CSE 301
Software Design and Architecture

5. Interaction-Oriented & Component Based Architectures

Dr. Salisu Garba
Salisu.garba@slu.edu.ng

Outline

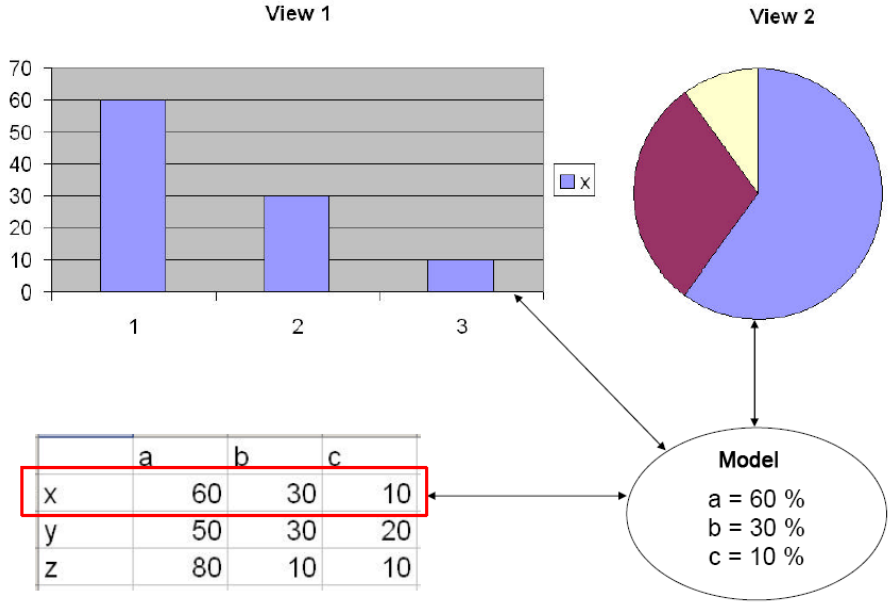
- Interaction-Oriented Software Architectures
  - MVC
- Component Based Architecture
- The architecture of User Interfaces
  - Interface Design
  - Styles of interfaces
  - Design Considerations
  - Evaluation of Design
  - User interface Design principles
  - Design Guidelines

Interaction-Oriented Software Architectures

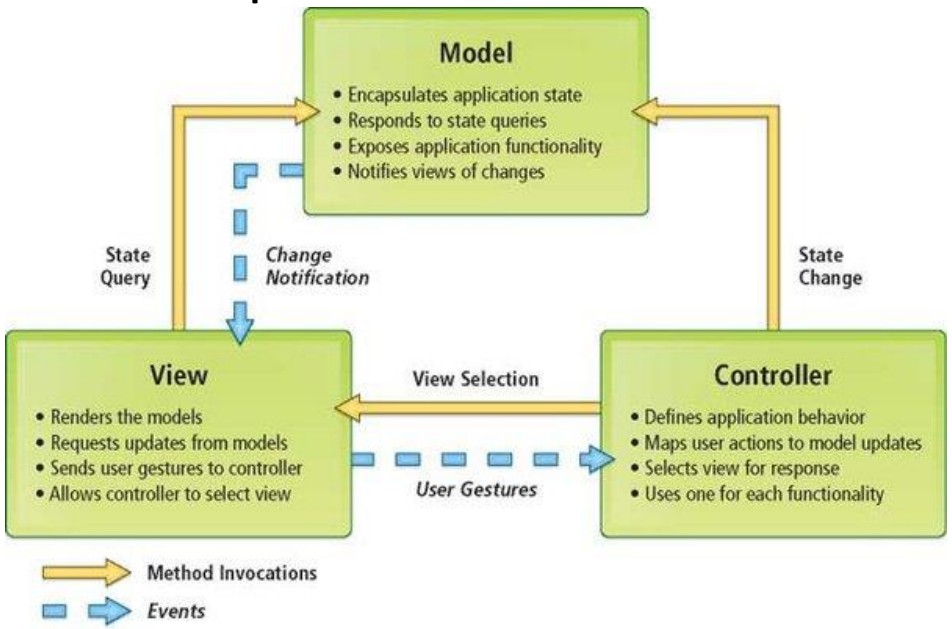
- The interaction-oriented software architecture decomposes the system into three major partitions:
  - data module,
  - control module, and
  - view presentation module.
- Each module has its own responsibilities.
- The key point of this architecture is its separation of user interactions from data abstraction and business data processing.
- The **data module** provides the data abstraction and all core business logic on data processing.
- The **view presentation module** is responsible for visual or audio data output presentation and may also provide user input interface when necessary.
- The **control module** determines the flow of control involving view selections, communications between modules, job dispatching, and certain data initialization and system configuration actions.

