National College of Ireland

Mobile Architecture & Security

Presentation Layer

Mikhail.Timofeev@ncirl.ie

Overview

① Quick Recap

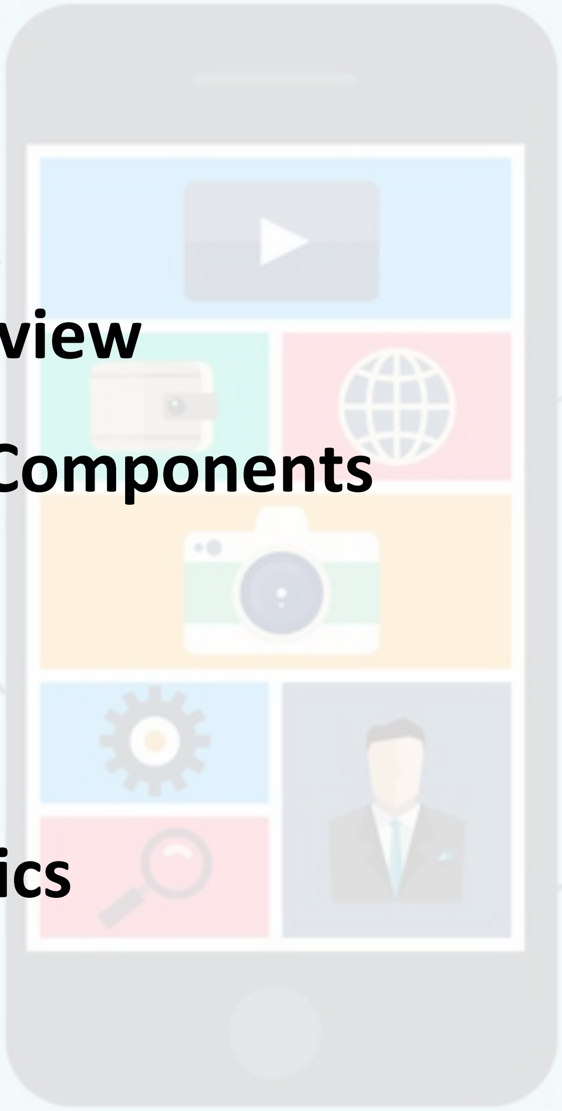
② Presentation Layer Overview

③ Designing Presentation Components

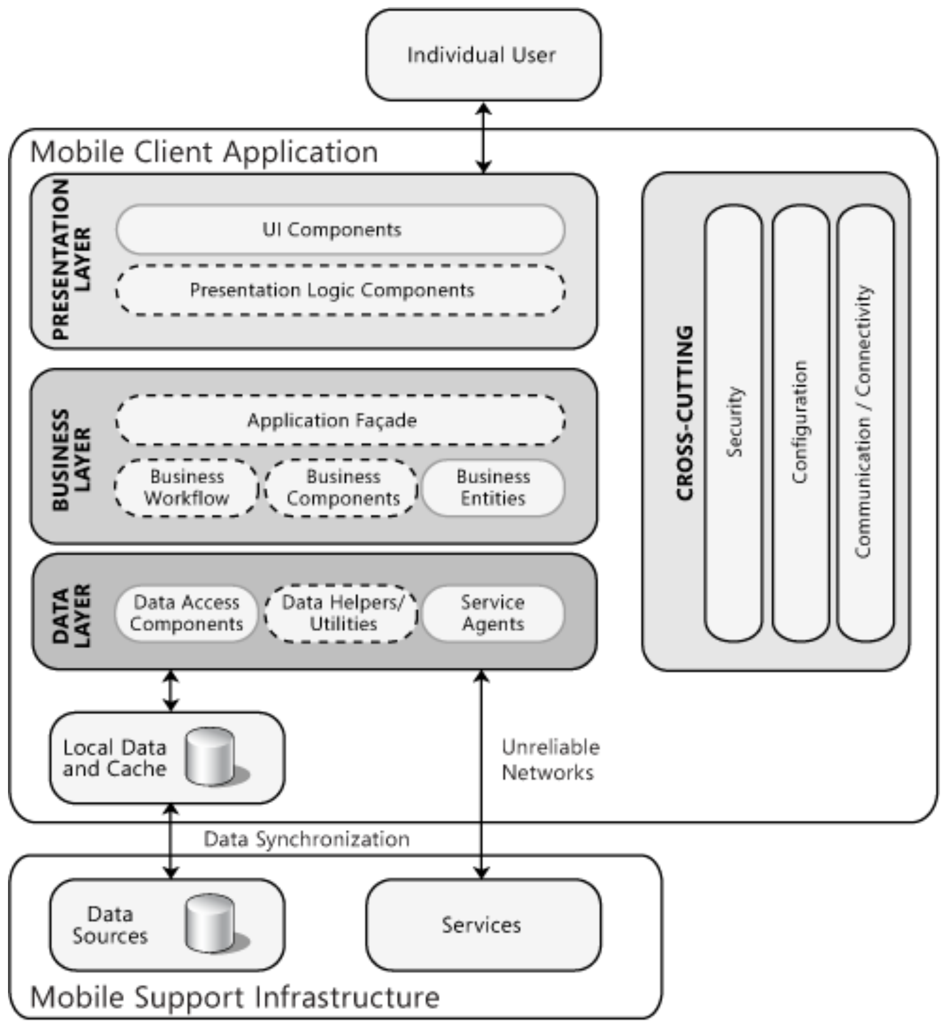
④ Design Considerations

⑤ MVC and MVP

⑥ Application Security Basics



Quick Recap

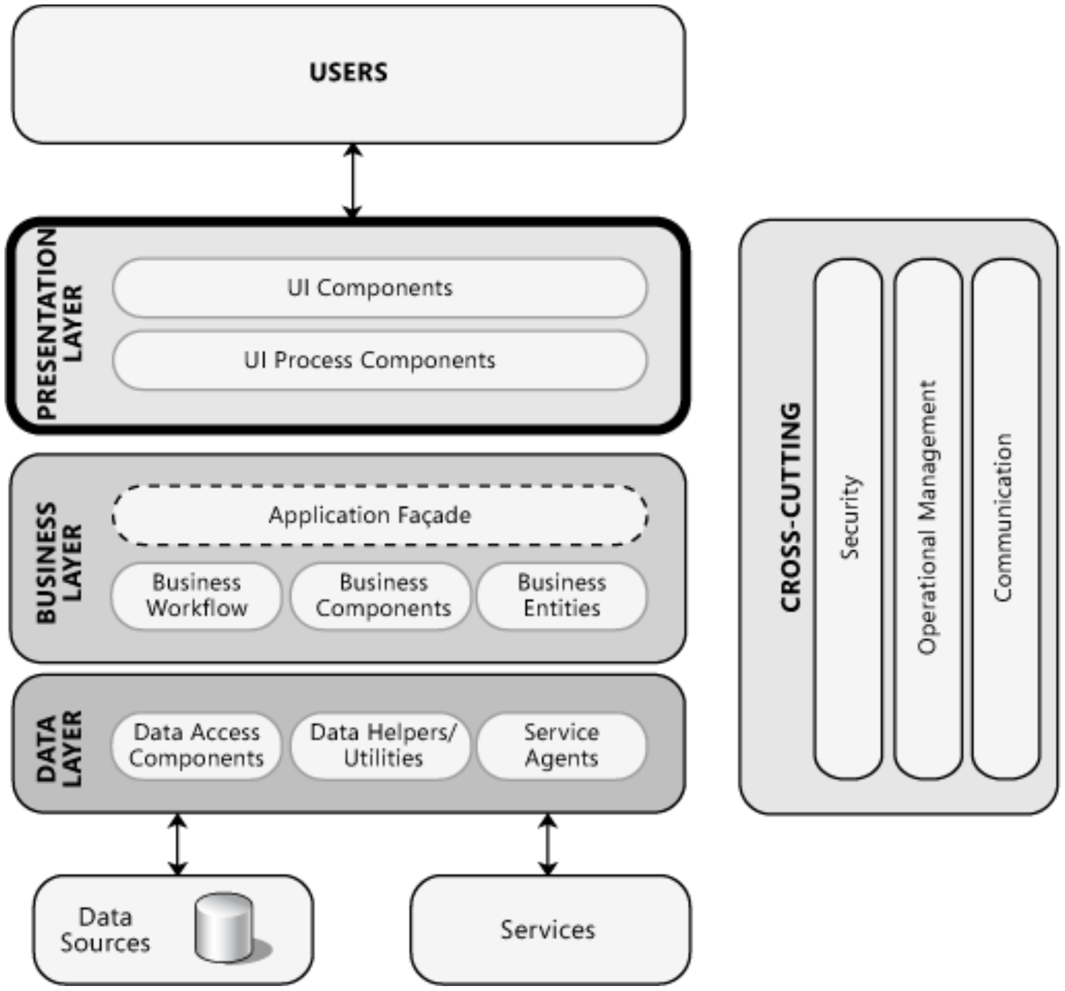


N-tier Architecture

- Presentation Layer
  - Choice of UI
  - Model-View-Presenter, Model-View-Controller
