# A SUMMARY OF SOFTWARE ARCHITECTURE GUIDE

<@Triet Ho/>

1

## Introduction

- This presentation (58 slides) is a summary of Microsoft Application Architecture Guide 2nd Edition book (560 pages).
  - http://msdn.microsoft.com/en-us/library/ff650706.aspx
- Use these slides as a reference if you want to want to get overview of Software Architecture.
  - Common What and Why, common Application Types and Considerations when design a specific Application Type
- Notes: This presentation is designed for reading offline, not a seminar or a training.

2

## What Is Software Architecture

- Software application architecture is the process of defining a structured solution that meets all of the technical and operational requirements, while optimizing common quality attributes such as performance, security, and manageability.
- It involves a series of decisions based on a wide range of factors, and each of these decisions can have considerable impact on the quality, performance, maintainability, and overall success of the application.

3

## Why Is Architecture Important?

- Like any other complex structure, software must be built on a solid foundation. Failing to consider key scenarios, failing to design for common problems, or failing to appreciate the long term consequences of key decisions can put your application at risk.
- Modern tools and platforms help to simplify the task of building applications, but they do not replace the need to design your application carefully, based on your specific scenarios and requirements.
- The risks exposed by poor architecture include software that is unstable, is unable to support existing or future business requirements, or is difficult to deploy or manage in a production environment.

4

## The Goals of Architecture

- Application architecture seeks to build a bridge between business requirements and technical requirements by understanding use cases, and then finding ways to implement those use cases in the software.
- The goal of architecture is to identify the requirements that affect the structure of the application.
- Keep in mind that the architecture should:
  - Expose system's structure but hide the implementation details.
  - Realize all of the use cases and scenarios.
  - Try to address the requirements of various stakeholders.
  - Handle both functional and quality requirements.

5

## The Architectural Landscape

- It is important to understand the key forces that are shaping architectural decisions today, and which will change how architectural decisions are made in the future.
- Consider the following key trends:
  - **User empowerment.** A design that supports user empowerment is flexible, configurable, and focused on the user experience.
  - **Market maturity.** Take advantage of market maturity by taking advantage of existing platform and technology options.
  - **Flexible design.** Increasingly, flexible designs take advantage of loose coupling to allow reuse and to improve maintainability
  - **Future trends.** When building your architecture, understand the future trends that might affect your design after deployment.

6

## Key Architecture Principles

- Build to change instead of building to last.
- Model to analyze and reduce risk.
- Use models and visualizations as a communication and collaboration tool.
- Identify key engineering decisions.

7

## Key Design Principles

- **Separation of concerns.** Divide your application into distinct features with as little overlap in functionality as possible.
- **Single Responsibility principle.** Each component or module should be responsible for only a specific feature or functionality, or aggregation of cohesive functionality
- **Principle of Least Knowledge** A component or object should not know about internal details of other components or objects.
- **Don't repeat yourself (DRY).** You should only need to specify intent in one place. The functionality should not be duplicated in any other component.
- **Minimize upfront design.** Only design what is necessary. If your application requirements are unclear, or if there is a possibility of the design evolving over time, avoid making a large design effort prematurely. This principle is sometimes known as YAGNI ("You ain't gonna need it").

8

## Key Design Considerations

- Determine the Application Type
- Determine the Deployment Strategy
- Determine the Appropriate Technologies
- Determine the Quality Attributes
  - (security, performance, and usability)
- Determine the Crosscutting Concerns
  - (Instrumentation and logging, Authentication, Authorization, Exception management, Communication, Caching.)

