**Multidisciplinary Projects**

**Class 4a: Design**

Contents

[**Design** 2](#_Toc140044442)

[**Creativity and Design** 3](#_Toc140044443)

[**Models for System Architecture** 3](#_Toc140044444)

[**Subsystems** 3](#_Toc140044445)

[**Component** 4](#_Toc140044446)

[**Components as Replaceable Elements** 4](#_Toc140044447)

[**Components and Classes** 4](#_Toc140044448)

[**Package** 4](#_Toc140044449)

[**Node** 4](#_Toc140044450)

[**Example: Simple Web System** 4](#_Toc140044451)

[**Deployment Diagram** 5](#_Toc140044452)

[**Component Diagram: Interfaces** 5](#_Toc140044453)

[**Application Programming Interface (API)** 5](#_Toc140044454)

[**Architectural Style: Pipe** 5](#_Toc140044455)

[**Architectural Style: Client/Server** 5](#_Toc140044456)

[**Architectural Style: Repository** 6](#_Toc140044457)

[**Architectural Style: Repository with Storage Access Layer** 6](#_Toc140044458)

[**Time-Critical Systems** 7](#_Toc140044459)

[**Time Critical System: Architectural Style – Daemon** 7](#_Toc140044460)

[**Architectural Styles for Distributed Data** 7](#_Toc140044461)

[**Batch Processing with Master File Update** 7](#_Toc140044462)

[**First Attempt** 8](#_Toc140044463)

[**Criticisms of First Attempt** 8](#_Toc140044464)

[**Batch Processing: Edit and Validation** 8](#_Toc140044465)

[**Deployment Diagram: Validation** 9](#_Toc140044466)

[**Batch Processing: Master File Update-MFU** 9](#_Toc140044467)

[**Benefits of Batch Processing with MFU** 9](#_Toc140044468)

[**Architectural Style: MFU (Basic Version)** 9](#_Toc140044469)

[**Online Inquiry** 10](#_Toc140044470)

[**Online Inquiry: Use Case** 10](#_Toc140044471)

[**Architectural Style: Master File Update (Full)** 10](#_Toc140044472)

[**Example 2: Three Tier Architecture-TTA** 11](#_Toc140044473)

[**Web Server with Data Store** 11](#_Toc140044474)

[**Architectural Style: TTA** 11](#_Toc140044475)

[**Component Diagram** 12](#_Toc140044476)

[**TTA: Broadcast Searching** 12](#_Toc140044477)

[**Extending the Architecture of the Web** 12](#_Toc140044478)

[**Extending the Web with Executable Code that can be Downloaded** 12](#_Toc140044479)

[**Extending the Three Tier Architecture** 13](#_Toc140044480)

[**Example 3: Model/View/Controller (MVC)** 13](#_Toc140044481)

[**Model/View/Controller in Robotics** 13](#_Toc140044482)

[**Example 3: MVC for Mobile Apps** 14](#_Toc140044483)

[**Model** 14](#_Toc140044484)

[**View** 14](#_Toc140044485)

[**Controller** 15](#_Toc140044486)

[**External Services for Mobile Apps** 15](#_Toc140044487)

[**Apple’s Version of MVC** 15](#_Toc140044488)

[**An Exam Question** 16](#_Toc140044489)

# **Design**

* The requirements describe the function of a system as seen by the client.
* For a given set of requirements, the software development team must design a system that will meet those requirements.
* In practice requirements and design are interrelated. In particular, working on the design often clarifies the requirements. This feedback is a strength of the iterative and agile methods of software development.
* We have already looked at user interface design.
* The next few lectures look at the following aspects of design:
  + system architecture
  + program design
  + security
  + performance

# **Creativity and Design**

* Software development is a craft.
* Software developers have a variety of tools that can be applied in different situations.
* Part of the art of software development is to select the appropriate tool for a given implementation.
* System and program design are a particularly creative part of software development, as are user interfaces. You hope that people will describe your designs as “elegant”, “easy to implement, test, and maintain.”
* Above all strive for simplicity. The aim is find simple ways to implement complex requirements.
* System architecture is the overall design of a system
  + Computers and networks (e.g., monolithic, distributed)
  + Interfaces and protocols (e.g., http, ODBC)
  + Databases (e.g., relational, distributed)
  + Security (e.g., smart card authentication)
  + Operations (e.g., backup, archiving, audit trails)
* At this stage of the development process, you should also be selecting:
  + Software environments (e.g., languages, database systems, class frameworks)
  + Testing frameworks

# **Models for System Architecture**

* Our models for systems architecture are based on UML
* For every system, there is a choice of models
  + Choose the models that best model the system and are clearest to everybody.
* When developing a system, every diagram must have supporting specification.
  + The diagrams shows the relationships among parts of the system, but much, much more detail is needed to specify a system explicitly.