- Business Logic Layer
  - Entities, Workflows, Rules
- Data Access Layer
  - Storage, Integration
- Service Layer
  - Interfaces, patterns
  - REST, SOAP

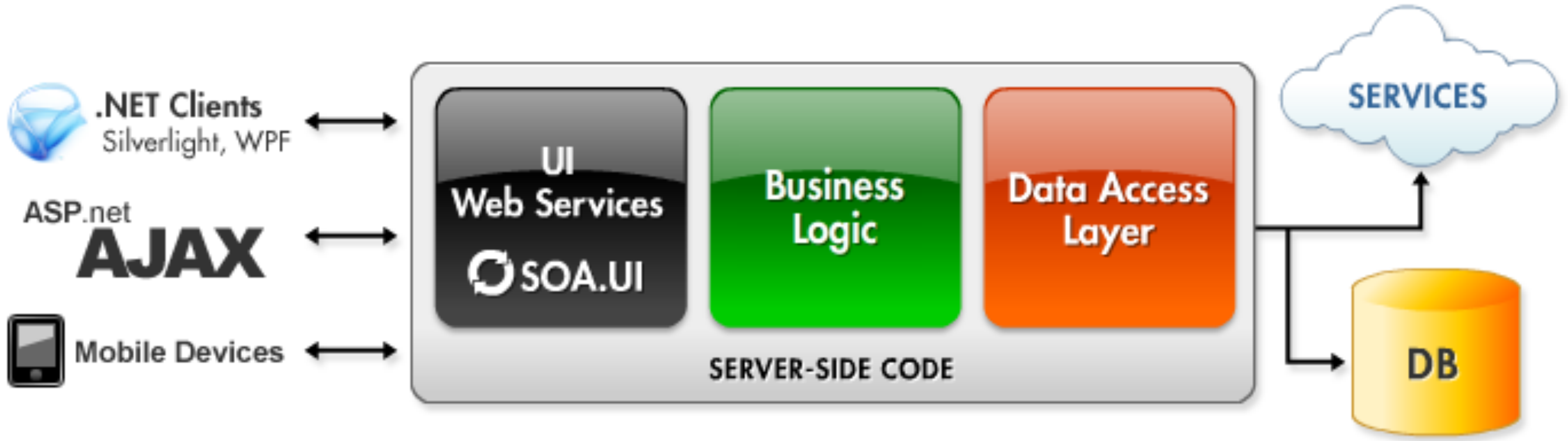
Presentation Layer Overview

- 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



Presentation Level Overview

- Important to know:
  - How the presentation layer fits into the typical layered application architecture
  - The components it usually contains
  - The key issues developers face when designing the presentation layer