9

## What Is an Architectural Style?

- An architectural style, sometimes called an architectural pattern, is a set of principles - a coarse grained pattern that provides an abstract framework for a family of systems.
- An architectural style improves partitioning and promotes design reuse by providing solutions to frequently recurring problems.
- You can think of architecture styles and patterns as sets of principles that shape an application

10

# Summary of Key Architectural Styles

| Architecture style | Description |
|---|---|
| *Client/Server* | Segregates the system into two applications, where the client makes requests to the server. In many cases, the server is a database with application logic represented as stored procedures. |
| *Component-Based Architecture* | Decomposes application design into reusable functional or logical components that expose well-defined communication interfaces. |
| *Domain Driven Design* | An object-oriented architectural style focused on modeling a business domain and defining business objects based on entities within the business domain. |
| *Layered Architecture* | Partitions the concerns of the application into stacked groups (layers). |
| *Message Bus* | An architecture style that prescribes use of a software system that can receive and send messages using one or more communication channels, so that applications can interact without needing to know specific details about each other. |
| *N-Tier / 3-Tier* | Segregates functionality into separate segments in much the same way as the layered style, but with each segment being a tier located on a physically separate computer. |
| *Object-Oriented* | A design paradigm based on division of responsibilities for an application or system into individual reusable and self-sufficient objects, each containing the data and the behavior relevant to the object. |
| *Service-Oriented Architecture (SOA)* | Refers to applications that expose and consume functionality as a service using contracts and messages. |

11

# Inputs, Outputs, and Design Steps

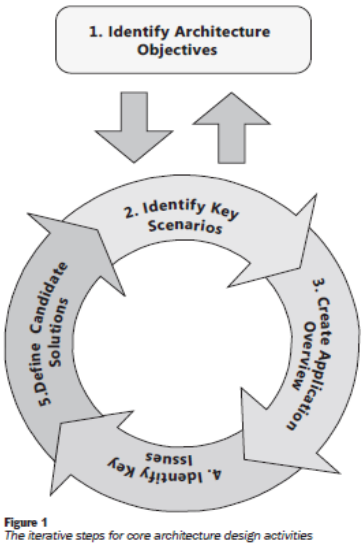



**Figure 1**
*The iterative steps for core architecture design activities*

12

# Inputs, Outputs, and Design Steps

- **Identify Architecture Objectives.** Clear objectives help you to focus on your architecture and on solving the right problems in your design.
- **Key Scenarios.** Use key scenarios to focus your design on what matters most, and to evaluate your candidate architectures when they are ready.
- **Application Overview.** Identify your application type, deployment architecture, architecture styles, and technologies in order to connect your design to the real world in which the application will operate.
- **Key Issues.** Identify key issues based on quality attributes and crosscutting concerns.
- **Candidate Solutions.** Create an architecture spike or prototype that evolves and improves the solution and evaluate it against your key scenarios, issues, and deployment constraints before beginning the next iteration of your architecture.

13

# Whiteboard Your Architecture

- If you cannot whiteboard the architecture then it suggests that it is not well understood

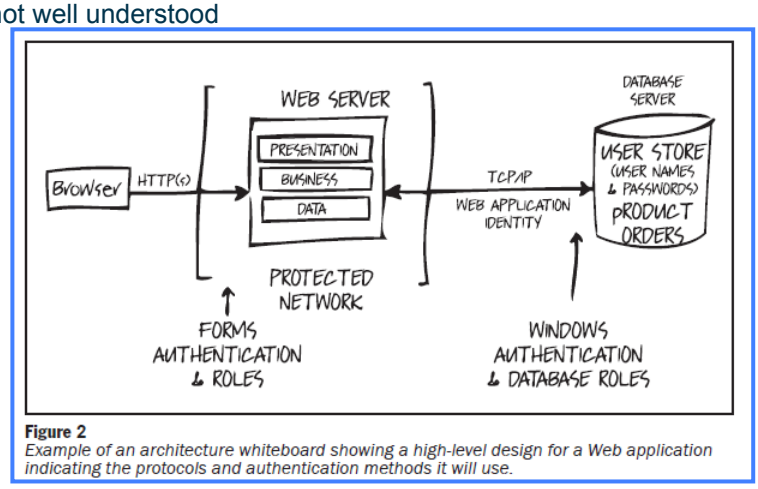



**Figure 2**
*Example of an architecture whiteboard showing a high-level design for a Web application indicating the protocols and authentication methods it will use.*

