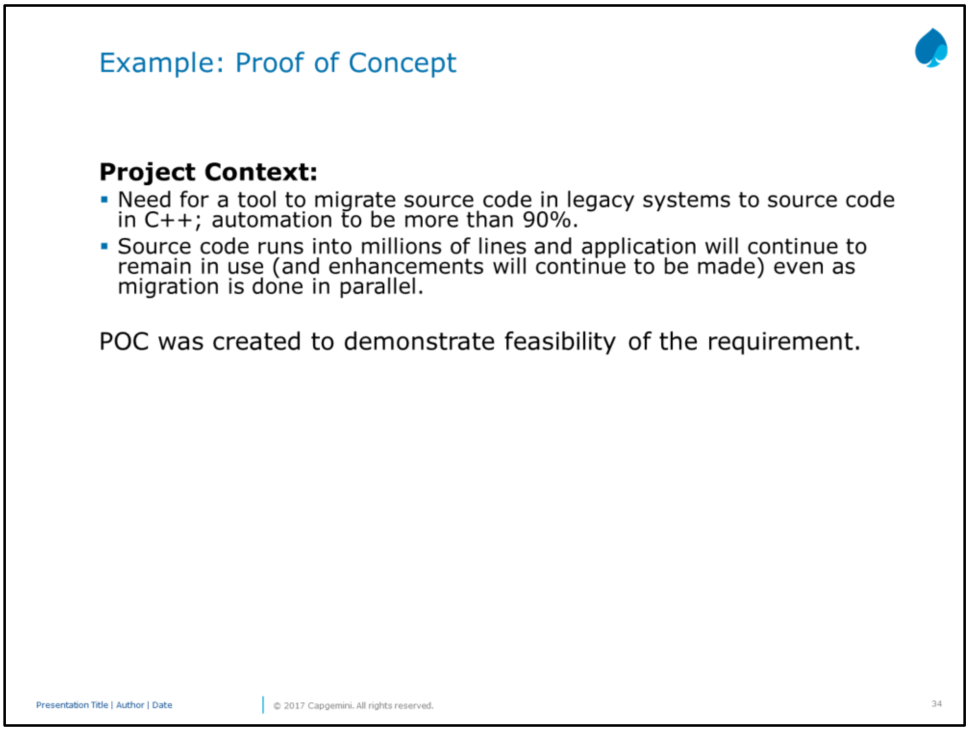
33

Example 1: Proof of Concept:

- In the example in the above slide, the areas for POCs were identified considering the team's knowledge and experience in those areas. External members were engaged in developing POCs since the project team members were occupied.
- POC brought forth a feasibility issue relating to size of TIFF file. A solution to the same was arrived at during the coding phase.
- POCs helped this project team to:
  - Mitigate risks early in the project cycle
  - Get sufficient time to work on technically complex areas
  - Make necessary provisions in design

Instructor Notes:

In this case, the POC won accolades from the client and has acted like a door-opener for other opportunties.



Example: Proof of Concept

Project Context:

- Need for a tool to migrate source code in legacy systems to source code in C++; automation to be more than 90%.
- Source code runs into millions of lines and application will continue to remain in use (and enhancements will continue to be made) even as migration is done in parallel.

POC was created to demonstrate feasibility of the requirement.

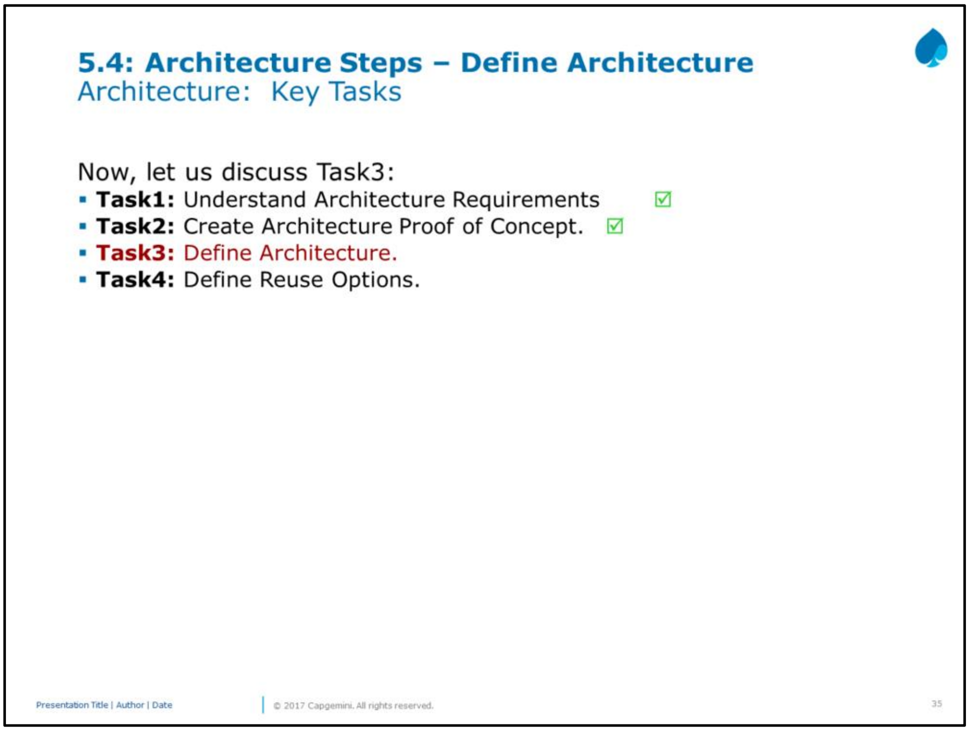
Presentation Title | Author | Date | © 2017 Capgemini. All rights reserved. 34

<u>Example 2: Proof of Concept</u>:

- This project had an interesting and challenging scenario. Not much expertise was available on the legacy platform. Besides, client wanted a high level of automation into a platform which was quite different from the legacy platform.
- The POC creation itself took several months. It also led to multiple refinements in the proposed approach. Now, the solution is such that the source code migration can be done from any platform to any platform by providing appropriate set of rules.

Instructor Notes:

The focus of this task is to put together the software architecture document.



5.4: Architecture Steps – Define Architecture

Architecture: Key Tasks

Now, let us discuss Task3:

- **Task1:** Understand Architecture Requirements ☑
- **Task2:** Create Architecture Proof of Concept. ☑
- **Task3:** Define Architecture.
- **Task4:** Define Reuse Options.