Presentation Level Overview

- The presentation layer will usually include the following:
  - **User Interface components**: These are the application's visual elements used to display information to the user and accept user input
  - **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
  - When implementing the Separated Presentation pattern, the presentation logic components may include Presenter, Presentation Model, and ViewModel components
  - The presentation layer may also include Presentation Layer Model components that encapsulate the data from your business layer, or Presentation Entity components that encapsulate business logic and data in a form that is easily consumable by the presentation layer

Designing Presentation Components

Step 1 – Understand the UI Requirements

- Start by identifying the users of application, and understanding the goals and tasks these users wish to accomplish when using the application
- Sequencing of tasks is very important: step-by-step or adhoc
- Determine the information required by the user and the format in which it is expected
- Consider the current levels of user experience, and compare this to the user experience required for your UI to ensure that it is logical and intuitive
- Identify if the UI must expose rich functionality or user interaction, must be highly responsive, or requires graphical or animation support
- Consider the data types, formats and presentation formatting requirements for data such as dates, times, currency, etc.
- Choose appropriate controls, UI technologies and physical requirements

Designing Presentation Components

Step 2 – Determine the UI Type Required

- **Rich client applications**:
  - Stand-alone or networked applications with a graphical user interface that display data using a range of controls, and are deployed to a desktop or laptop computer for use by a local user
  - They are suitable for disconnected and occasionally connected scenarios because the application runs on the client
  - A rich client UI is a good choice when the UI must support rich functionality and rich user interaction or provide a highly dynamic and responsive user experience; or when the application must work in both connected and disconnected scenarios, take advantage of local system resources on the client machine, or integrate with other applications on that machine.

Designing Presentation Components

Step 2 – Determine the UI Type Required

- **Rich Internet applications (RIAs):**
  - Web applications with a rich graphical user interface that run inside a browser
  - RIAs are typically used for connected scenarios
  - A RIA is a good choice when your UI must support a dynamic and responsive user experience or use streaming media, and be widely accessible on a range of devices and platforms

Designing Presentation Components

Step 2 – Determine the UI Type Required

- **Web applications**:
  - Support connected scenarios and can support many different browsers running on a range of operating systems and platforms
  - A Web UI is a good choice when your UI must be standards-based, accessible on the widest range of devices and platforms, and work only in a connected scenario
  - Web applications are also well suited to applications whose content is to searchable by Web search engines

Designing Presentation Components

Step 3 – Choose the UI Technology

- **Rich client (Mobile)**
  - Microsoft .NET Compact Framework
  - ASP.NET for Mobile
  - Multiple frameworks for iOS & Android
- **Rich Internet Application (RIA)**
  - Silverlight
  - AngularJS
  - PhoneGap
  - Google Web Toolkit
  - Unity

  http://en.wikipedia.org/wiki/List_of_rich_Internet_application_frameworks
- **Web Application**
  - All traditional frameworks and tools

Designing Presentation Components

Step 4 – Design the Presentation Components

- **User Interface Components**
- **Presentation Logic Components**
- **Presentation Model Components**
  - These components support a separation of concerns in the presentation layer, and are often used to implement a separated presentation pattern such as MVP (Model-View-Presenter) or MVC (Model-View-Controller) by dividing UI processing into three distinct roles: Model, View, and Controller/Presenter
  - Separating the concerns in the presentation layer in this way increases maintainability, testability, and opportunities for reuse
  - The use of abstraction patterns such as dependency injection also makes it easier to test your presentation logic.

Designing Presentation Components

- **User Interface Components**
  - UI components are the visual elements that display information to the user and accept user input
  - Consider dividing your pages or windows into discrete user controls in order to minimize complexity and to allow reuse of these user controls.
  - Try to avoid inheritance hierarchies of user controls and pages to enable code reuse
  - Favor composition over inheritance and consider creating reusable presentation logic components instead
  - Try to avoid creating custom controls unless it is necessary for specialised display or data collection
  - If you find that your UI requirements cannot be achieved with the standard controls, consider buying a control toolkit before deciding to write your own custom controls

Designing Presentation Components

- **Presentation Logic Components**
  - Presentation logic components handle the non-visualization aspects of the user interface
  - If the UI requires complex processing or must communicate with other layers, consider using presentation logic components to decouple this processing from the UI components
  - Use presentation logic components to store state related to (but not specific to) the UI
  - Avoid implementing business logic and business rules, other than input and data validation, within the presentation logic components
  - Also, avoid implementing rendering or UI specific logic in the presentation logic components
  - Use presentation logic components to help your application recover from a failure or error by using them to make sure after recovery that the user interface is in a consistent state