14

# Logical Layered Design

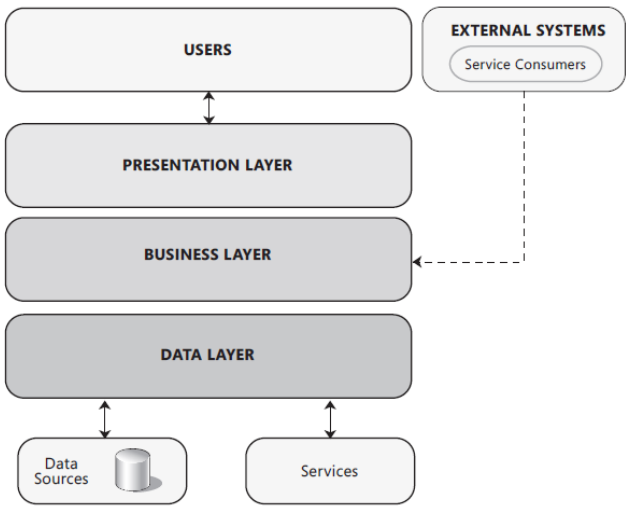



**Figure 1**
*The logical architecture view of a layered system*

15

# Logical Layered Design

- **Presentation layer.** This layer contains the user oriented functionality responsible for managing user interaction with the system, and generally consists of components that provide a common bridge into the core business logic encapsulated in the business layer.
- **Business layer.** This layer implements the core functionality of the system, and encapsulates the relevant business logic. It generally consists of components, some of which may expose service interfaces that other callers can use.
- **Data layer.** This layer provides access to data hosted within the boundaries of the system, and data exposed by other networked systems; perhaps accessed through services. The data layer exposes generic interfaces that the components in the business layer can consume.

16

# Services and Layers

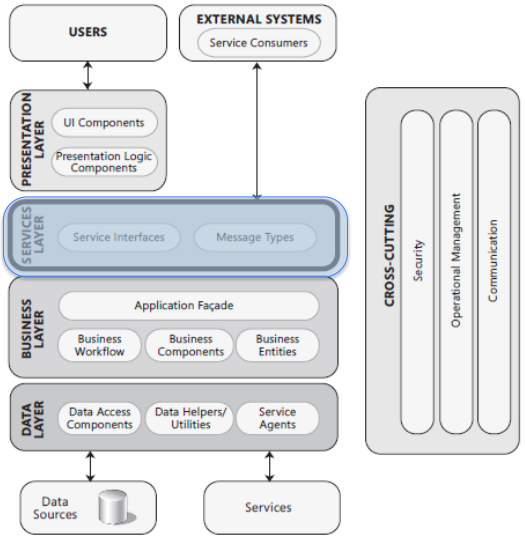



**Figure 2**
*Incorporating a services layer in an application*

17

# Design Steps for a Layered Structure

- Step 1 – Choose Your Layering Strategy
- Step 2 – Determine the Layers You Require
- Step 3 – Decide How to Distribute Layers and Components
- Step 4 – Determine If You Need to Collapse Layers
- Step 5 – Determine Rules for Interaction between Layers
- Step 6 – Identify Cross Cutting Concerns
- Step 7 – Define the Interfaces between Layers
- Step 8 – Choose Your Deployment Strategy
- Step 9 – Choose Communication Protocols

18

# Presentation Layer Guidelines

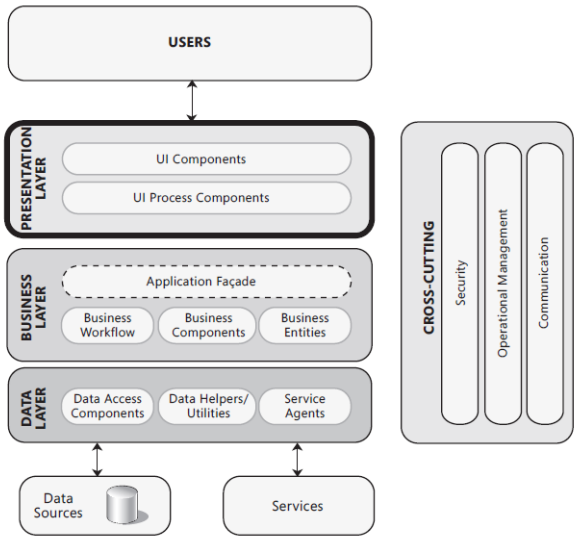



**Figure 1**
*A typical application showing the presentation layer and the components it may contain*

19

# Presentation Layer Guidelines

- General Design Considerations
  - Choose the appropriate application type
  - Choose the appropriate UI technology
  - Use the relevant patterns
  - Design for separation of concerns.
  - Consider human interface guidelines
  - Adhere to user driven design principles

