

PRESENTATION LAYER



OUTLINE

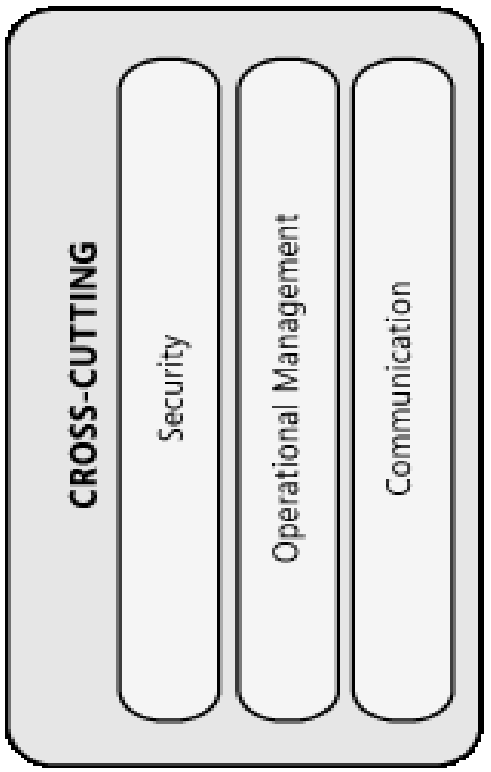
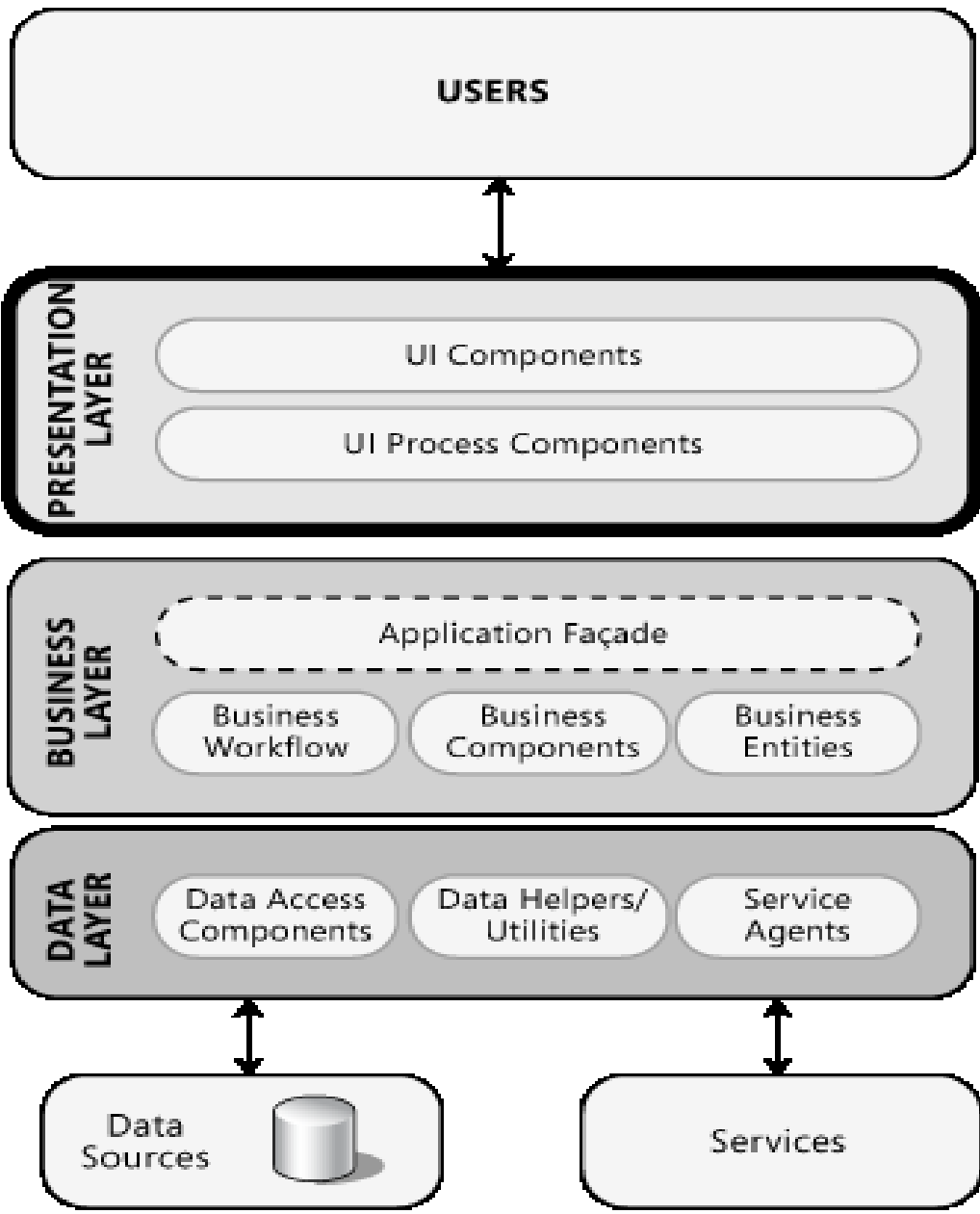
1. Presentation Layer
2. Typical Components in the Presentation Layer
3. General design Considerations
4. Specific design issues
5. Technology Considerations
6. Performance Considerations
7. Design Steps for the presentation layer
8. Relevant Design Patterns