Designing Presentation Components

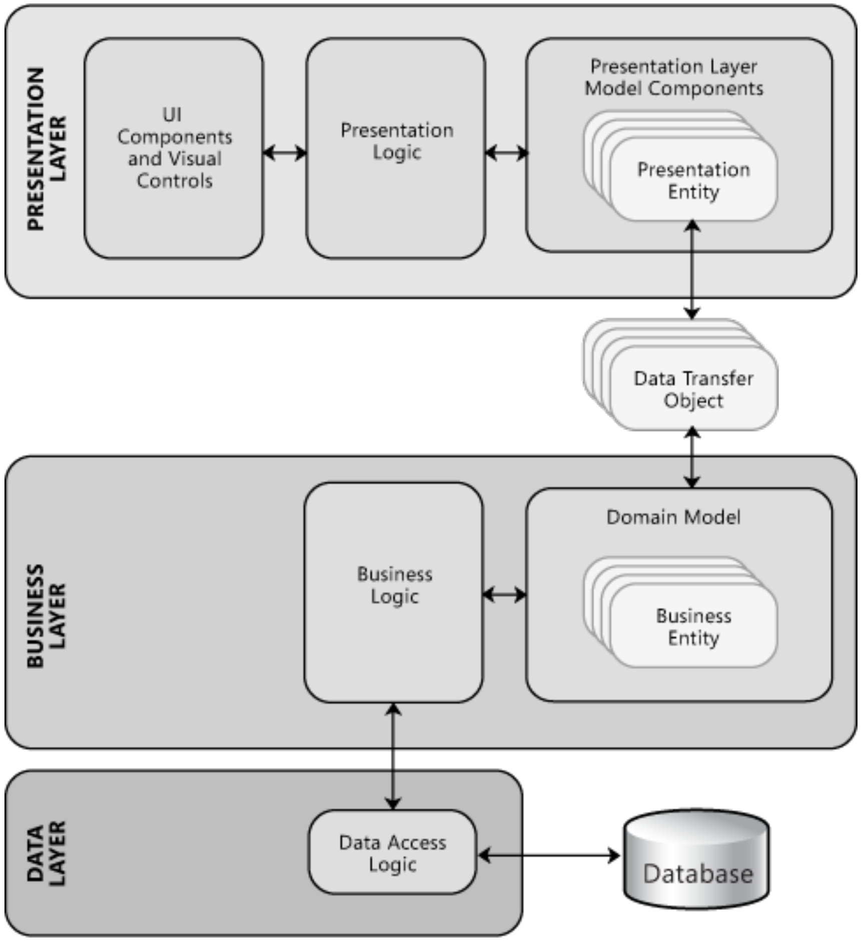
- **Presentation Model Components**
  - Presentation model components represent data from your business layer in a consumable format for your UI and presentation logic components in the presentation layer.
  - Determine if you require presentation model components. Typically, you might use presentation layer models if the data or the format to be displayed is specific to the presentation layer, or if you are using a separated presentation pattern such as MVP or MVC.
  - If you are working with data-bound controls, design or choose appropriate presentation model components that you can easily bind to UI controls. If using custom objects, collections, or data sets as your presentation model component format, ensure that they implement the correct interfaces and events to support data binding.
  - If you perform data validation in the presentation layer, consider adding the code for this to your presentation model components. However, also consider how you can take advantage of centralized validation code or code libraries.
  - Consider the serialisation requirements for the data you will pass to your presentation model components if this data will be passed over the network or stored on disk on the client.

Designing Presentation Components

- **Presentation Entities**
  - Presentation model components should, where possible, encapsulate both the data from your business layer, and business logic and behavior
  - This helps to ensure data consistency and validity in the presentation layer, and helps to improve the user's experience
  - Choose a suitable data type:
    - **Custom class:** Use a custom class if you want to represent your data as a complex object that maps directly to your business entities
    - **Array** and **Collection**: Use an array or a collection when you must bind data to controls such as list boxes and drop-down lists that use single column values
    - **DataSet** and **DataTable:** Use a DataSet or a DataTable when you are working with simple table-based data with data-bound controls such as grids, list boxes, and drop-down lists
    - **XML:** This format is useful when working with a Web client, where the data can be embedded in a Web page or retrieved via a Web service or HTTP request

Designing Presentation Components

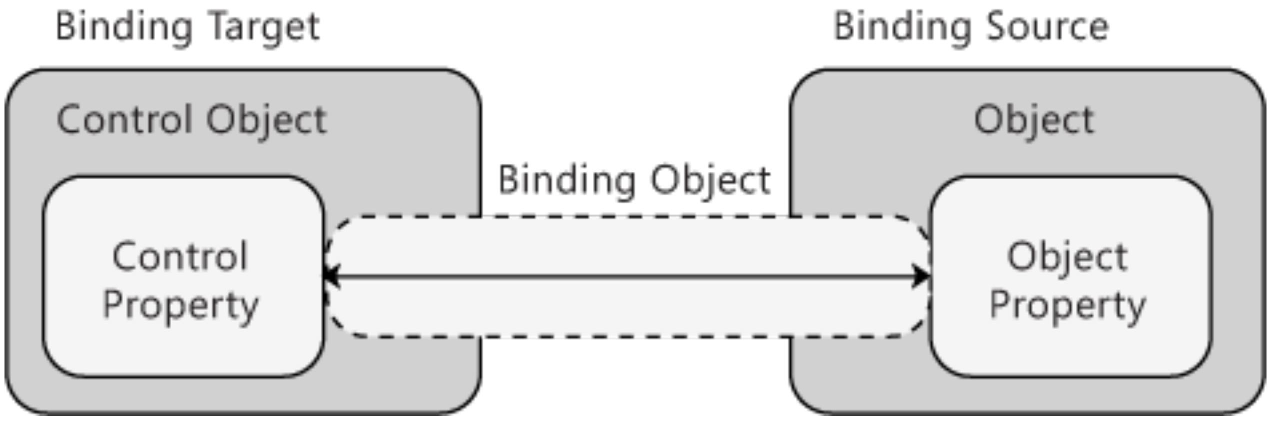
- **Presentation Entities**
  - When the business layer and presentation layer are both located on the client, a typical scenario for rich client applications, you will usually consume the business entities directly from the business layer
  - When the business layer is located on a separate tier from the presentation layer, you may be able to consume the business entities in the presentation tier by serializing them across the network using data transfer objects, and then resurrecting them as business entity instances on the presentation tier
  - Alternatively, you can resurrect the data as presentation entities if the required format and behavior differs from that of the business entities