Model-View-Controller (MVC)

- Architectural design pattern which works to separate data and UI for a more cohesive and modularized system
- The MVC pattern was originally proposed in the 1980s as an approach to GUI design that allowed for
  - multiple presentations of an object and
  - separate styles of interaction with each of these presentations.



	a	b	c
x	60	30	10
y	50	30	20
z	80	10	10



Model-View-Controller

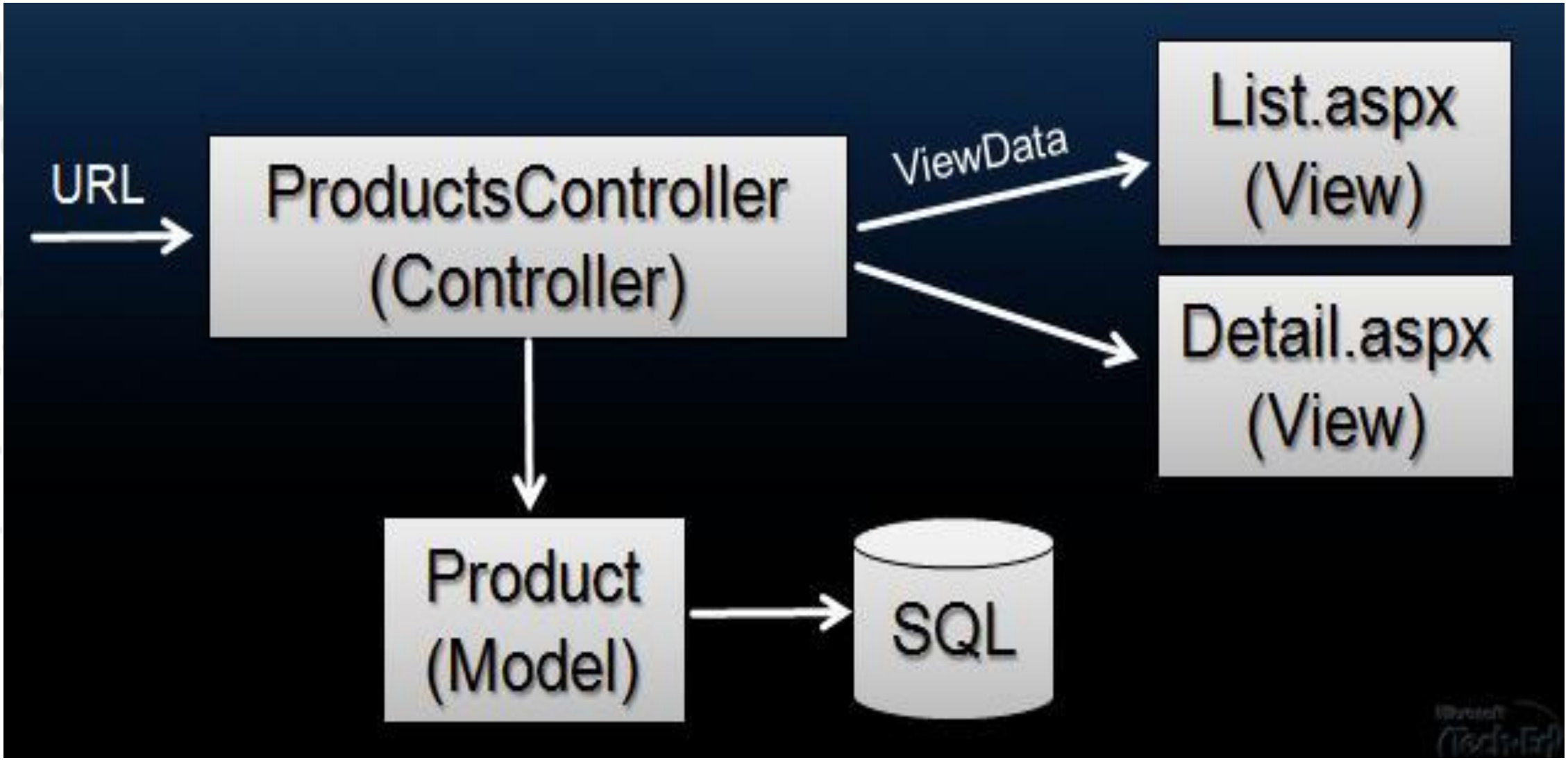
- The model is another name for the application logic layer
  - The domain-specific representation of the information on which the application operates.
  - Application (or domain) logic adds meaning to raw data
  - Many applications use a persistent storage mechanism (such as a database) to store data.
- The View renders the model into a form suitable for interaction, typically a user interface element.
  - MVC is often seen in web applications, where the view is the HTML page and the code which gathers dynamic data for the page.
- Processes and responds to events, typically user actions, and may invoke changes on the model and view.

Example Control Flow in MVC

- Though MVC comes in different flavors, the control flow generally works as follows:

1. The user interacts with the user interface in some way (e.g., user presses a button)
2. A controller handles the input event from the user interface, often via a registered handler or callback.
3. The controller accesses the model, possibly updating it in a way appropriate to the user's action (e.g., controller updates user's shopping cart).
4. A view uses the model to generate an appropriate user interface (e.g., view produces a screen listing the shopping cart contents).
   - The view gets its own data from the model.
   - The model has no direct knowledge of the view.
5. The user interface waits for further user interactions, which begins the cycle anew.

Example: Model 2 and ASP.NET



Pros & Cons of MVC

PROS

- ✓ Easier To Test
- ✓ Multiple Components Creation Feasibility
- ✓ Offers Clean Separation Of Concerns
- ✓ Less Complicated
- ✓ Phenomenally Well For Web Applications