PRESENTATION LAYER

- The presentation layer contains the components that implement and display the user interface and manage user interaction.
- This layer includes controls for user input and display, in addition to components that organize user interaction.





TYPICAL COMPONENTS IN THE PRESENTATION LAYER

1. **User Interface components:** These are the application's visual elements used to display information to the user and accept user input.
 1. **Presentation Logic components:** Presentation logic is the application code that defines the logical behavior and structure of the application in a way that is independent of any specific user interface implementation. The presentation logic components may include Presenter, Presentation Model, and ViewModel components.
- 

GENERAL DESIGN CONSIDERATIONS

1. Choose the appropriate application type
2. Choose the appropriate UI technology
3. Use the relevant patterns
4. Design for separation of concerns
5. Consider human interface guidelines
6. Adhere to user driven design principles



SPECIFIC DESIGN ISSUES

There are several common issues that you must consider as you develop your design. These issues can be categorized into specific areas of the design.

- Caching
- Communication
- Composition
- Exception Management
- Navigation
- User Experience
- User Interface
- Validation



CACHING

- Caching improves application performance and UI responsiveness.
- You can use data caching in the presentation layer to optimize data lookups and avoid network round trips, and to store the results of expensive or repetitive processes to avoid unnecessary duplicated processing.
- Guidelines for designing the caching strategy:
 - Choose the appropriate location for your cache, such as in memory or on disk.

Example: If your application is deployed in **Web farm**, **avoid using local caches** that must be synchronized. In general, for **Web and application farm deployments**, consider using a transactional resource manager such as **Microsoft SQL Server**, or a product that supports distributed caching such as the Danga Interactive "Memcached" technology or the Microsoft "Velocity" caching mechanism.



- Consider caching data in a ready to use format when working with an in-memory cache.

For example, use a specific object instead of caching raw database data. However, avoid caching volatile data as the cost of caching may exceed that of recreating or fetching the data again if it constantly changes.

- Do not cache sensitive data unless you encrypt it.
- Do not depend on data still being in your cache; it may have been removed. Also, consider that the cached data may be stale.
- Consider authorization rights for cached data. Only cache data for which you can apply appropriate authorization if users in different roles may access the data.
- If you are using multiple threads, ensure that all access to the cache is thread-safe.



COMMUNICATION

- Handle long-running requests with user responsiveness in mind, as well as code maintainability and testability.
- Guidelines for designing request processing:
 - Consider using asynchronous operations or worker threads to avoid blocking the UI for long-running actions in Windows Forms and WPF applications. In ASP.NET, consider using AJAX to perform asynchronous requests. Provide feedback to the user on the progress of the long running action.
 - Avoid mixing your UI processing and rendering logic.
 - When making expensive calls to remote sources or layers, such as when calling Web services or querying a database, consider if it makes more sense to make these calls *chatty* (many smaller requests) or *chunky* (one large request).