Designing Presentation Components

Step 5 – Determine the Binding Requirements

- Data binding provides a way to create a link between the controls in the user interface and the data or logic components in your application
- Data binding allows you to display and interact with data from databases as well as data in other structures, such as arrays and collections
- Data binding is the bridge between a binding target (typically a user interface control) and a binding source (typically a data structure, model, or presentation logic component)



Designing Presentation Components

Step 5 – Determine the Binding Requirements

- **One-way binding**
  - Changes to the source property automatically update the target property, but changes to the target property are not propagated back to the source property
  - This type of binding is appropriate if the control being bound is implicitly read-only
  - An example of a one-way binding is a stock ticker
  - If there is no need to monitor changes to the target property, using one-way binding avoids unnecessary overhead
- **Two-way binding**
  - Changes to either the source property or the target property automatically update the other
  - This type of binding is appropriate for editable forms or other fully interactive UI scenarios

Designing Presentation Components

Step 6 – Determine the Error Handling Strategy

- Design a centralized exception handling strategy
- Log exceptions
- Display user friendly messages
- Allow retry
- Display generic messages

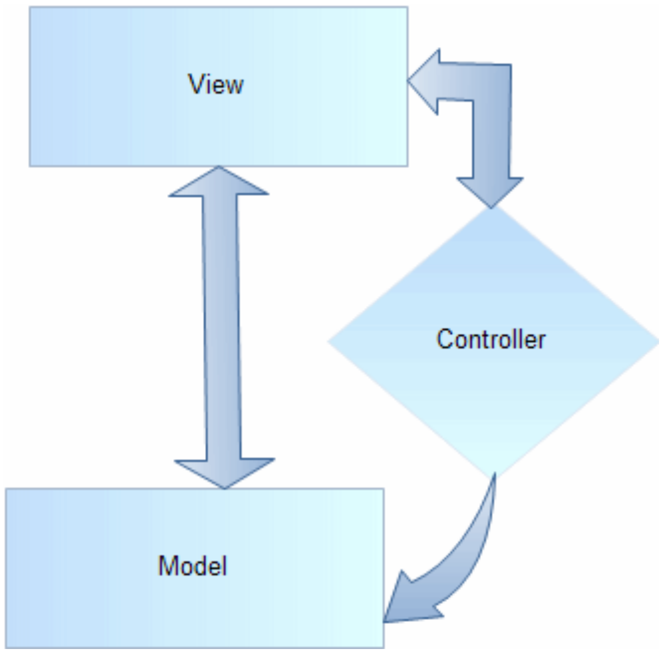
- Step 7 – Determine the Validation Strategy
  - Accept known good (Allow list, or positive validation)
  - Reject known bad (Block list, or negative validation)
  - Sanitize

Design Considerations

- **Choose the appropriate UI technology.** Determine if you will implement a rich (smart) client, a Web client, or a rich Internet application (RIA). Base your decision on application requirements, and on organizational and infrastructure constraints.
- **Use the relevant patterns.** Review the presentation layer patterns for proven solutions to common presentation problems.
- **Design for separation of concerns.** Use dedicated UI components that focus on rendering and display. Use dedicated presentation entities to manage the data required to present your views. Use dedicated UI process components to manage the processing of user interaction.
- **Consider human interface guidelines.** Review your organization’s guidelines for UI design. Review established UI guidelines based on the client type and technologies that you have chosen.
- **Adhere to user-driven design principles.** Before designing your presentation layer, understand your customer. Use surveys, usability studies, and interviews to determine the best presentation design to meet your customer’s requirements.

MVC and MVP

- Model View Controller (MVC) and Model View presenter (MVP)
- MVC architecture is one of the oldest patterns available for achieving the separation of concerns
- MVC consists of three layers: **Model**, **View** and **Controller**
- **Model**
  - Model receives input from the controller and decides what data it needed to fulfill the request raised by the controller
  - Model is also capable of doing transactions with databases or other persistent storage as needed
  - Once it gathers the required information, it informs the view (via controller) about the new data by raising appropriate events