20

## Presentation Layer Guidelines

- Specific Design Issues
  - Caching
  - Communication
  - Composition
  - Exception Management
  - Navigation
  - User Experience
  - User Interface
  - Validation

21

## Business Layer Guidelines

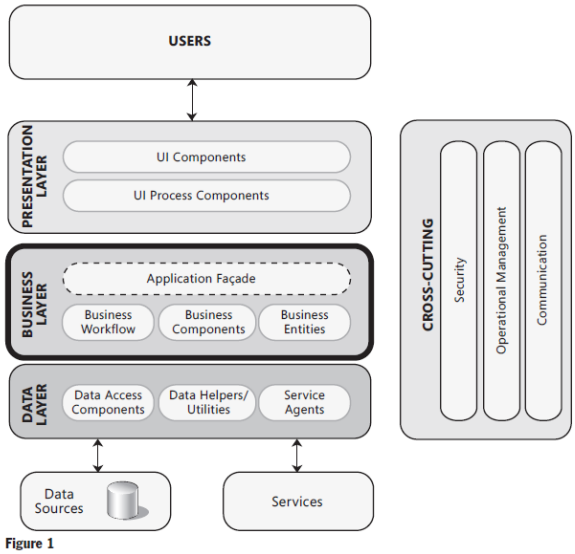



**Figure 1**
*A typical application showing the business layer and the components it may contain*

22

## Business Layer Guidelines

- **Application façade.** This optional component typically provides a simplified interface to the business logic components, often by combining multiple business operations into a single operation that makes it easier to use the business logic.
- **Business Logic components.** Business logic is defined as any application logic that is concerned with the retrieval, processing, transformation, and management of application data; application of business rules and policies; and ensuring data consistency and validity.
  - **Business Workflow components.** Business workflow components define and coordinate long running, multistep business processes, and can be implemented using business process management tools. (Behaviour)
  - **Business Entity components.** or—more generally—business objects, encapsulate the business logic and data necessary to represent real world elements. (Static Structure)

23

## Business Layer Guidelines

- General Design Considerations
  - Decide if you need a separate business layer.
  - Identify the responsibilities and consumers of your business layer.
  - Do not mix different types of components in your business layer.
  - Reduce round trips when accessing a remote business layer
  - Avoid tight coupling between layers

24

## Business Layer Guidelines

- Specific Design Issues
  - Authentication
  - Authorization
  - Caching
  - Coupling and Cohesion
  - Exception Management
  - Logging, Auditing, and Instrumentation
  - Validation

25

## Data Layer Guidelines

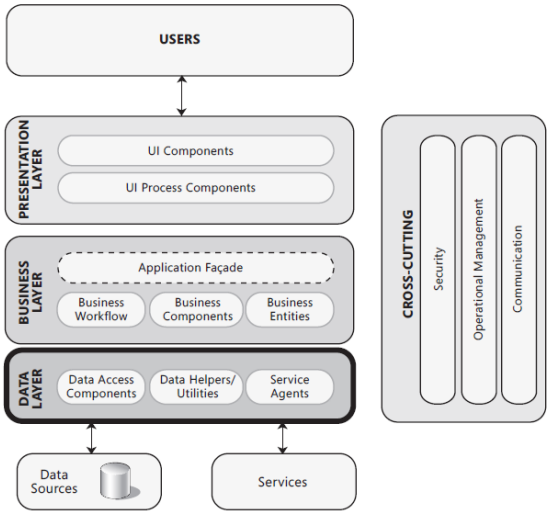



**Figure 1**
*A typical application showing the data layer and the components it may contain*

26

## Data Layer Guidelines

- **Data Access components**. These components abstract the logic required to access the underlying data stores. They centralize common data access functionality in order to make the application easier to configure and maintain.
- **Service agents**. When a business component must access data provided by an external service, you might need to implement code to manage the semantics of communicating with that particular service. Service agents implement data access components that isolate the varying requirements for calling services from your application, and may provide additional services such as caching, offline support, and basic mapping between the format of the data exposed by the service and the format your application requires.