CONS

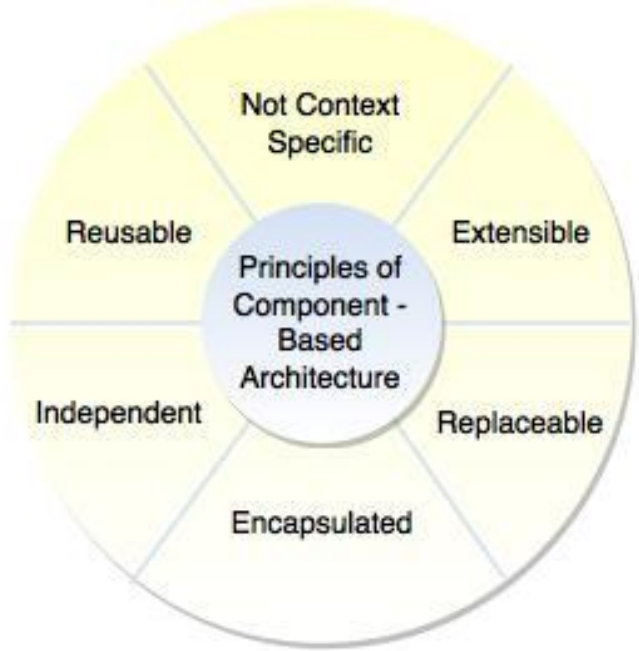
- ✗ No Formal Validation Support
- ✗ Managing Data Get Increasingly Complicated
- ✗ Challenging For Modern UI
- ✗ Hard To Reuse And Implement Tests

Component Based Software Architecture

- Component-Based Architecture is a branch of software engineering which provides a higher level of abstraction than object-oriented design principles.
- Component-Based architecture does not focus on issues such as communication protocol and shared state.
- This architecture focuses on the decomposition of the design into logical components which contain events, methods and properties.
- Component-Based architecture divides the problem into sub-problems and each problem associated with component partitions.

Principles of Component-Based Architecture

1. Extensible: A component can be extended from existing components to provide new behavior.
2. Replaceable: Components may be readily substituted with other similar components.
3. Encapsulated: Components exposes interfaces which allow the caller to use its functionality and hides the details of the internal processes or any internal variables or state.
4. Independent: Components are independent. It can be designed to have minimal dependencies on other components.
5. Reusable: Components are designed to be reused in different scenarios in different applications.
6. Not Context Specific: Components are designed to operate in different environments and contexts. Specific information such as state data, should be passed to the component instead of being included in or accessed by the component.



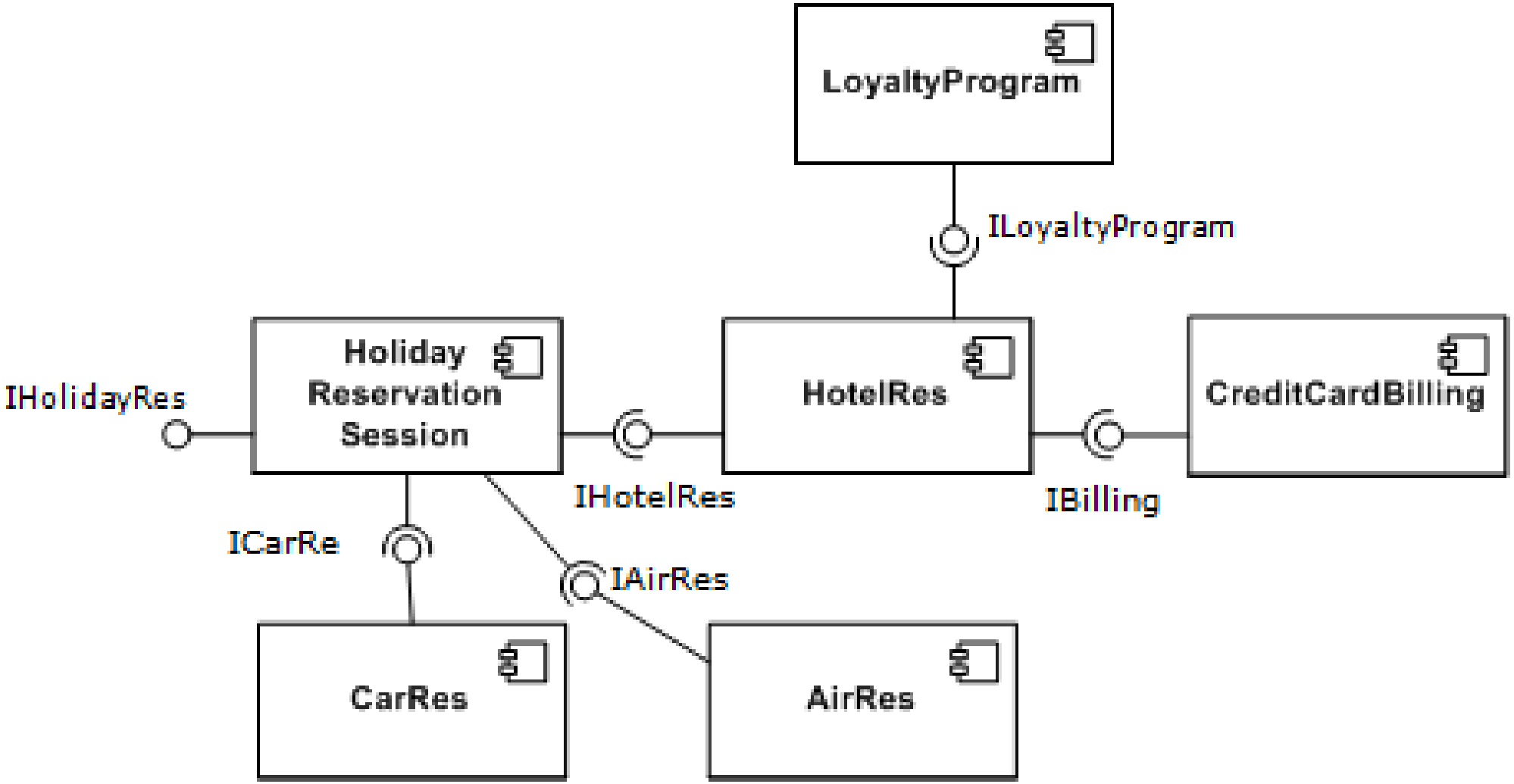
Component

Services	Components
A unit of functionality that is design to be deployed independently and reused by multiple systems.	A unit of functionality that is designed to serve as a part of a system or application
Services are ideal for highly resilient (loosely coupled systems)	Components are designed to fit together to deliver functionality
Service can be made from a components	Component may call services