MVC and MVP

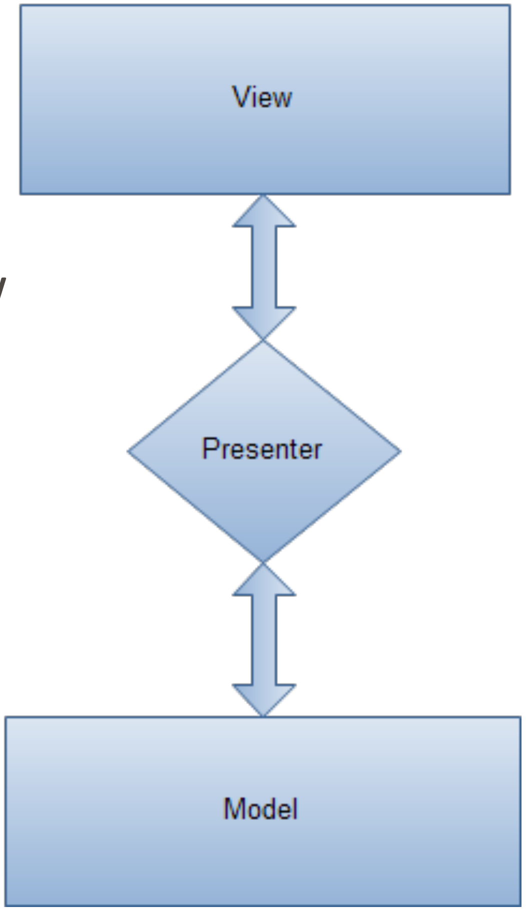
- **View**
  - View hosts all the UI controls needed by the end-user to interact with the application
  - Its job is to monitor end-user gestures and pass it to the controller for further handling
  - View handles the event notification received from the Model and may request for any new data required to render it on the screen
  - However, in some variations, Controller can be designated to retrieve dataset from the Model, format it and send it to the View
  - Here the view's job is to render the data received from the view without any other processing

MVC and MVP

- **Controller**
  - Controller can be imagined as an extension or a personal assistant to the view, all the events originated from the view is handled by the controller
  - Controller will also inform the Model on behalf of view that some event happened on the view and some new data might be required

MVC and MVP

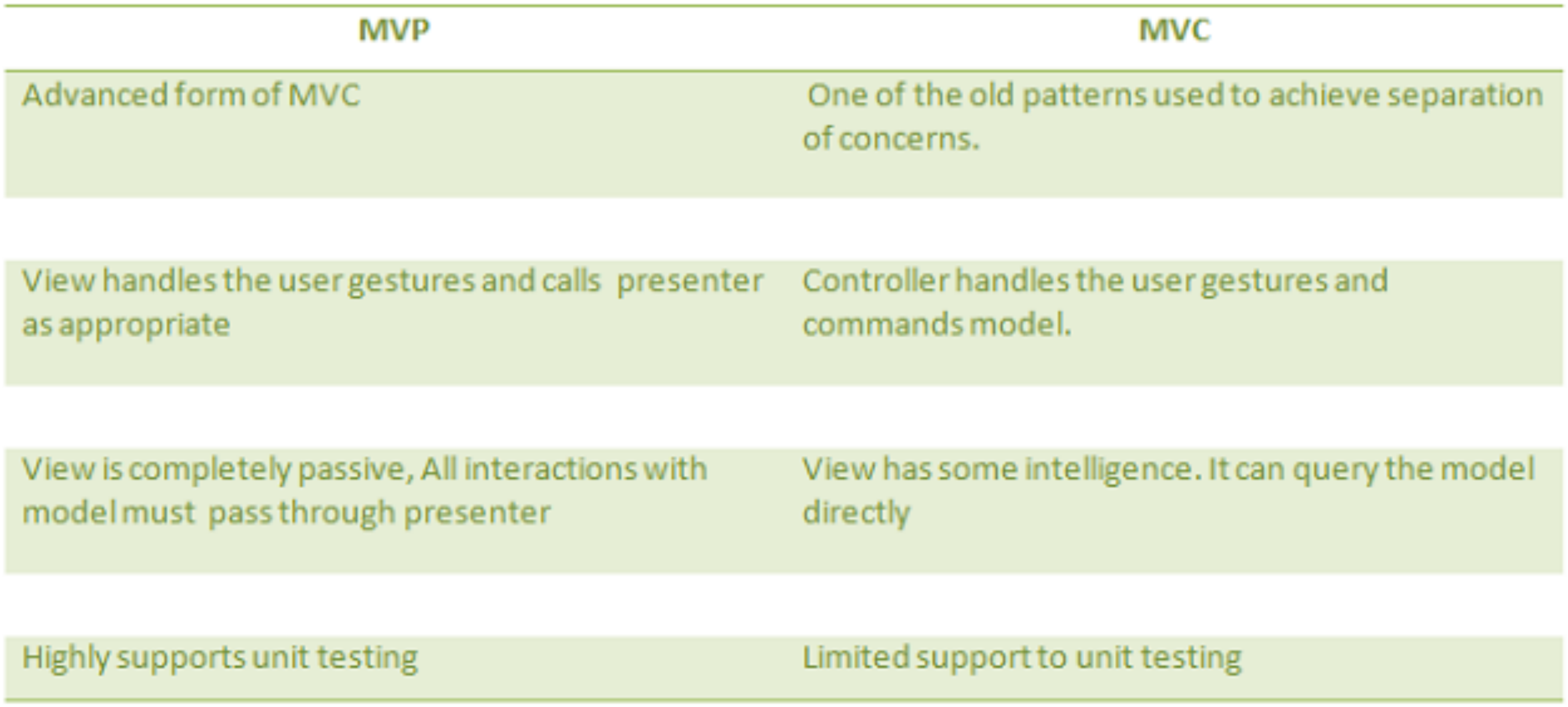
- **(M)odel (V)iew (P)resenter** is an evolved version of MVC
- In this pattern, view receives the UI events and calls the presenter as needed
- Presenter is also responsible for updating the view with the new data generated by the model
- **Model**
  - Model can be thought of as the interface to the data
  - Any part of the program which needs some data to work on must go through the interface or functions defined by the developer who is maintaining the model part
  - Typically, model houses all the validation routines for the data submitted by the end user