27

## Data Layer Guidelines

- General Design Considerations
  - Choose an appropriate data access technology.
  - Use abstraction to implement a loosely coupled interface to the data access layer.
  - Encapsulate data access functionality within the data access layer.
  - Decide how to map application entities to data source structures
  - Consider consolidating data structures.
  - Decide how you will manage connections.
  - Determine how you will handle data exceptions
  - Consider security risks.
  - Reduce round trips.
  - Consider performance and scalability objectives.

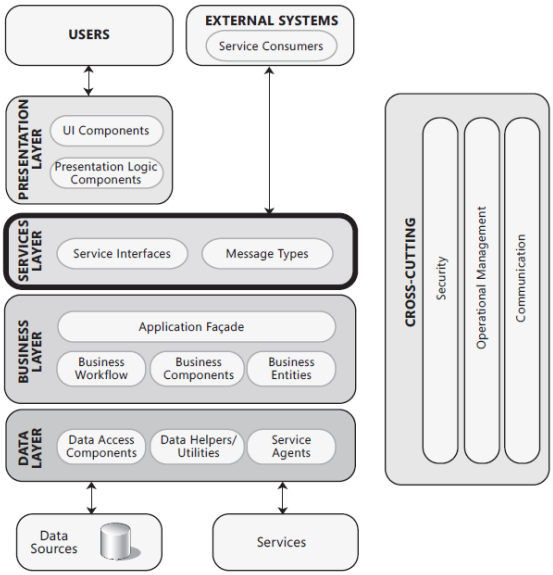
28

# Data Layer Guidelines

- Specific Design Issues
  - Batching
  - Binary Large Objects (BLOBs)
  - Connections
  - Data Format
  - Exception Management
  - Object Relational Mapping
  - Queries
  - Stored Procedures
  - Stored Procedures vs. Dynamic SQL
  - Transactions
  - Validation
  - XML

29

# Service Layer Guidelines




30

## Service Layer Guidelines

- **Service interfaces**. Services expose a service interface to which all inbound messages are sent. You can think of a service interface as a façade that exposes the business logic implemented in the application (typically, logic in the business layer) to potential consumers.
- **Message types.** When exchanging data across the service layer, data structures are wrapped by message structures that support different types of operations. The services layer will also usually include data types and contracts that define the data types used in messages.

31

## Service Layer Guidelines

- Design Considerations
  - Design services to be application-scoped and not component-scoped
  - Design services and data contracts for extensibility and without the assumption that you know who the client is.
  - Design only for the service contract
  - Separate service layer concerns from infrastructure concerns
  - Compose entities from standard elements
  - Design to assume the possibility of invalid requests.
  - Ensure that the service can detect and manage repeated messages. Ensure that the service can manage messages arriving out of order

32

## Service Layer Guidelines

- Specific Design Issues
  - Authentication
  - Authorization
  - Communication
  - Exception Management
  - Messaging Channels
  - Message Construction
  - Message Endpoint
  - Message Protection
  - I Message Routing
  - Message Transformation
  - Service Interface
  - Validation

33

34

## Application Archetypes Summary

- Mobile applications.
  - Applications of this type can be developed as thin client or rich client applications. Rich client mobile applications can support disconnected or occasionally connected scenarios. Web or thin client applications support connected scenarios only. Device resources may prove to be a constraint when designing mobile applications
- Rich client applications.
  - Applications of this type are usually developed as standalone applications with a graphical user interface that displays data using a range of controls. Rich client applications can be designed for disconnected and occasionally connected scenarios if they need to access remote data or functionality.
- Rich Internet applications
  - Applications of this type can be developed to support multiple platforms and multiple browsers, displaying rich media or graphical content. Rich Internet applications run in a browser sandbox that restricts access to some features of the client.

35

## Application Archetypes Summary

- Service applications
  - Services expose shared business functionality and allow clients to access them from a local or a remote system. Service operations are called using messages, based on XML schemas, passed over a transport channel. The goal of this type of application is to achieve loose coupling between the client and the server.
- Web applications.
  - Applications of this type typically support connected scenarios and can support different browsers running on a range of operating systems and platforms.