# **Subsystems**

* A subsystem is a grouping of elements that form part of a system.
* Coupling is a measure of the dependencies between two subsystems. If two systems are strongly coupled, it is hard to modify one without modifying the other.
* Cohesion is a measure of dependencies within a subsystem. If a subsystem contains many closely related functions its cohesion is high.
* An ideal division of a complex system into subsystems has low coupling between subsystems and high cohesion within subsystems.

# **Component**

* A component is a replaceable part of a system that conforms to and provides the realization of a set of interfaces.
* A component can be thought of as an implementation of a subsystem.
* UML definition of a component: “a distributable piece of implementation of a system, including software code (source, binary, or executable), but also including business documents, etc., in a human system.”

# **Components as Replaceable Elements**

* Components allow systems to be assembled from binary replaceable elements
* A component is bits not concepts
* A component can be replaced by any other component(s) that conforms to the interfaces
* A component is part of a system
* A component provides the realization of a set of interfaces

# **Components and Classes**

* Classes represent logical abstractions. They have attributes (data) and operations (methods).
* Components have operations that are reachable only through interfaces.

# **Package**

* A package is a general-purpose mechanism for organizing elements into groups.

# **Node**

* A node is a physical element that exists at run time and provides a computational resource, e.g., a computer, a smartphone, a router.
* Components may live on nodes.

# **Example: Simple Web System**

* Static pages from serve
* All interaction requires communication with server

A blue and white line

Description automatically generated

# **Deployment Diagram**

****

# **Component Diagram: Interfaces**

**A diagram of a network connection

Description automatically generated**

# **Application Programming Interface (API)**

* An API is an interface that is realized by one or more components.

A diagram of a web server

Description automatically generated

# **Architectural Style: Pipe**

* Example: A three-pass compiler
* Output from one subsystem is the input to the next.

A white rectangular object with black text

Description automatically generated

# **Architectural Style: Client/Server**

* Example: A mail system
* The control flows in the client and the server are independent.
* Communication between client and server follows a protocol.
* In a peer-to-peer architecture, the same component acts as both a client and a server.

A black line with a white background

Description automatically generated

# **Architectural Style: Repository**

* Advantages: Flexible architecture for data-intensive systems.
* Disadvantages: Difficult to modify repository since all other components are coupled to it.

A diagram of a diagram

Description automatically generated

# **Architectural Style: Repository with Storage Access Layer**

* Advantages: Data Store subsystem can be changed without modifying any component except the Storage Access.



# **Time-Critical Systems**

* A time-critical (real time) system is a software system whose correct functioning depends upon the results produced and the time at which they are produced.
* A hard real time system fails if the results are not produced within required time constraints e.g., a fly-by-wire control system for an airplane must respond within specified time limits
* A soft real time system is degraded if the results are not produced within required time constraints e.g., a network router is permitted to time out or lose a packet

# **Time Critical System: Architectural Style – Daemon**

* A daemon is used when messages might arrive at closer intervals than the time to process them
* Example: Web server
  + The daemon listens at port 80
  + When a message arrives it: spawns a processes to handle the message returns to listening at port 80

A diagram of a computer

Description automatically generated

# **Architectural Styles for Distributed Data**

* Replication: Several copies of the data are held in different locations.
  + Mirror: Complete data set is replicated
  + Cache: Dynamic set of data is replicated (e.g., most recently used)
* With replicated data, the biggest problems are concurrency and consistency

# **Batch Processing with Master File Update**

* Electricity utility customer billing (e.g., NYSEG)
* Telephone call recording and billing (e.g., Verizon)
* Car rental reservations (e.g., Hertz)
* Bank (e.g., Tompkins Trust)
* University grade registration (e.g., IU)
* Example: Electricity Utility Billing
* Requirements analysis identifies several transaction types:
  + Create account/ close account
  + Meter reading
  + Payment received
  + Other credits/ debits
  + Check cleared/ check bounced
  + Account query
  + Correction of error
  + etc., etc., etc.,

# **First Attempt**

* Each transaction is handled as it arrives.

A black arrow pointing to a rectangular object

Description automatically generated

# **Criticisms of First Attempt**

* Where is this first attempt weak?
  + All activities are triggered by a transaction.
  + A bill is sent out for each transaction, even if there are several per day.
  + Bills are not sent out on a monthly cycle.
  + Awkward to answer customer queries.
  + No process for error checking and correction.
  + Inefficient in staff time.