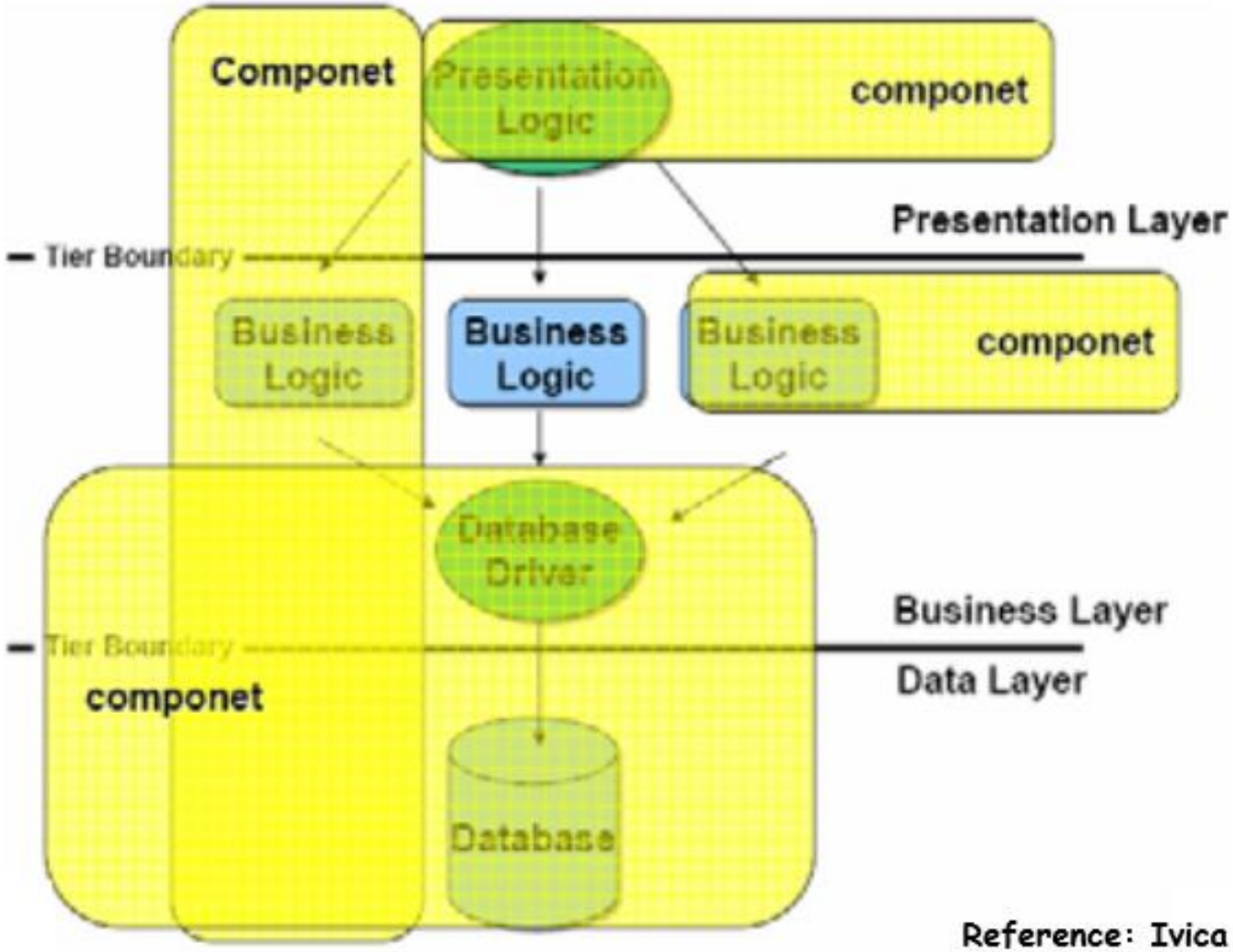
Component

- For instance, consider the web page with elements such as header, search bar, content body, etc. When considering the component-based architecture, all these are independent of each other and called components.
- Components can be made of:
- Source code
  - Classes – one or more, possibly related
- Executable code
  - Object code
  - Virtual object code
- Other files
  - Images, text, indices

Component


LoyaltyProgram
ILoyaltyProgram
Holiday Reservation Session
IHolidayRes
HotelRes
CreditCardBilling
IHotelRes
IBilling
ICarRe
IAirRes
CarRes
AirRes

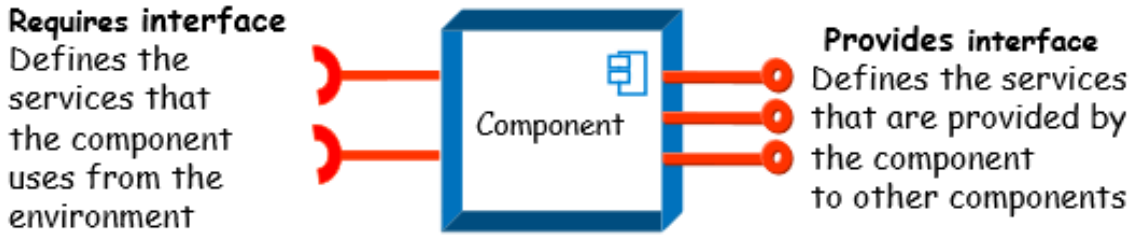
N tier Architecture



Reference: Ivica Crnkovic

Connectors

- Connectors connect components, specifying and ruling their interaction.
- Component interaction can take the form of
  - method invocations,
  - asynchronous invocations such as
    - event listener and registrations,
    - broadcasting,
  - message-driven interactions,
  - data stream communications, and
  - other protocol specific interactions.