MVC and MVP

- **View**
  - View is the part where end user interacts
  - The development of this part can be delegated to a specialized designer
  - A program may have any number of views
- **Presenter**
  - Presenter acts as an intermediary to make the decoupling possible
  - All the business logic required for responding to a user event is written inside the Presenter layer
  - Typically the view only has the event handler and the logic to call the appropriate presenter functions, helping the person working on the view to concentrate upon designing the user interface without worrying about the code behind file
  - Presenter is also responsible for retrieving the requested data from the model and formats it so that the view can render it without any overhead

MVC and MVP

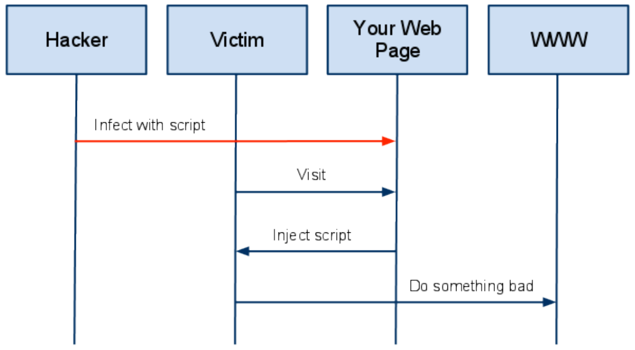


MVP	MVC
Advanced form of MVC	One of the old patterns used to achieve separation of concerns.
View handles the user gestures and calls presenter as appropriate	Controller handles the user gestures and commands model.
View is completely passive, All interactions with model must pass through presenter	View has some intelligence. It can query the model directly
Highly supports unit testing	Limited support to unit testing

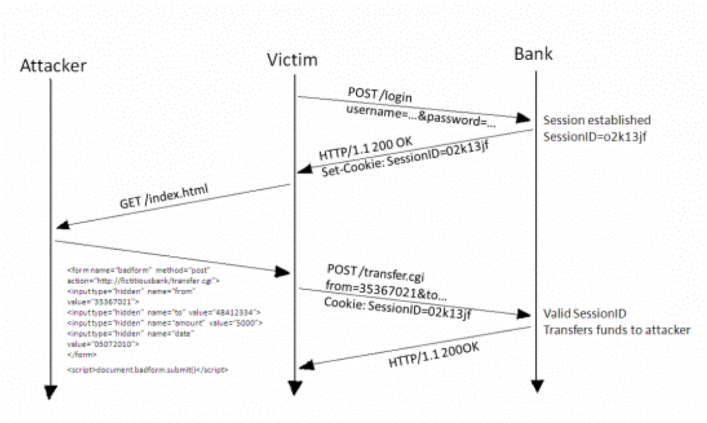
Application Security Basics

- Large spectrum of applications
  - Multi-user e-commerce applications
  - Community applications
    - Wikis
    - CMS
    - Blogs
  - Custom built applications
    - Framework developed applications
      - PHP / Ruby on Rails / .NET / Java EE

- Exploits / Attacks
  - XSS / CSRF / SQL Injection
  - Large potential attack surface



A High Level View of a typical XSS Attack



```
<form name="badform" method="post"
action="http://fictitiousbank/transfer.cgi">
<inputtype="hidden" name="from"
value="35367021">
<inputtype="hidden" name="to" value="48412334">
<inputtype="hidden" name="amount" value="5000">
<inputtype="hidden" name="date"
value="05072010">
</form>
<script>document.badform.submit()</script>
```

A High Level View of a typical CSRF Attack

Application Security Basics

- ABC of Threat Modelling
  - Step 1: Identify **A**ssets
  - Step 2: Identify **B**oundaries (Entry, Exit, Flows)
  - Step 3: Identify **C**ontrols