COMPOSITION

- Composition patterns help you to implement sharing, reuse, and replacement of presentation logic and views.
- Guidelines for designing your UI composition strategy:
 - Avoid dependencies between components.
 - Consider creating templates with placeholders.
 - Consider composing views from reusable modular parts.
 - Be cautious when using layouts generated dynamically at run time, which can be difficult to load and maintain. Investigate patterns and third-party libraries that support dynamic layout and injection of views and presentation at runtime.
 - When communicating between presentation components, consider using loosely coupled communication patterns such as Publish/Subscribe. This will lower the coupling between the components and improve testability and flexibility.




EXCEPTION MANAGEMENT

- Design a centralized exception management mechanism for your application that catches and manages unexpected exceptions in a consistent way.
- Guidelines for designing your exception management strategy:
 - Provide user friendly error messages to notify users of errors in the application, but ensure that you avoid exposing sensitive data in error pages, error messages, log files, and audit files.
 - Ensure that you catch exceptions that will not be caught elsewhere (such as in a global error handler), and clean up resources and state after an exception occurs.
 - Differentiate between system exceptions and business errors. In the case of business errors, display a user friendly error message and allow the user to retry the operation. In the case of system exceptions, check to see if an issue such as a service or database failure caused the exception, display a user friendly error message, and log the error message to assist in troubleshooting.
 - Only catch exceptions that you can handle, and avoid the use of custom exceptions when not necessary. Do not use exceptions to control application logic flow.



NAVIGATION

- Design your navigation strategy so that users can navigate easily through your screens or pages.
 - Ensure that you display navigation links and controls in a consistent way throughout your application to reduce user confusion and to hide application complexity.
 - Guidelines for designing your navigation strategy:
 - Design toolbars and menus to help users find functionality provided by the UI.
 - Consider using wizards to implement navigation between forms in a predictable way, and determine how you will preserve navigation state between sessions if this is necessary.
 - Avoid duplication of logic for navigation event handlers, and avoid hard-coding navigation paths where possible. Consider using the Command pattern to handle common actions from multiple sources.
- 

USER EXPERIENCE

- Consider the following guidelines when designing for user experience:
 - Do not design overloaded or over complex interfaces. Provide a clear path through the application for each key user scenario, and consider using colors and noninvasive animations to draw the user's attention to important changes in the UI, such as state changes.
 - Provide helpful and informative error messages, without exposing sensitive data.
 - For actions that might take longer to complete, try to avoid blocking the user. At a minimum, provide feedback on the progress of the action, and consider if the user should be able to cancel the process.
 - Consider empowering the user by providing flexibility and customization of the UI through configuration and, where appropriate, personalization.



USER INTERFACE

- Guidelines to be considered when designing your user interface:
 - Consider using a Separated Presentation pattern such as MVP to separate the layout design from interface processing. Use templates to provide a common look and feel to all of the UI screens, and maintain a common look and feel for all elements of your UI to maximize accessibility and ease of use. Avoid over complex layouts.
 - Consider using forms-based input controls for data collection tasks, a document-based input mechanism for collecting more free form input such as text or drawing documents, or a wizard-based approach for more sequenced or workflow driven data collection tasks.
 - Avoid using hard-coded strings, and using external resources for text and layout information, especially if your application will be localized
 - Consider accessibility in your design. You should consider users with disabilities when designing your input strategy.
 - Take into account different screen sizes and resolutions, and different device and input types such as mobile devices, touch screens, and pen and ink—enabled devices.



VALIDATION

- Designing an effective input and data validation strategy is critical for the security and correct operation of your application.
- Consider the following guidelines when designing your input and data validation strategy:
 - Input validation should be handled by the presentation layer, whilst business rule validation should be handled by the business layer.
 - Design your validation strategy to constrain, reject, and sanitize malicious input. Investigate design patterns and third party libraries that can assist in implementing validation. Identify business rules that are appropriate for validation, such as transaction limits, and implement comprehensive validation to ensure that these rules are not compromised.
 - Ensure that you correctly handle validation errors, and avoid exposing sensitive information in error messages.