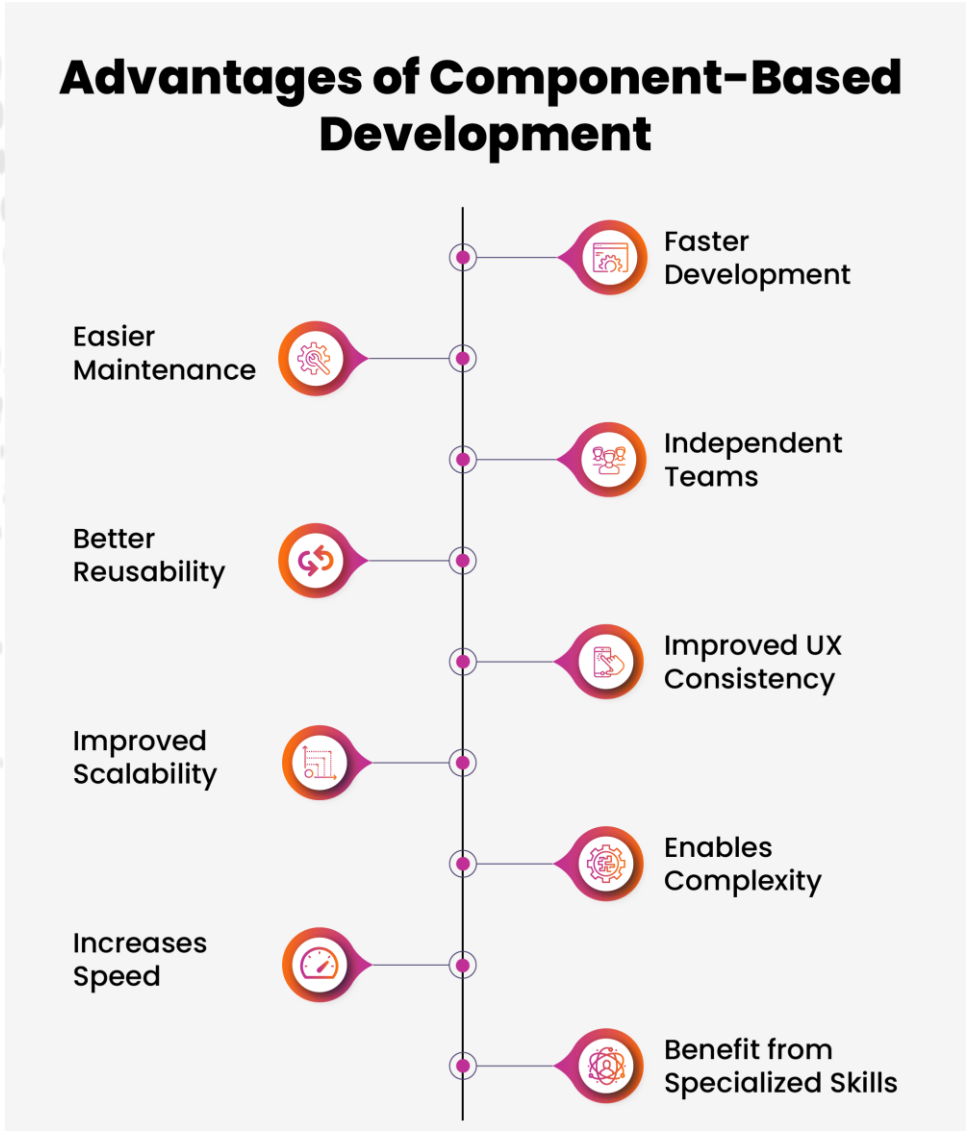


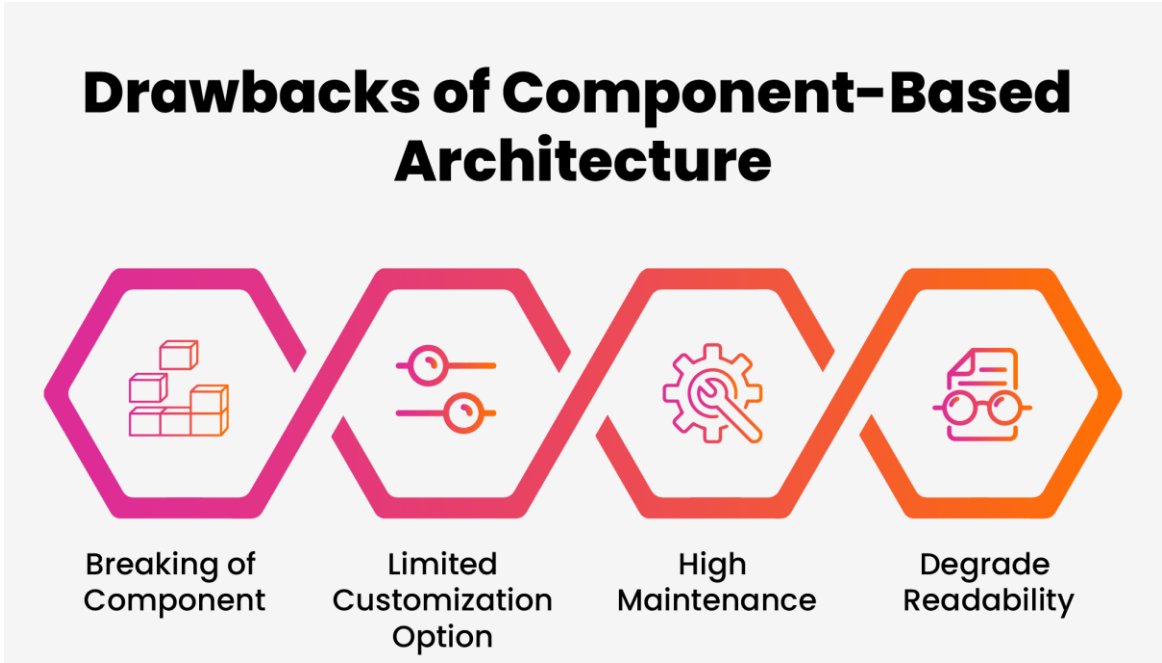
- The interaction type is specified by the interfaces of the components.

Component Model

- A component model is a definition of standards for component implementation, documentation and deployment
  - Examples of component models
    - EJB model (Enterprise Java Beans)
    - .NET model
    - Corba Component Model
- The component model specifies how interfaces should be defined and the elements that should be included in an interface definition
- Different application domains have different needs for component-based systems
  - Different non-functional properties: performance, security, reliability, scalability, etc.

Benefits


Advantages of Component-Based Development
Faster Development
Easier Maintenance
Independent Teams
Better Reusability
Improved UX Consistency
Improved Scalability
Enables Complexity
Increases Speed
Benefit from Specialized Skills


Drawbacks of Component-Based Architecture
Breaking of Component
Limited Customization Option
High Maintenance
Degrade Readability

The architecture of User Interfaces

- The architecture of user interfaces supports its look and feel and usability aspects.
- The evolution from textual user interfaces to graphical user interfaces can give us an idea about why the look and feel and the usability came to be.