Presentation Title | Author | Date

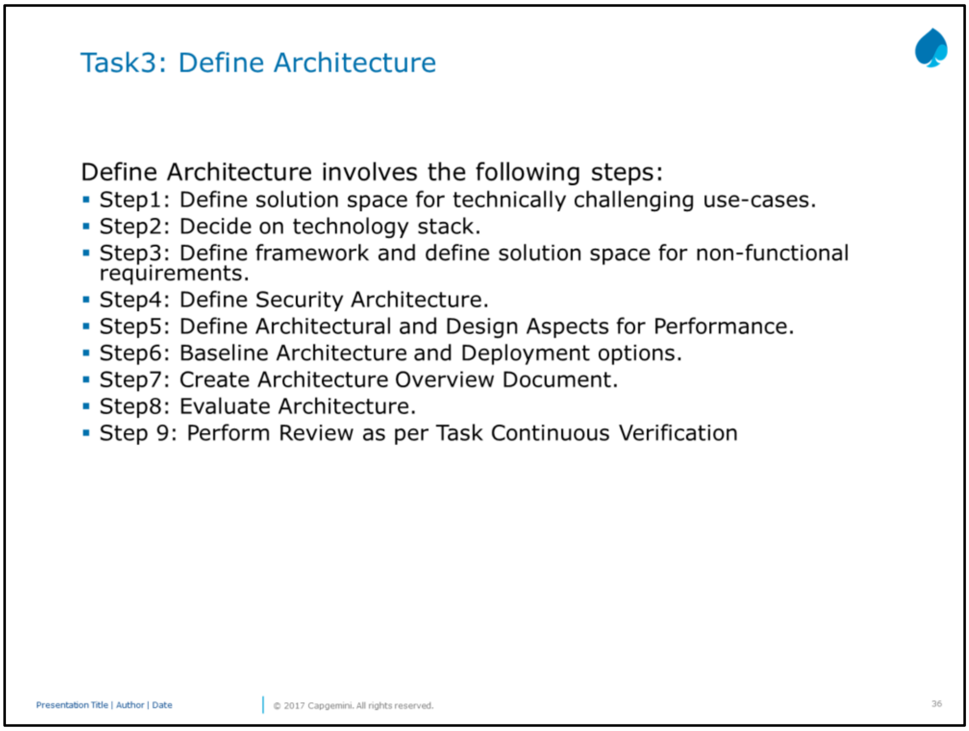
35

<u>Architecture: Key Tasks</u>:

- We have looked at the steps of “Understand Architecture Requirements” and “Create Architecture Proof of Concept”.
- Let us now understand what is done as part of “Define Architecture”.

Instructor Notes:

Let us now look at the steps involved as part of Define Architecture



Task3: Define Architecture

Define Architecture involves the following steps:

- Step1: Define solution space for technically challenging use-cases.
- Step2: Decide on technology stack.
- Step3: Define framework and define solution space for non-functional requirements.
- Step4: Define Security Architecture.
- Step5: Define Architectural and Design Aspects for Performance.
- Step6: Baseline Architecture and Deployment options.
- Step7: Create Architecture Overview Document.
- Step8: Evaluate Architecture.
- Step 9: Perform Review as per Task Continuous Verification

Presentation Title | Author | Date

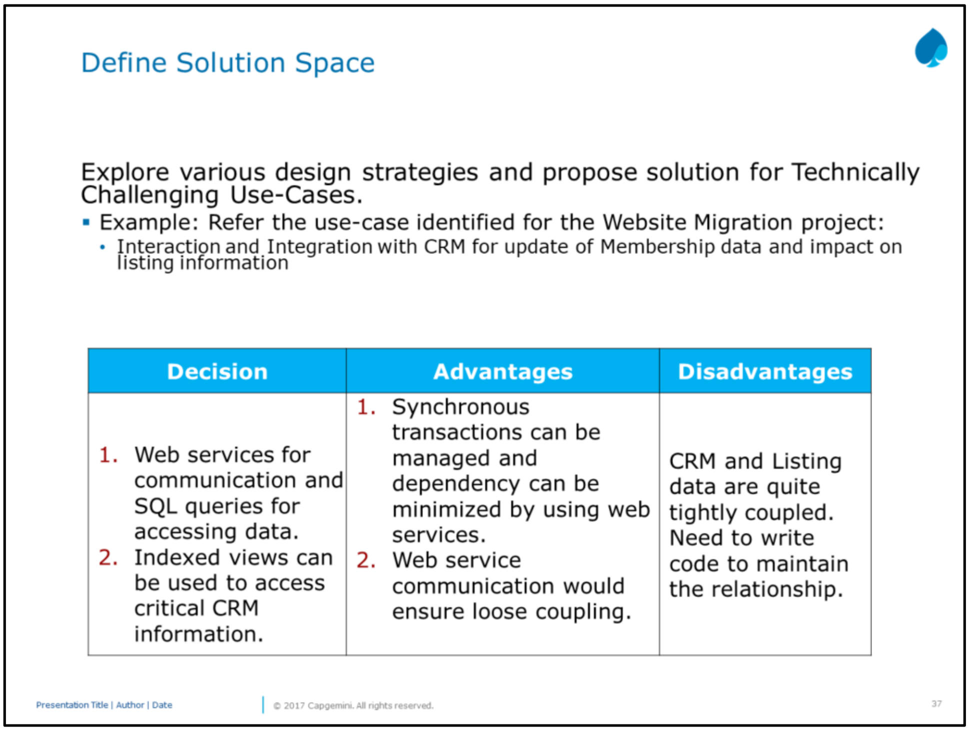
36

Task3: Define Architecture:

- Having understood the architectural requirements, let us now focus on arriving at a solution for the same.
- The decisions taken related to architecture are documented in the software architecture document.

Instructor Notes:

In this example from a Project, key part is the interaction with CRM. That being an external system, this use-case has a bearing on the architecture of the system.



Define Solution Space

Explore various design strategies and propose solution for Technically Challenging Use-Cases.

- Example: Refer the use-case identified for the Website Migration project:
  - Interaction and Integration with CRM for update of Membership data and impact on listing information

Decision	Advantages	Disadvantages
1. Web services for communication and SQL queries for accessing data. 2. Indexed views can be used to access critical CRM information.	1. Synchronous transactions can be managed and dependency can be minimized by using web services. 2. Web service communication would ensure loose coupling.	CRM and Listing data are quite tightly coupled. Need to write code to maintain the relationship.

Presentation Title | Author | Date

37

<u>Task3: Define Architecture</u>:

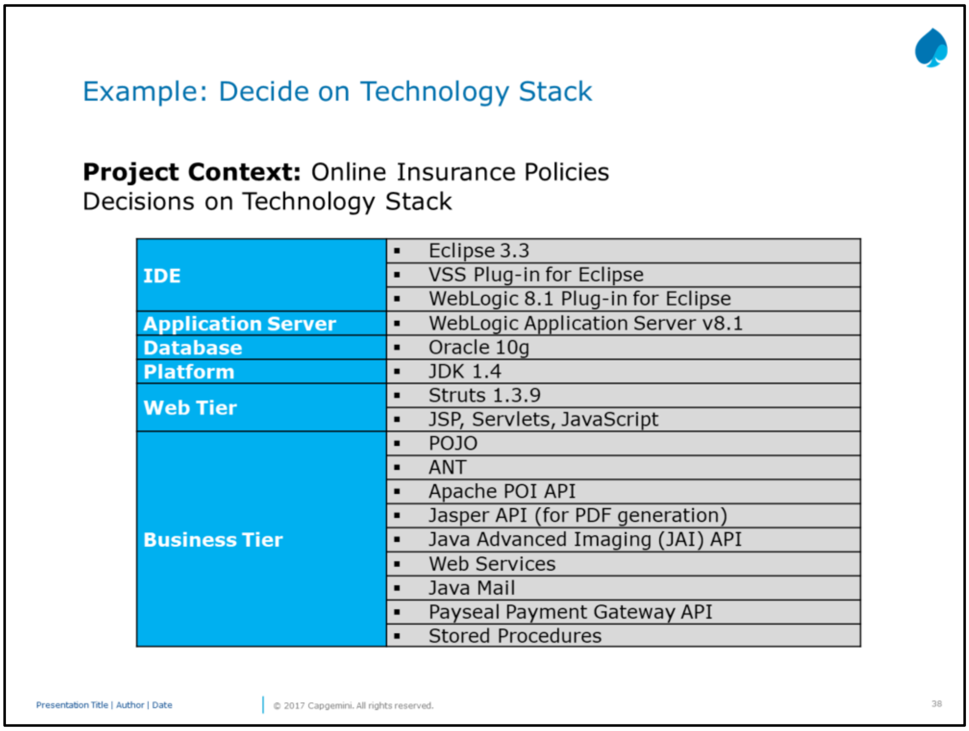
<u>Step1: Define Solution Space for Technically challenging Use-Cases</u>:

- The technically challenging use-cases, especially those that have an impact on the architecture, have been identified earlier. We now look at different design approaches, evaluate them, and propose a solution for the same.
- For the technically challenging use-case identified earlier, the design decision arrived at is as follows:
  - ➢ Use of web services and indexed views.