36

# Application Type Considerations

| Application type | Benefits | Considerations |
|---|---|---|
| *Mobile applications* | Support for handheld devices.<br>Availability and ease of use for out of office users.<br>Support for offline and occasionally-connected scenarios. | Input and navigation limitations.<br>Limited screen display area. |
| *Rich client applications* | Ability to leverage client resources.<br>Better responsiveness, rich UI functionality, and improved user experience.<br>Highly dynamic and responsive interaction.<br>Support for offline and occasionally connected scenarios. | Deployment complexity; however, a range of installation options such as ClickOnce, Windows Installer, and XCOPY are available.<br>Challenging to version over time.<br>Platform specific. |

37

# Application Type Considerations

| Application type | Benefits | Considerations |
|---|---|---|
| *Rich Internet applications (RIA)* | The same rich user interface capability as rich clients.<br>Support for rich and streaming media and graphical display.<br>Simple deployment with the same distribution capabilities (reach) as Web clients.<br>Simple upgrade and version updating.<br>Cross-platform and cross-browser support. | Larger application footprint on the client compared to a Web application.<br>Restrictions on leveraging client resources compared to a rich client application.<br>Requires deployment of a suitable runtime framework on the client. |
| *Service applications* | Loosely coupled interactions between client and server.<br>Can be consumed by different and unrelated applications.<br>Support for interoperability. | No UI support.<br>Dependent on network connectivity. |
| *Web applications* | Broad reach and a standards-based UI across multiple platforms.<br>Ease of deployment and change management. | Dependent on continual network connectivity.<br>Difficult to provide a rich user interface. |

38

# Web Application Archetype

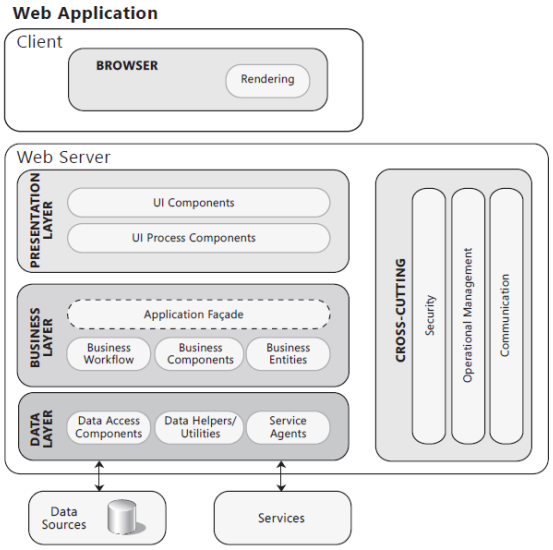



**Figure 5**
*The typical structure of a Web application*

39

# Designing Web Applications

- General Design Considerations
  - Partition your application logically.
  - Use abstraction to implement loose coupling between layers.
  - Understand how components will communicate with each other
  - Consider caching to minimize server round trips.
  - Consider logging and instrumentation
  - Consider authenticating users across trust boundaries
  - Do not pass sensitive data in plaintext across the network
  - Design your Web application to run using a least-privileged account

40

## Designing Web Applications

- Specific Design Issues
  - Application Request Processing
  - Authentication
  - Authorization
  - Caching
  - Exception Management
  - Logging and Instrumentation
  - Navigation
  - Page Layout
  - Page Rendering
  - Session Management
  - Validation

41

## Designing Web Applications

- Other Considerations:
  - Design Considerations for Layers
    - Presentation Layer, Business Layer, Data Layer, Service Layer
  - Testing and Testability Considerations
  - Technology Considerations
  - Deployment Considerations
    - Non-distributed Deployment, Distributed Deployment, Load Balancing