- A user interface itself, especially the graphical user interface, is sophisticated.
- It needs codes not only for
  - displaying,
  - arranging, and
  - laying out all buttons, labels, and components but also
  - for linking these elements in the user interface with the internal functions of the software.

Software User Interface Design

- User interface is the front-end application view to which user interacts in order to use the software.
- User interface is part of software and is designed such a way that it is expected to provide the user insight of the software.
- UI provides fundamental platform for human-computer interaction.
- UI can be graphical, text-based, audio-video based, depending upon the underlying hardware and software combination.
- The software becomes more usable and popular if its user interface is:
  - Attractive
  - Simple to use
  - Responsive in short time
  - Clear to understand
  - Consistent on all interfacing screens

Look and Feel (Syntax) of User Interfaces

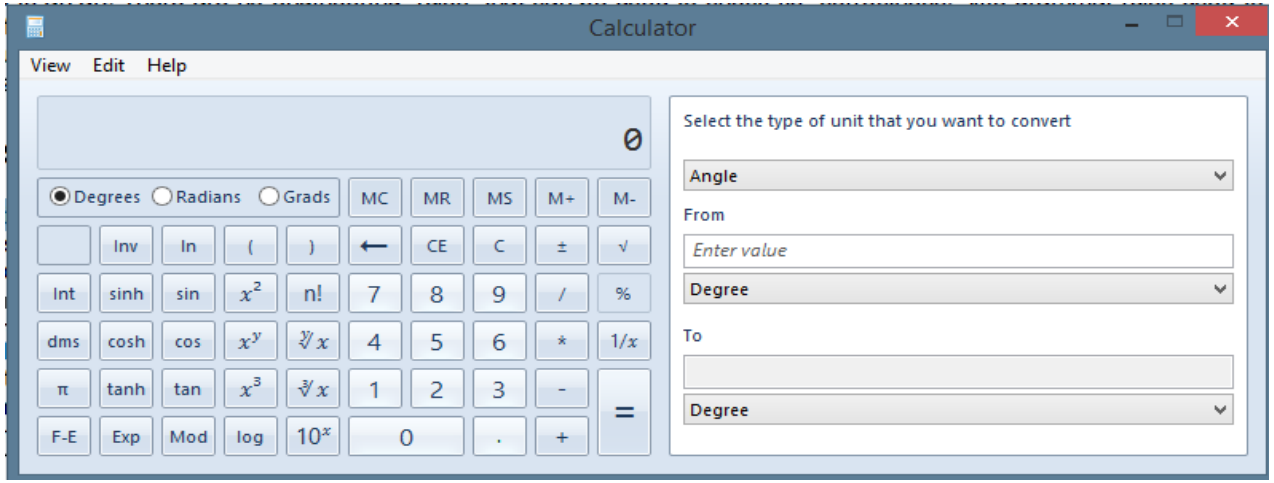
- The design and implementation of the look and feel are based on the belief that most humans know how to appreciate artwork.

- In general, graphical user interfaces have three styles:
  - static,
  - dynamic, and
  - customizable.
- Many programming languages, such as Visual Basic, Java, etc., provide built-in components and layout managers for implementing user interfaces.