Instructor Notes:

Deciding on technology stack is one of the key items.

In many cases, this decision is already made by the customer organization! But usually it is made at broader level and details can be thought about at this stage.



Example: Decide on Technology Stack

Project Context: Online Insurance Policies
Decisions on Technology Stack

IDE	▪ Eclipse 3.3
	▪ VSS Plug-in for Eclipse
	▪ WebLogic 8.1 Plug-in for Eclipse
Application Server	▪ WebLogic Application Server v8.1
Database	▪ Oracle 10g
Platform	▪ JDK 1.4
Web Tier	▪ Struts 1.3.9
	▪ JSP, Servlets, JavaScript
Business Tier	▪ POJO
	▪ ANT
	▪ Apache POI API
	▪ Jasper API (for PDF generation)
	▪ Java Advanced Imaging (JAI) API
	▪ Web Services
	▪ Java Mail
	▪ Payseal Payment Gateway API
	▪ Stored Procedures

Presentation Title | Author | Date

38

Task3: Define Architecture:
Step2: Decide on Technology Stack:
Here is an example. Note the variety of technologies that are in use and clear indication of what will be used on which tier.

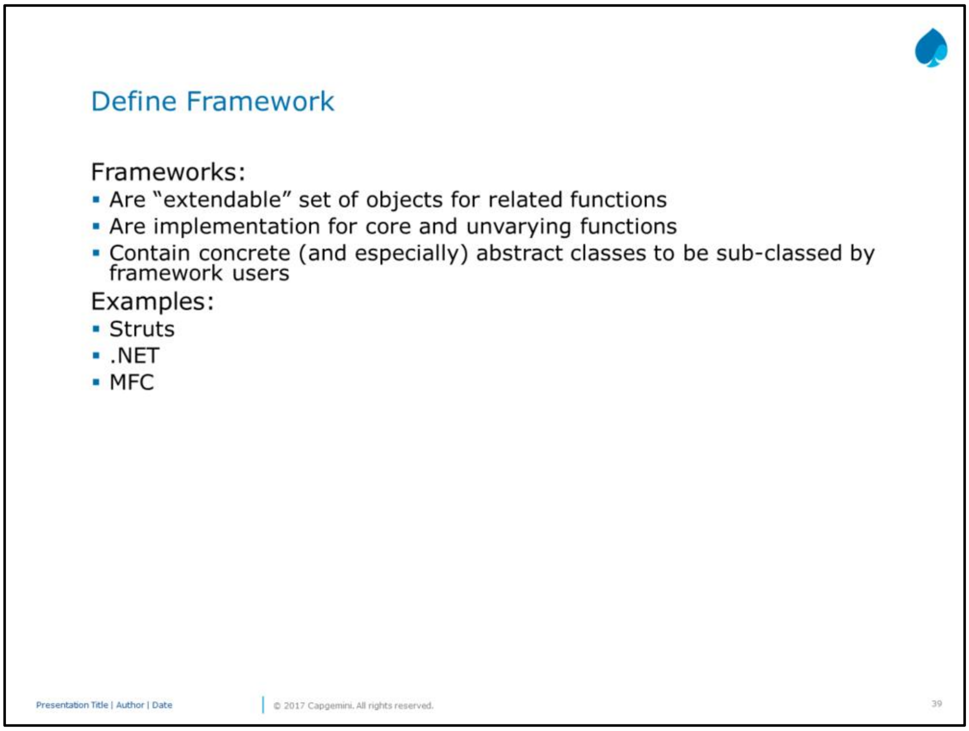
Instructor Notes:

What is the difference between a Framework and a Library?

Framework has a much broader context with certain default behaviours already coded. Extensibility is a key characteristic though the core framework code itself cannot be modified.

Libraries on the other hand focus on a specific functionality without any kind of an overall flow defined.

In a way, set of relating libraries put together with a defined broader objective will lead to a framework.



Define Framework

Frameworks:

- Are "extendable" set of objects for related functions
- Are implementation for core and unvarying functions
- Contain concrete (and especially) abstract classes to be sub-classed by framework users

Examples:

- Struts
- .NET
- MFC

Presentation Title | Author | Date

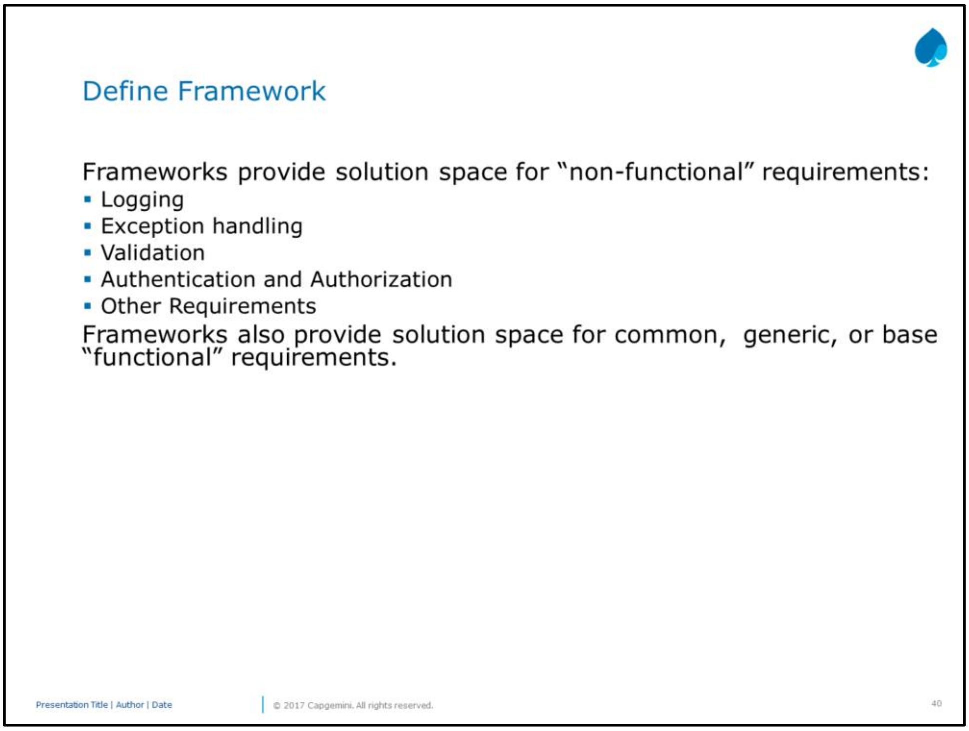
39

Task3: Define Architecture:
Step3: Define Framework and Define Solution Space for Non-functional Requirements:

- Frameworks provide certain behaviors, which can be extended or over-ridden by the users of the framework. Using a framework gives the benefit of productivity because most of the core code and flows needed are already coded. These can be adapted as per requirements.
- Today, there are a variety of frameworks available and choice of the right framework for the given requirements is a critical activity in itself.

Instructor Notes:

Different frameworks will provide different features and one has to choose the right one based on our requirements.



Define Framework

Frameworks provide solution space for “non-functional” requirements:

- Logging
- Exception handling
- Validation
- Authentication and Authorization
- Other Requirements

Frameworks also provide solution space for common, generic, or base “functional” requirements.