42

# Rich Client Application Archetype

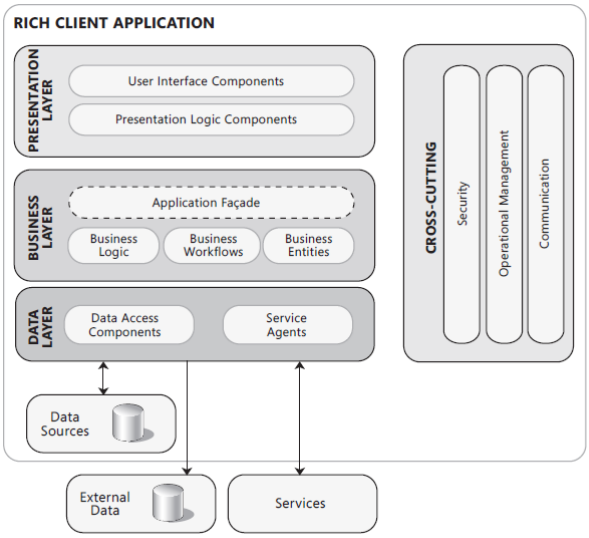



**Figure 2**
*The typical structure of a rich client application*



# Rich Client Application Archetype

- General Design Considerations
  - Choose an appropriate technology based on application requirements.
  - Separate presentation logic from interface implementation.
  - Identify the presentation tasks and presentation flows.
  - Design to provide a suitable and usable interface.
  - Apply separation of concerns across all layers.
  - Reuse common presentation logic.
  - Loosely couple your client from any remote services it uses
  - Avoid tight coupling to objects in other layers
  - Reduce round trips when accessing remote layers

## Rich Client Application Archetype

- Specific Design Issues
  - Business Layer
  - Communication
  - Composition
  - Configuration Management
  - Data Access
  - Exception Management
  - Maintainability
  - Presentation Layer
  - State Management
  - Workflow

45

## Rich Client Application Archetype

- Other Considerations
  - Security Considerations
    - protecting sensitive data, user authentication and authorization, guarding against attack from malicious code and users, and auditing and logging events and user activity.
  - Data Handling Considerations
    - Caching Data, Data Concurrency, Data Binding
  - Offline/Occasionally Connected Considerations
  - Technology Considerations
  - Deployment Considerations
    - Stand-alone Deployment, Client/Server Deployment, N-Tier Deployment, Deployment Technologies

46

# Rich Internet Application Archetype

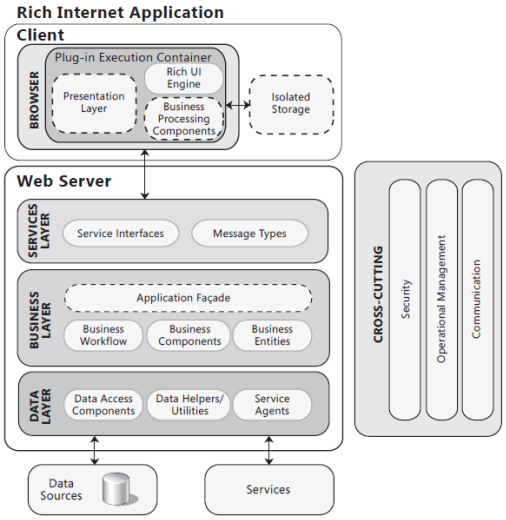



**Figure 3**
*The typical structure of a rich Internet application*

47

# Designing Rich Internet Applications

- General Design Considerations
  - Choose a RIA based on audience, rich interface, and ease of deployment.
  - Design to use a Web infrastructure utilizing services
  - Design to take advantage of client processing power
  - Design for execution within the browser sandbox
  - Determine the complexity of your UI requirements
  - Use scenarios to increase application performance or responsiveness.
  - Design for scenarios where the plug-in is not installed

48

## Designing Rich Internet Applications

- Specific Design Issues
  - Business Layer
  - Caching
  - Communication
  - Composition
  - Data Access
  - Exception Management
  - Logging
  - Media and Graphics
  - Mobile
  - Portability
  - Presentation
  - State Management
  - Validation

49

## Designing Rich Internet Applications

- Other Considerations
  - Security Considerations
  - Data Handling Considerations
    - Read-only reference data, Transient data.
  - Technology Considerations
    - Versions and Target Platforms, Security
  - Deployment Considerations
    - Installation of the RIA Plug-In, Distributed Deployment, Load Balancing, Web Farm Considerations