- Many IDEs support drag-and-drop metaphors for making the implementation of graphical user interfaces easier.

Static Style of User Interfaces

- The static style of user interfaces refers to those user interfaces that have statically “prefixed” components and compositions.
- The components used on the user interface include button, radio button.
- Example
  - Windows calculator



- The frequently used digits and commands are in the right-hand column.
- The infrequently used commands, including scientific, statistic, and memory access commands, are in the left-hand column.
- The digits [0-9] are grouped into a panel that is almost the same as the digit panels on other popular devices like phones.

Static Style

- Many user interfaces fall into this style, such as text editors and IDEs for software development.
- These user interfaces are well-defined, in the sense that they are usually developed by professionals, and many are commercial products.
- Secondly, the IDEs have a clearly defined application domain and serving targets, that of supporting program development with a selected programming language;
  - an IDE deals with a certain set of menus, such as File, Edit, Search, Build, Run, and so on.
- The organization and layout of these user interfaces of IDEs and Editors can be 1 Dimensional, 2 Dimensional, 3 Dimensional, 4 Dimensional

Dynamic Style of User Interfaces

- Websites allow users to ask, browse, and search for information that the user would like to obtain.
- This kind of user interface also provides menus, tabs, and some static links.
- Specifically, these user interfaces leave space for displaying information, and the information may embed many “dynamic” links, known as hyperlinks.

- These dynamic hyperlinks are not part of prefixed UI components but are dynamic occurrences of components that guide users from one web page to another or from one website to the world.

Dynamic Style of User Interfaces

- Consequently, a website needs to accept a user's query, conduct searching, and display the results with further searching links that dynamically depend on the displayed contents.

- A browser (e.g., MS Internet Explorer) can be used to access any available web object uniquely linked by the Unified Resource Locator (URL).

- Today, almost all organizations have their own websites and so do many individuals. These websites provide different content and embedded hyperlinks to guide users to search more URLs.

Dynamic Style of User Interfaces

- Web pages, with their hyperlinks and displayed content, need to have a structure, an organization, and a labeling system.
- The structure determines the appropriate levels of granularity for the information units.
- The organization involves grouping those components into meaningful and distinctive categories.
- The labeling system can be used for figuring out what to call those categories for the series of navigation links that lead to them.

Customizable Style of User Interfaces

- A developing trend of user interfaces is the customizable style. This style of user interface is tightly integrated with the entire software system.
  - This means that the entire software system supports all needed functions with a specific customizable user interface.

- Many user interfaces that belong to the static style also have some setting functions for customizing the user interface appearance, such as text colors, contents of tool bars, etc.
- However, they do not change the functionality provided via the user interface.

Latest Interactive GUIs

- Multi-touch sensing enables a user to interact with a system with more than one finger at a time.
- Such sensing devices are also inherently able to accommodate multiple users simultaneously, which is especially useful for larger interaction scenarios such as interactive walls and tabletops.
- The technique is force sensitive, and provides unprecedented resolution and scalability that can be used to create sophisticated multipoint widgets for displaying panels large enough to accommodate both hands and multiple users.

Design Considerations of User Interfaces

- User-centered:
  - User interface design must take into account the needs, experiences, and capabilities of the system users.
  - A user interface must be a user-centered product.
  - In order to understand users' needs, prototyping is an essential step.
- Intuitive:
  - No matter how many variants there are, the central attributes remain the same: simplicity, intuitiveness, and usability.
  - The “best” user interface is one that people can figure out how to use quickly and effectively even without instructions.
  - Graphical user interfaces are better than textual user interfaces in the sense that a graphical user interface usually consists of windows, menus, buttons, and icons, and is operated by using a mouse.
- Consistency:
  - Designers must consider the physical and mental limitations of users of the software system.
  - They need to recognize the limitations on the size of short-term memory and to avoid overloading users with information.
  - The interface should use terms and concepts drawn from the experience of the anticipated class of users;
  - should be consistent in that comparable operations can be activated in the same way
- Integration:
  - The software system should integrate smoothly with other applications.
  - For example, different editors in the Windows operating system, such as MS Notepad and MS Office applications, can use “Clipboard” commands (Cut, Copy, and Paste) to directly perform data interchange.

Design guidelines of User Interfaces

- Strive for consistency
- Enable frequent users to use short-cuts
- Offer informative feedback
- Offer simple error handling
- Permit easy reversal of actions

Evaluation of User Interfaces

Possible usability attributes of a user interface include:

- How long does it take a new user to become productive with the user interface?
  - For example, a new user can master the usage of the user interface in a three-day training seminar.
- What percentage of the user interface components and the system functions are usable for users?
  - For example, some components of the user interface and functions of the system are “redundant.”
- What is the user's observation of the look and feel of the user interface?
  - For example, the user may operate the user interface without looking at it, the system is well implemented with consistency and integration, and the system uses colors to distinguish important information.
- How effective is the user interface at recovering from user errors?
  - For example, the system can check spelling and automatically correct spelling errors.
  - The system can format source codes and automatically import packages. The system can redo 50 step actions.

Evaluation of User Interfaces

- Can the user interface be customized and adapted to a new environment?
  - For example, the user interface can be switched from one version to the other for different users.
  - The user interface can incorporate additional components with plug-and-play.
- How well does the system response match the user's work practice?
  - For example, a right-click context sensitive menu saves user's actions.
- How tolerant is the system of user error?
  - For example, what happens if the user accidentally clicked two components or hit three keys at the same time? Will the system be aborted?
- Is the user interface complete?
  - For example, the system also includes a user guidance, tutorials, and help tools.