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

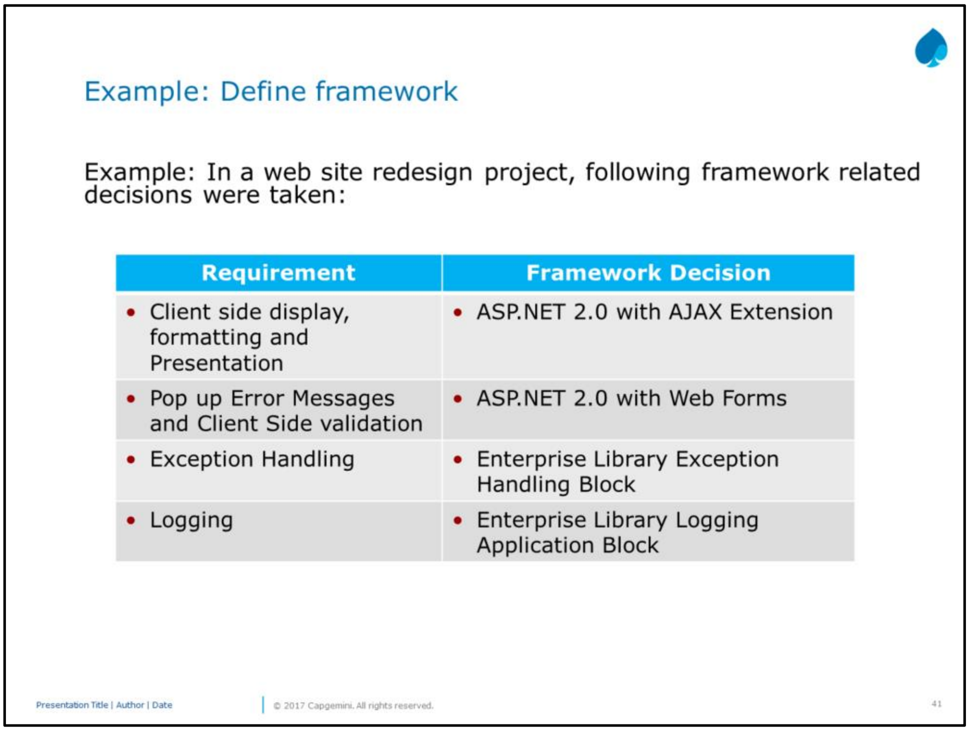
40

Task3: Define Architecture:
Step3: Define Framework and Define Solution Space for Non-functional Requirements:

- Many frameworks provide in-built solutions for non-functional requirements as well as generic functional requirements.
- By using a specific framework, we can take advantage of the existing behavior and extend or over-ride the behavior as needed.

Instructor Notes:

The Patni Enterprise Library provides solutions to some of the common required mechanisms. Not sure which team has created this; but it has been used by this project. Microsoft Enterprise Library also provides these features.



Example: Define framework

Example: In a web site redesign project, following framework related decisions were taken:

Requirement	Framework Decision
• Client side display, formatting and Presentation	• ASP.NET 2.0 with AJAX Extension
• Pop up Error Messages and Client Side validation	• ASP.NET 2.0 with Web Forms
• Exception Handling	• Enterprise Library Exception Handling Block
• Logging	• Enterprise Library Logging Application Block

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

41

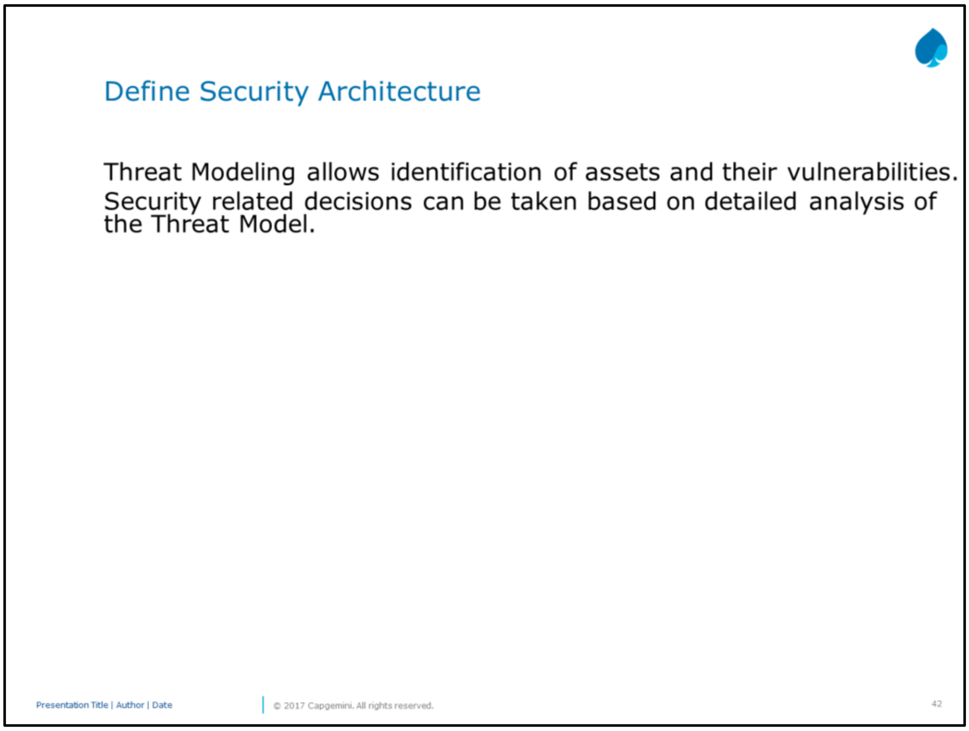
<u>Example: Define Framework and Define Solution Space for Non-functional Requirements</u>:

- Here is an example of framework decisions taken.
- At times, there would also be the build or buy decisions for frameworks. Building a framework is a lot of efforts and the investment is worth it if there is a well defined ROI that we can get on this. Frameworks can be built for specific customers or domains and can be leveraged for similar projects that will get executed for the customer or domain.

Instructor Notes:

Detailed guidelines on Threat Modeling is available in Qzen.

Many of these decisions get implemented at the code level, hence these details have to flow into the coding phase.



Define Security Architecture

Threat Modeling allows identification of assets and their vulnerabilities. Security related decisions can be taken based on detailed analysis of the Threat Model.

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

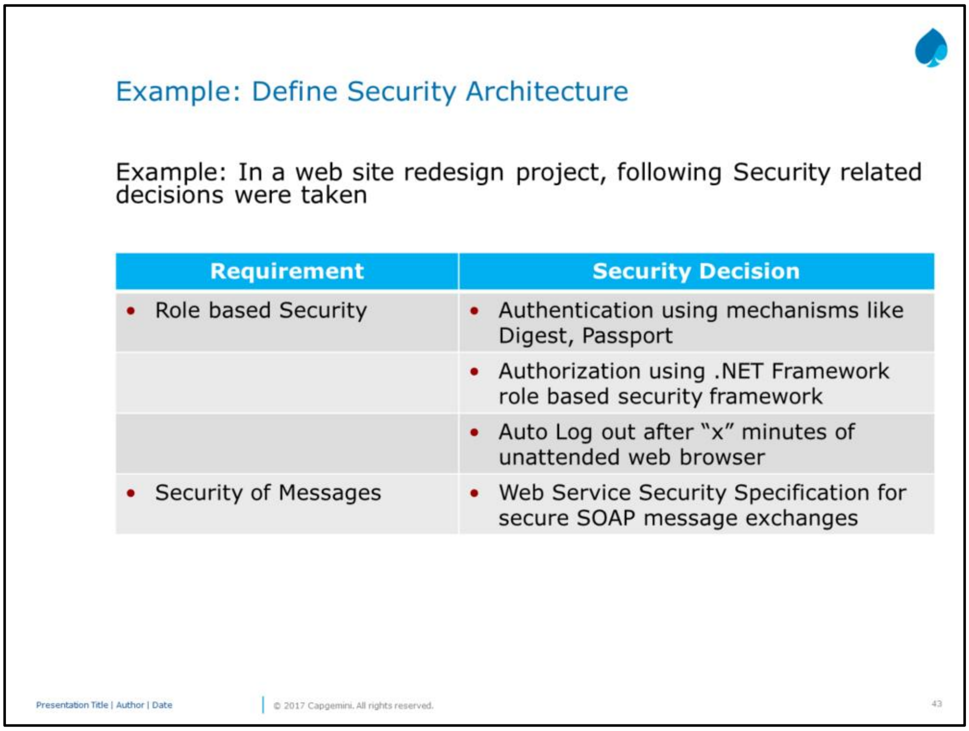
42

<u>Task3: Define Architecture</u>:
<u>Step4: Define Security Architecture</u>:

- **Threat Modeling** is a formal process that allows identification of security vulnerabilities for the assets. As part of threat modeling:
  - ➢ The security objectives are understood
  - ➢ An asset list gets created
  - ➢ Potential threats and vulnerabilities are identified for the assets
- There are models like **STRIDE** or **Threat Category list** that help identify all potential threats.
- Based on the analysis of the threat model, the security related decisions can be taken.
- Refer Patni PLUS for more details on Threat Modeling.

Instructor Notes:

Apart from these design level decisions, many other aspects to take care of the vulnerabilities will have to be implemented in the code.



Example: Define Security Architecture

Example: In a web site redesign project, following Security related decisions were taken

Requirement	Security Decision
• Role based Security	• Authentication using mechanisms like Digest, Passport
	• Authorization using .NET Framework role based security framework
	• Auto Log out after "x" minutes of unattended web browser
• Security of Messages	• Web Service Security Specification for secure SOAP message exchanges

Presentation Title | Author | Date

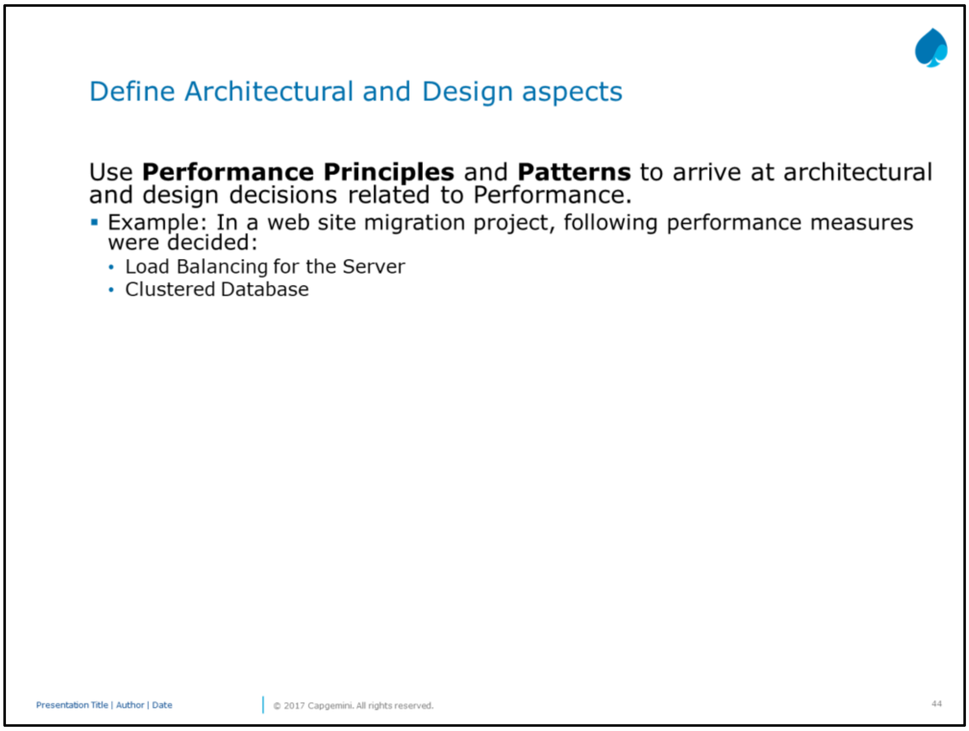
43

Task3: Define Architecture:
Step4: Define Security Architecture:
Example:

- The above slide provides an example on how security decisions are taken.
- For role based security, **authentication** ensures that the right people get access to the system. Further, **authorization** ensures that right permissions are assigned for users to use the right features of the system.

Instructor Notes:

Detailed guidelines on are available in Qzen



Define Architectural and Design aspects

Use **Performance Principles** and **Patterns** to arrive at architectural and design decisions related to Performance.

- Example: In a web site migration project, following performance measures were decided:
  - Load Balancing for the Server
  - Clustered Database

Presentation Title | Author | Date © 2017 Capgemini. All rights reserved. 44

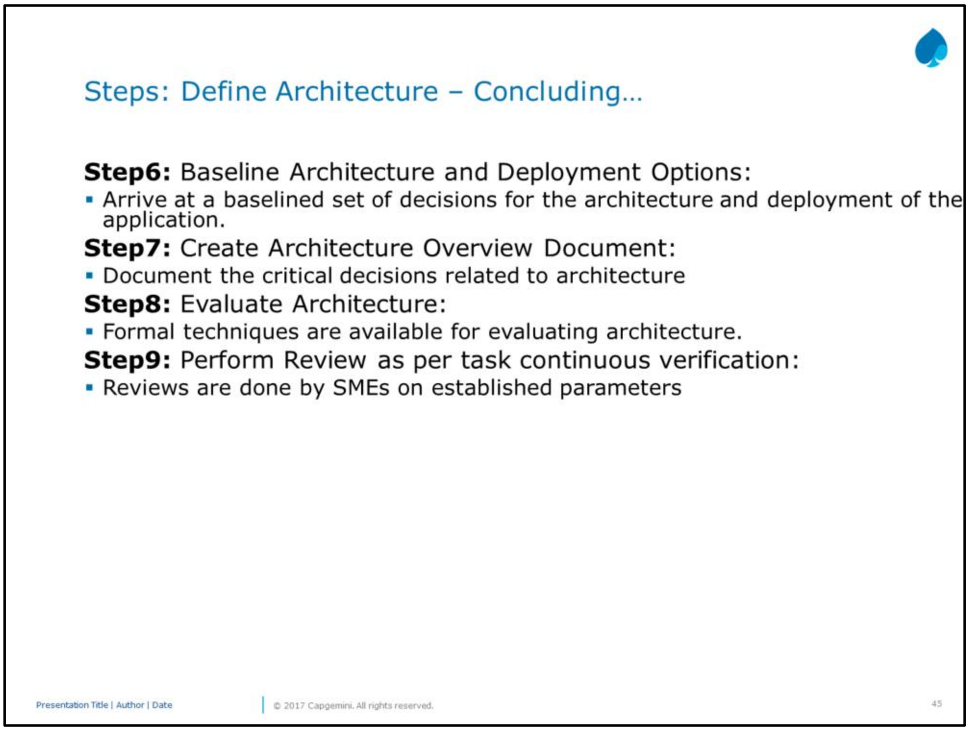
Task3: Define Architecture:

Step5: Define Architectural and Design aspects for Performance:

- There are certain universal principles (that is, principles not specific to a technology) related to Performance which can be used across all stages of the project lifecycle. Further, there are Performance patterns which allow incorporation of best practices pertaining to performance.
- Performance of an application depends not only on decisions taken during architecture and design, but also on how it is coded and infrastructure is used to deploy the application.
- Refer Qzen performance guidelines for more details.