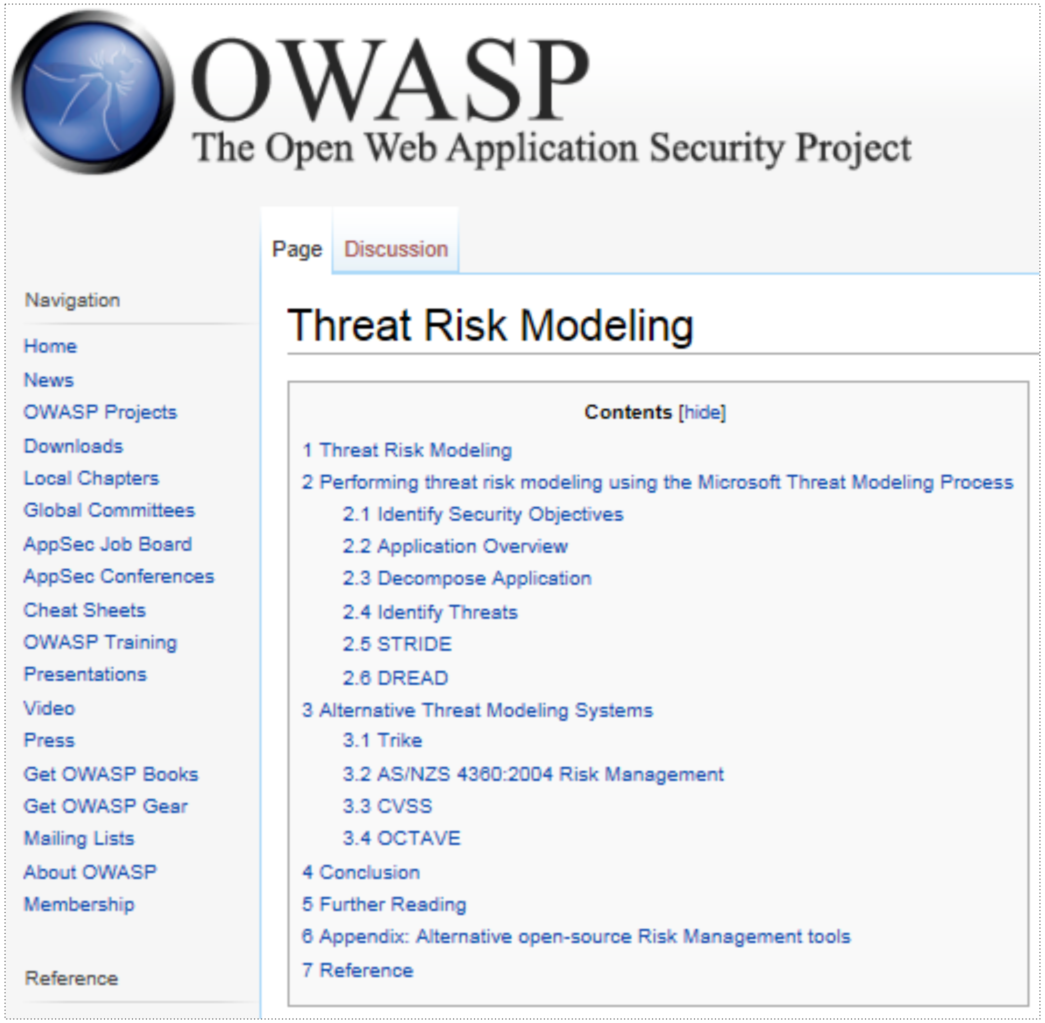
Application Security Basics: STRIDE

Threat	Controls
Spoofing user identity	• Use strong authentication • Do not store secrets (for example, passwords) in plaintext • Do not pass credentials in plaintext over the wire • Protect authentication cookies with Secure Sockets Layer (SSL)
Tampering with data	• Use data hashing and signing • Use digital signatures • Use strong authorisation • Use tamper-resistant protocols across communication links • Secure communication links with protocols that provide message integrity
Repudiation	• Create secure audit trails • Use digital signatures
Information Disclosure	• Use strong authorization • Use strong encryption • Secure communication links with protocols that provide message confidentiality • Do not store secrets (for example, passwords) in plaintext
Denial of Service	• Use resource and bandwidth throttling techniques • Validate and filter input
Elevation of Privilege	• Follow the principle of least privilege and use least privileged service accounts to run processes and access resources

Application Security Basics: Threat Model

- Other Models

http://owasp.com/index.php/Threat_Risk_Modeling

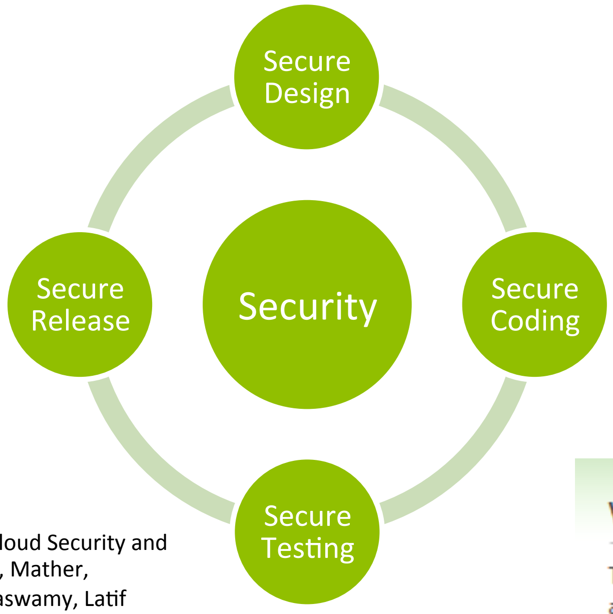

OWASP
The Open Web Application Security Project

Threat Risk Modeling

Contents [hide]

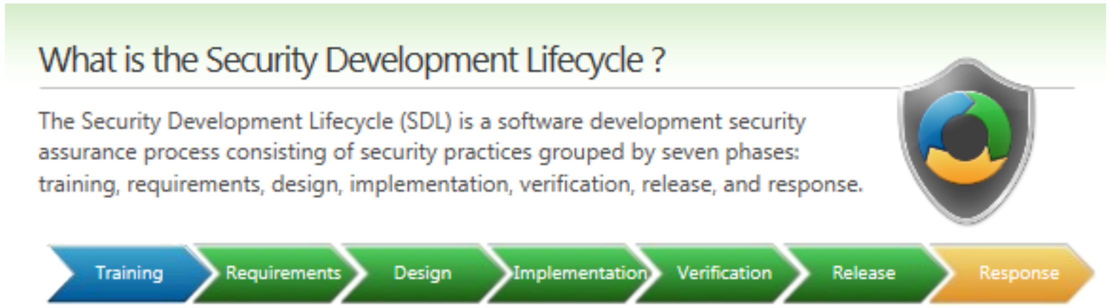
Application Security Basics: SDL

- Application Level Security
  - The Security Development Life Cycle



From Cloud Security and Privacy, Mather, Kumaraswamy, Latif

Security is an integral part of the development process



What is the Security Development Lifecycle ?

The Security Development Lifecycle (SDL) is a software development security assurance process consisting of security practices grouped by seven phases: training, requirements, design, implementation, verification, release, and response.

National College of Ireland

Questions?

Mikhail Timofeev
Office 3.18
MTimofeev@ncirl.ie