User Interface Design Principles

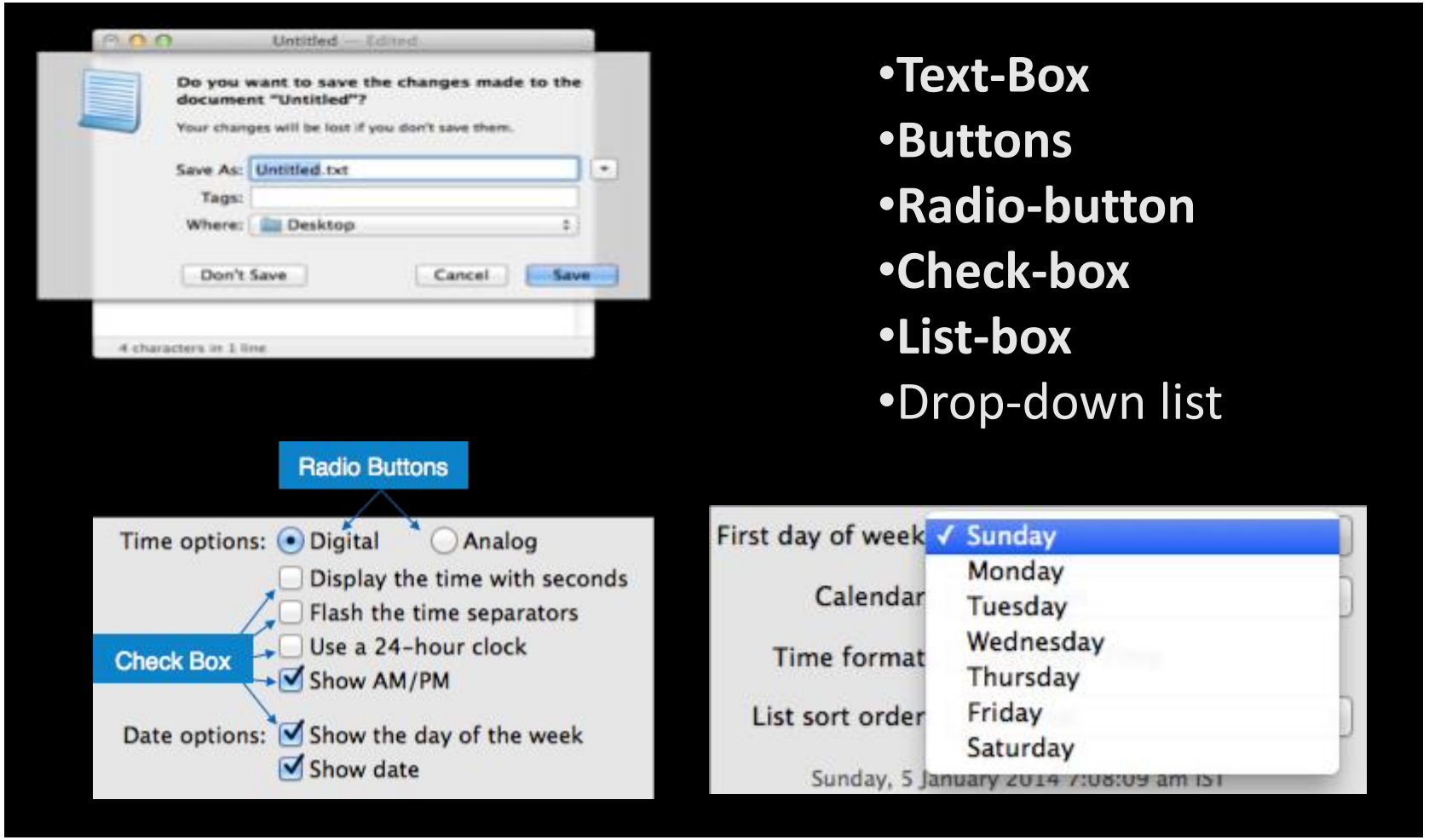
- Affinity:
  - Bring objects to life through good visual design
- Assistance:
  - Provide proactive assistance
- Availability:
  - Make all objects available at any time
- Encouragement:
  - Make actions predictable and reversible
- Familiarity:
  - Build on the user's prior knowledge
- Obviousness:
  - Make objects and controls visible and intuitive

User Interface Design Principles

- Personalization:
  - Enable the user to customize an interface
- Safety:
  - Keep the user out of trouble
- Satisfaction:
  - Create a feeling of progress and achievement
- Simplicity:
  - Do not compromise usability for functionality
- Support:
  - Place the user in control
- Versatility:
  - Support alternate interaction techniques.

Elements of GUI

Application specific GUI components



- Text-Box
- Buttons
- Radio-button
- Check-box
- List-box
- Drop-down list

Summary

- Interaction-Oriented Software Architectures
  - MVC
- Component Based Architecture
- The architecture of User Interfaces
  - Interface Design
  - Styles of interfaces
  - Design Considerations
  - Evaluation of Design

CSE301-SDA Assignment

- **Use any tools available E.g.** Lucidchart, Cacoo, EdrawMax, Microsoft Visio Pro, Astah, StarUML **to design the Software based on** 4 +1 View Model

- **Use any tools available E.g. FLUID, AppInventor (Android), LucidChart, Wavemaker, Visual Studio to design a GUI for your project**

- **Document/Compile/Zip and Submit to salisu.garba@slu.edu.ng**

- **Due Date: 20-05-2024**

Q & A

Any Question?