TECHNOLOGY CONSIDERATIONS

Mobile Applications:

- Consider the following guidelines when designing a mobile application:
 - If you want to build full-featured connected, occasionally connected, or disconnected executable applications that run on a wide range of Microsoft Windows—based devices, consider using the Microsoft Windows Compact Framework.
 - If you want to build connected applications that support a wide variety of mobile devices, or require Wireless Application Protocol (WAP), compact HTML (cHTML), or similar rendering formats, consider using ASP.NET for Mobile.



Rich Client Applications:

- Consider the following guidelines when designing a rich client application:
 - If you want to build rich media and graphics capable applications, consider using Windows Presentation Foundation (WPF).
 - If you want to build applications that are downloaded from a Web server and execute on a Windows client, consider using XAML Browser Applications (XBAP).
 - If you want to build applications that are predominantly document-based, or are used for reporting, consider designing a Microsoft Office Business Application (OBA).
 - If you want to take advantage of the extensive range of third party controls, and rapid application development tools, consider using Windows Forms. If you decide to use Windows Forms and you are designing a composite application, consider using the patterns & practices Smart Client Software Factory.
- 

Rich Internet Applications:

- Consider the following guidelines when designing a Rich Internet Application (RIA):
 - If you want to build browser-based, connected applications that have broad cross-platform reach, are highly graphical, and support rich media and presentation features, consider using Silverlight.
 - If you decide to build an application using Silverlight, consider the following:
 - Consider using the Presentation Model (Model-View-ViewModel) pattern.
 - If you are designing an application that must last and change, consider using the patterns & practices Composite Client Application Guidance.



Web Applications:

- Consider the following guidelines when designing a Web application:
 - If you want to build applications that are accessed through a Web browser or specialist user agent, consider using ASP.NET.
 - If you decide to build an application using ASP.NET, consider the following:
 - Consider using master pages to simplify development and implement a consistent UI across all pages.
 - For increased interactivity and background processing, with fewer page reloads, consider using AJAX with ASP.NET Web Forms.
 - If you want to include islands of rich media content and interactivity, consider using Silverlight controls with ASP.NET.
 - If you want to improve the testability of your application, or implement a more clear separation between your application user interface and business logic, consider using the ASP.NET MVC Framework. This framework supports a model-view-controller based approach to Web application development.



PERFORMANCE CONSIDERATIONS

- Consider the following guidelines to maximize the performance of your presentation layer:
 - Design your presentation layer carefully so that it contains the functionality required to deliver a rich and responsive user experience.
 - Interaction between the presentation layer and the business or services layer of the application should be asynchronous. This avoids the possibility of high latency or intermittent connectivity adversely affecting the usability and responsiveness of the application.
 - Consider caching data in the presentation layer that will be displayed to the user.
 - In general, avoid maintaining session data or caching per-user data unless the number of users is limited, or the total size of the data relatively small.
 - Always use data paging when querying for information. Do not rely on queries that may return an unbounded volume of data, and use a data page size that is appropriate for the amount of data you will display.
 - In ASP.NET, use view state cautiously because it increases the volume of data included in each round trip, and can reduce the performance of the application.



DESIGN STEPS FOR THE PRESENTATION LAYER

1. Identify your client type
2. Choose your presentation layer technology
3. Design your user interface
4. Determine your data validation strategy
5. Determine your business logic strategy
 - UI Validation
 - Business Process Components
 - Domain Model
 - Rules Engine
6. Determine your strategy for communication with other layers
 - Direct Method Calls
 - Web services



RELEVANT DESIGN PATTERNS



Category	Relevant patterns
Caching	Cache Dependency Page Cache
Composition and Layout	Composition View Presentation Model Template View Transform View Two-step View
Exception Management	Exception Shielding
Navigation	Application Controller Front Controller Page Controller Command
User Experience	Asynchronous Callback Chain of Responsibility



THANK YOU !!!