Instructor Notes:

Walk through the software architecture document.



Steps: Define Architecture – Concluding…

Step6: Baseline Architecture and Deployment Options:
- Arrive at a baselined set of decisions for the architecture and deployment of the application.

Step7: Create Architecture Overview Document:
- Document the critical decisions related to architecture

Step8: Evaluate Architecture:
- Formal techniques are available for evaluating architecture.

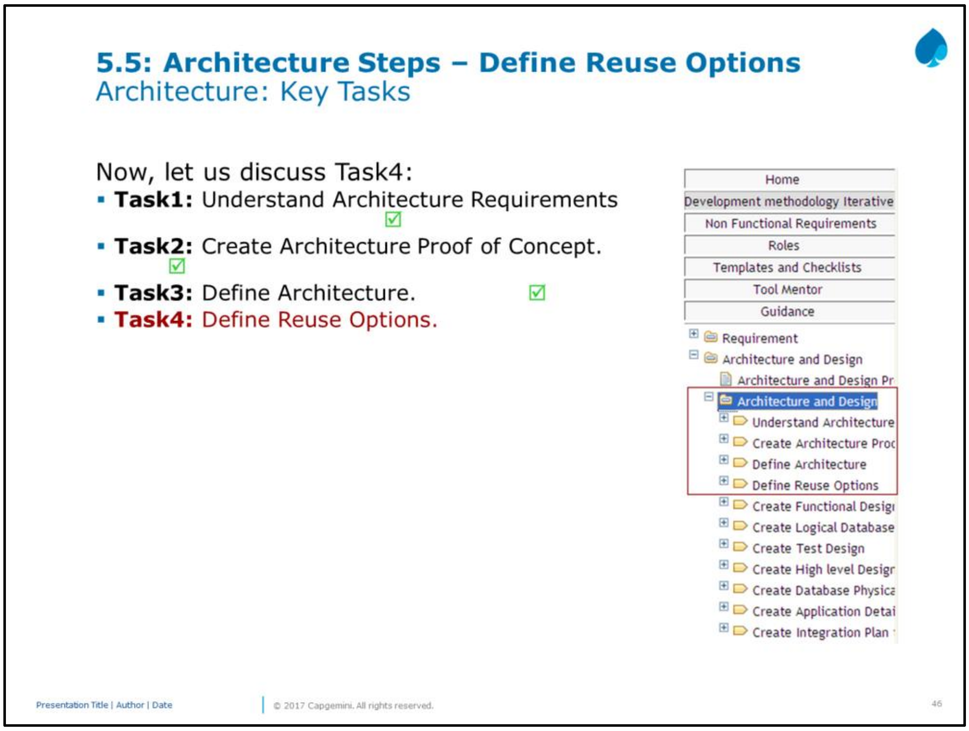
Step9: Perform Review as per task continuous verification:
- Reviews are done by SMEs on established parameters

Presentation Title | Author | Date © 2017 Capgemini. All rights reserved. 45

Task3: Define Architecture:
- The above slide enlists the final set of steps to conclude the task related to Define Architecture. All the decisions taken related to Architecture are consolidated into a software architecture document.
- Formal evaluation techniques are available for architecture. These can be used to review and refine the architecture. These include ATAM (Architecture Trade Off Analysis Method, SAAM (Software Architecture Analysis Method). FMEA (Failure Mode and Effect Analysis) techniques are also used to analyze the effective of software architecture.
- The template for software architecture document is available on Qzen.

Instructor Notes:

We would already have this in mind when we choose frameworks…an exclusive focus here for reuse options.



5.5: Architecture Steps – Define Reuse Options

Architecture: Key Tasks

Now, let us discuss Task4:

- **Task1:** Understand Architecture Requirements ☑
- **Task2:** Create Architecture Proof of Concept. ☑
- **Task3:** Define Architecture. ☑
- **Task4:** Define Reuse Options.

Presentation Title | Author | Date

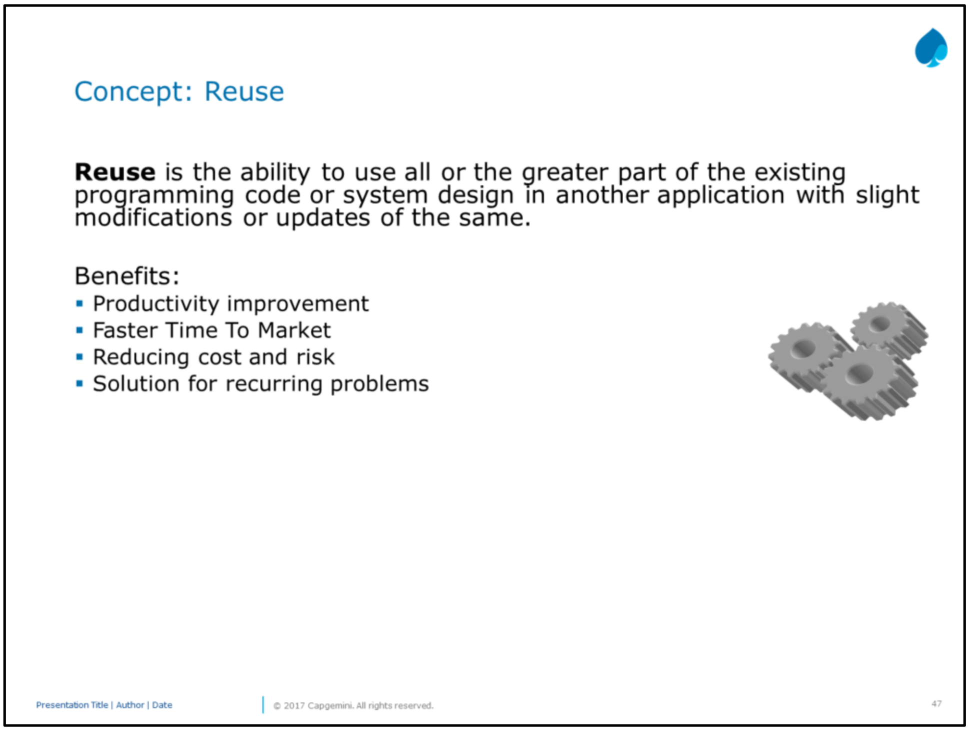
46

<u>Architecture: Key Tasks</u>:

Let us now look at the last task, that is “Define Reuse Options”.

Instructor Notes:

A survey in the organization some years ago indicated that thousands of “Login” components have been created! Login is one of the most generic features that is today implemented in almost all applications…imagine if reuse was practiced here.



Concept: Reuse

Reuse is the ability to use all or the greater part of the existing programming code or system design in another application with slight modifications or updates of the same.