50

# Mobile Application Archetype

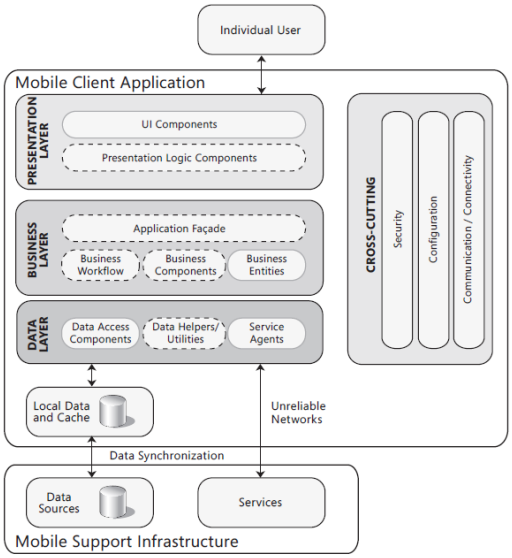



**Figure 1**
*The typical structure of a mobile application*

51

# Designing Mobile Applications

- General Design Considerations
  - Decide if you will build a rich client, a thin Web client, or rich Internet application (RIA).
  - Determine the device types you will support.
  - Consider occasionally connected and limited-bandwidth scenarios when appropriate.
  - Design a UI appropriate for mobile devices, taking into account platform constraints.
  - Design a layered architecture appropriate for mobile devices that improves reuse and maintainability.
  - Consider device resource constraints such as battery life, memory size, and processor speed.

52

## Designing Mobile Applications

- Specific Design Issues
  - Authentication and Authorization
  - Caching
  - Communication
  - Configuration Management
  - Data Access
  - Device Specifics
  - Exception Management
  - Logging
  - Porting Applications
  - Power Management
  - Synchronization
  - Testing
  - User Interface
  - Validation

53

## Designing Mobile Applications

- Other considerations
  - Technology Considerations
  - Deployment Considerations

54

# Service Archetype

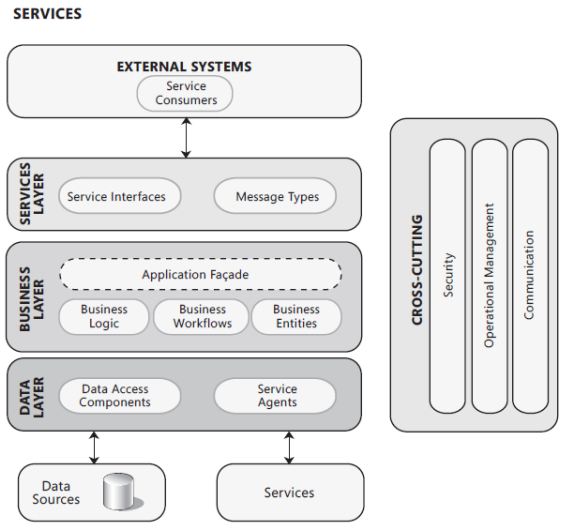



**Figure 4**
*The typical structure of a service application*

55

# Designing Service Applications

- General Design Considerations
  - Consider using a layered approach to designing service applications and avoid tight coupling across layers.
  - Design coarse-grained operations
  - Design data contracts for extensibility and reuse.
  - Design only for the service contract
  - Design services to be autonomous
  - Design to assume the possibility of invalid requests
  - Design services based on policy and with explicit boundaries
  - Separate service concerns from infrastructure operational concerns
  - Use separate assemblies for major components in the service layer
  - Avoid using data services to expose individual tables in a database
  - Design workflow services to use interfaces supported by your workflow engine.

56

## Designing Service Applications

- Specific Design Issues
  - Authentication
  - Authorization
  - Business Layer
  - Communication
  - Data Layer
  - Exception Management
  - Message Construction
  - Message Endpoint
  - Message Protection
  - Message Transformation
  - Message Exchange Patterns
  - Representational State Transfer
  - Service Layer
  - SOAP
  - Validation

57

- Other considerations
  - Technology Considerations
  - Deployment Considerations

58

trietho@gmail.com

# THANK YOU! - 2011

59