Benefits:

- Productivity improvement
- Faster Time To Market
- Reducing cost and risk
- Solution for recurring problems

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

47

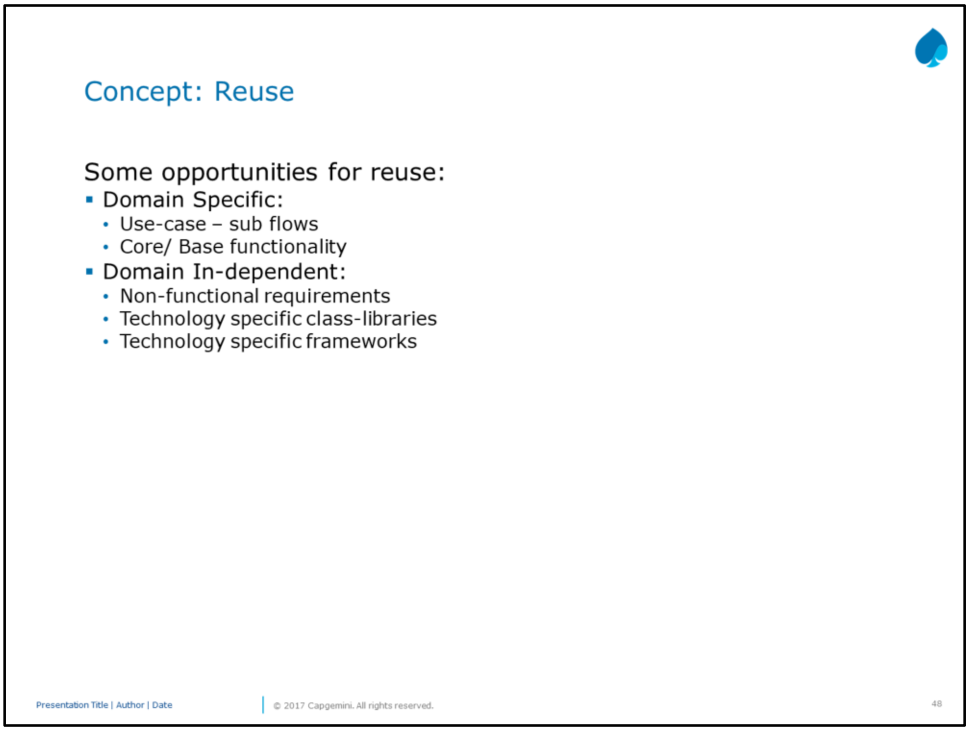
Task4: Define Reuse Options:

Concept: Reuse:

- An important characteristic of Object-Oriented technology is the “**reuse**” feature. Reuse can be at different levels, such as:
  - ➢ Code
  - ➢ Design
  - ➢ Documentation
- While we all understand the benefits of reuse, unfortunately it remains more talked about rather than actually implemented. Quite a lot of thought and efforts do go into this activity. However, the long terms benefits that are accrued are worth the efforts.

Instructor Notes:

Now scale up that imagination and think of all such generic functionalities that exist across projects/domains…and if reuse was practiced there!



Concept: Reuse

Some opportunities for reuse:

- Domain Specific:
  - Use-case – sub flows
  - Core/ Base functionality
- Domain In-dependent:
  - Non-functional requirements
  - Technology specific class-libraries
  - Technology specific frameworks

Presentation Title | Author | Date

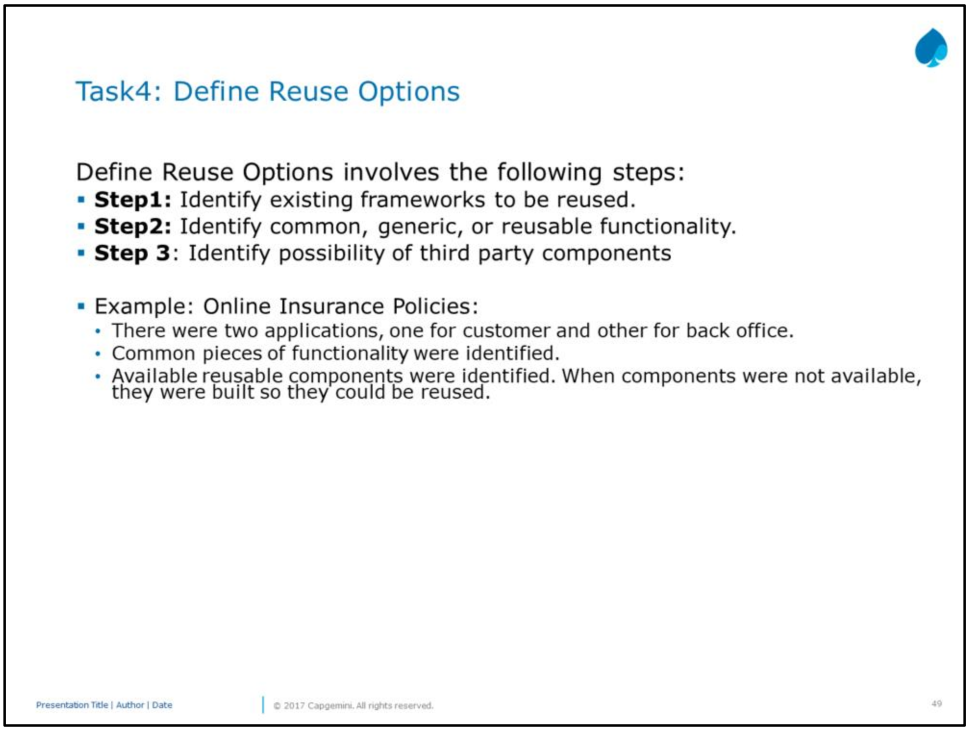
48

Task4: Define Reuse Options:
Concept: Reuse:

- Several reuse opportunities can be considered. These can be domain specific as well as domain independent. Some common and generic functionality are as follows:
  - ➢ Validation classes
  - ➢ Data converters
  - ➢ CRUD base classes , that is, those that have a class which gets extended for all create, retrieve, update, and delete classes
  - ➢ Database and Interface update interfaces, that is, a single interface which gets implemented for all writes happening outside your system

Instructor Notes:

An example from Insurance Project where reuse was effectively used.



Task4: Define Reuse Options

Define Reuse Options involves the following steps:

- **Step1:** Identify existing frameworks to be reused.
- **Step2:** Identify common, generic, or reusable functionality.
- **Step 3**: Identify possibility of third party components

- Example: Online Insurance Policies:
  - There were two applications, one for customer and other for back office.
  - Common pieces of functionality were identified.
  - Available reusable components were identified. When components were not available, they were built so they could be reused.

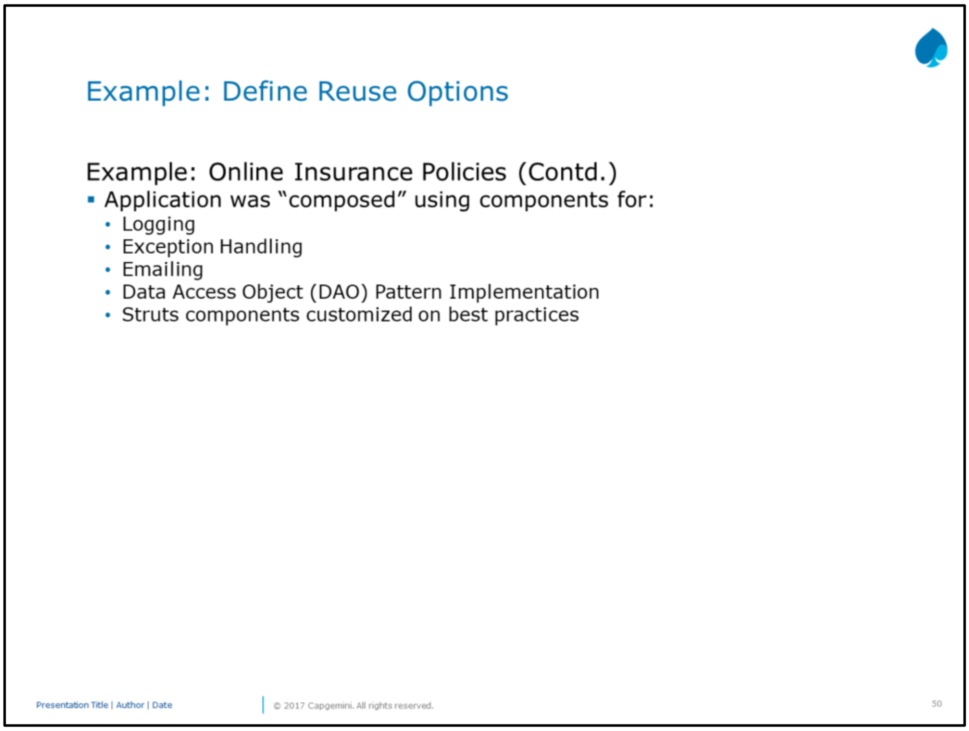
Presentation Title | Author | Date | © 2017 Capgemini. All rights reserved. 49

Task4: Define Reuse Options:

- Steps of "Define Reuse options" involve:
  - ➢ Identifying existing components or frameworks for reuse
  - ➢ Identifying functionality that can be reused within the application
  - ➢ Identifying possibility of third party components. These could either be purchased or downloaded. In case of usage of third party components, very important to comply to IPR.
- As you can see in the above example, there can be enough potential for reuse even within the same project.

Instructor Notes:

An example from Insurance Project where reuse was effectively used.



Example: Define Reuse Options

Example: Online Insurance Policies (Contd.)

- Application was “composed” using components for:
  - Logging
  - Exception Handling
  - Emailing
  - Data Access Object (DAO) Pattern Implementation
  - Struts components customized on best practices

Presentation Title | Author | Date

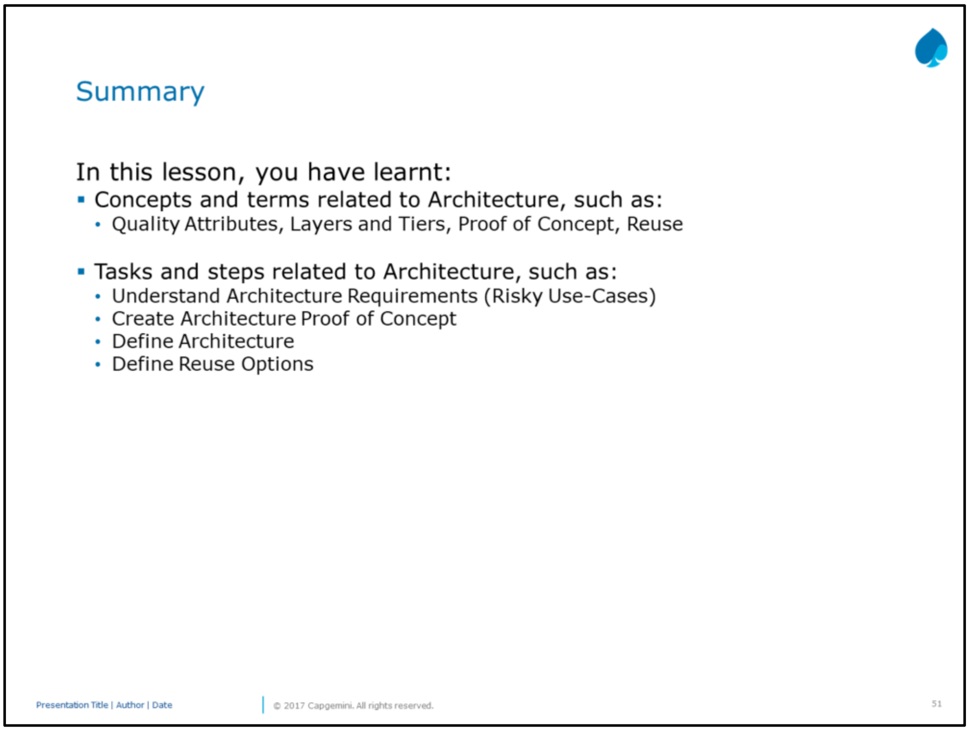
© 2017 Capgemini. All rights reserved.

50

Example: Define Reuse Options:

- In the case of the project in the above slide, a J2EE framework development project done couple of years ago was leveraged.
- The development team extracted the required functionalities from the J2EE framework, packaged them as standalone components, and made them available as standard JAR files for wide acceptance and reuse.

Instructor Notes:



Summary

In this lesson, you have learnt:

- Concepts and terms related to Architecture, such as:
  - Quality Attributes, Layers and Tiers, Proof of Concept, Reuse
- Tasks and steps related to Architecture, such as:
  - Understand Architecture Requirements (Risky Use-Cases)
  - Create Architecture Proof of Concept
  - Define Architecture
  - Define Reuse Options

Presentation Title | Author | Date

51

Instructor Notes:

Answers for Review Questions:

Answer 1: Performance, Security, all “alities” like usability, reliability, availability etc.

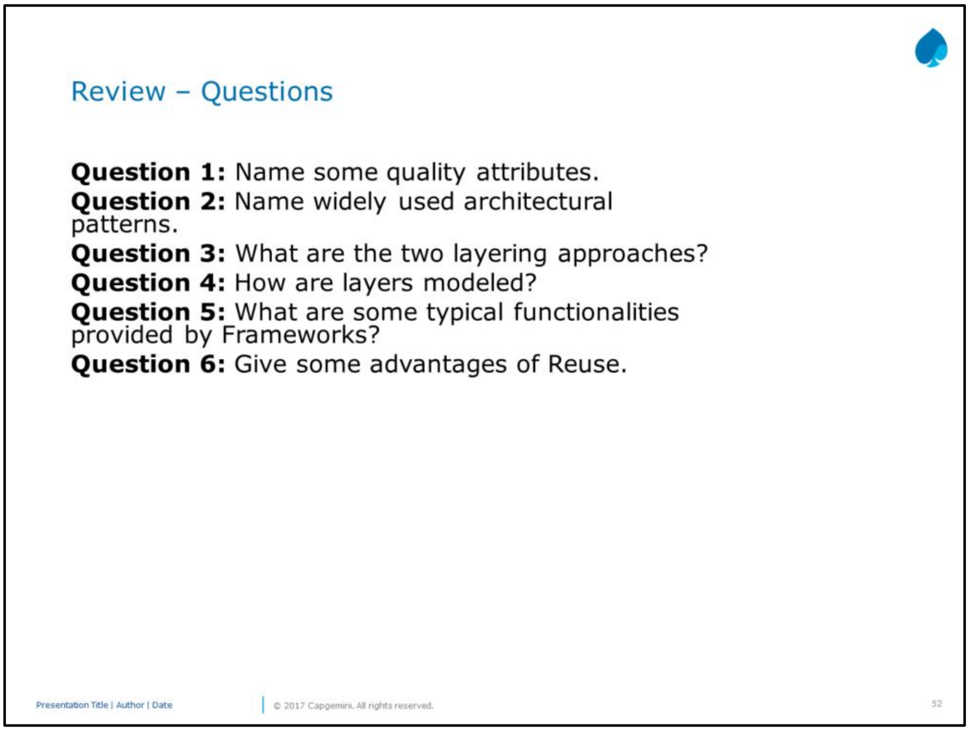
Answer 2: Layers and MVC

Answer 3: Responsibility and Reuse based

Answer 4: Using Stereotyped packages with dependency relationship

Answer 5: Validations, Exception Handling, Authentication and Authorization, Logging

Answer 6: Increased productivity, reduced costs, faster to market



Review – Questions

Question 1: Name some quality attributes.

Question 2: Name widely used architectural patterns.

Question 3: What are the two layering approaches?

Question 4: How are layers modeled?

Question 5: What are some typical functionalities provided by Frameworks?

Question 6: Give some advantages of Reuse.

Presentation Title | Author | Date

52