# **Batch Processing: Edit and Validation**

****

# **Deployment Diagram: Validation**

**A diagram of a computer system

Description automatically generated**

# **Batch Processing: Master File Update-MFU**

****

# **Benefits of Batch Processing with MFU**

* All transactions for an account are processed together at appropriate intervals, e.g., monthly.
* Backup and recovery have fixed checkpoints.
* Better management control of operations.
* Efficient use of staff and hardware.
* Error detection and correction is simplified.

# **Architectural Style: MFU (Basic Version)**

* Advantages:
  + Efficient way to process batches of transactions.
* Disadvantages:
  + Information in master file is not updated immediately.
  + No good way to answer customer inquiries.

**A diagram of a computer

Description automatically generated**

# **Online Inquiry**

* A customer calls the utility and speaks to a customer service representative.
* Customer service department can read the master file, make annotations, and create transactions, but cannot change the master file.

**A diagram of a computer

Description automatically generated**

# **Online Inquiry: Use Case**

* The representative can read the master file, but not make changes to it.
* If the representative wishes to change information in the master file, a new transaction is created as input to the master file update system.

A person holding a speech bubble

Description automatically generated

# **Architectural Style: Master File Update (Full)**

* Advantage:
  + Efficient way to answer customer inquiries.
* Disadvantage:
  + Information in master file is not updated immediately.

A diagram of a software system

Description automatically generated

# **Example 2: Three Tier Architecture-TTA**

* The basic client/server architecture of the web has:
  + a server that delivers static pages in HTML format
  + a client (known as a browser) that renders HTML pages
  + Both client and server implement the HTTP interface.
* Extend the architecture of the server so that it can configure HTML pages dynamically.

# **Web Server with Data Store**

* Advantage:
  + Server-side code can configure pages, access data, validate information, etc.
* Disadvantage:
  + All interaction requires communication with server

A close-up of a server

Description automatically generated

# **Architectural Style: TTA**

* Each of the tiers can be replaced by other components that implement the same interfaces.



# **Component Diagram**

**A diagram of a web server

Description automatically generated**

# **TTA: Broadcast Searching**

* This is an example of a multicast protocol.
* The primary difficulty is to avoid troubles at one site degrading the entire system (e.g., every transaction cannot wait for a system to time out).

****

# **Extending the Architecture of the Web**

* Using a three tier architecture, the web has:
* a server that delivers dynamic pages in HTML format
* a client (known as a browser) that renders HTML pages
* Both server and client implement the HTTP interface.
* Every interaction with the user requires communication between the client and the server.
* Problem 2 Extend the architecture so that simple user interactions do not need messages to be passed between the client and the server.

# **Extending the Web with Executable Code that can be Downloaded**

* Executable code in a scripting language such as JavaScript can be downloaded from the server
* Advantage:
  + Scripts can interact with user and process information locally.
* Disadvantage:
  + All interactions are constrained by web protocol

A black arrow pointing to a white square with a black line

Description automatically generated

# **Extending the Three Tier Architecture**

* In the three tier architecture, a web site has:
  + a client that renders HTML pages and executes scripts
  + a server that delivers dynamic pages in HTML format
  + a data store
* The three tier architecture with downloadable scripts is one the ways in which the basic architecture has been extended. There are some more:
  + Protocols: e.g., HTTPS, FTP, proxies
  + Data types: e.g., helper applications, plug-ins
  + Executable code: e.g., applets, servlets
  + Style sheets: e.g., CSS

# **Example 3: Model/View/Controller (MVC)**

* The definition of Model/View/Controller (MVC) is in a state of flux. The term is used to describe a range of architectures and designs.
  + Some are system architectures, where the model, view, and controller are separate components.
  + Some are program designs, with classes called model, view, and controller.
* We will look at three variants:
  + An MVC system architecture used in robotics.
  + A general purpose MVC system architecture used for interactive systems.
  + Apple’s version of MVC as a program design for mobile apps.

# **Model/View/Controller in Robotics**

* Controller: Receives instrument readings from the aircraft, updates the view, and sends controls signals to the aircraft.
* Model: Translates data received from and sent to the aircraft, and instructions from the user into a model of flight performance. Uses domain knowledge about the aircraft and flight.
* View: Displays information about the aircraft to the user on the ground and transmits instructions to the model via the controller.

A diagram of a model

Description automatically generated

# **Example 3: MVC for Mobile Apps**

****

# **Model**

* The model records the state of the application and notifies subscribers. It does not depend on the controller or the view.
  + stores the state of the application in suitable data structures or databases
  + responds to instructions to change the state information
  + notifies subscribers of events that change the state
  + may be responsible for validation of information

# **View**

* The view is the part of the user interface that presents the state of the interface to the user. It subscribes to the model, which notifies it of events that change the state.
  + renders data from the model for the user interface
  + provides editors for properties, such as text fields, etc.
  + receives updates from the model
  + sends user input to the controller
* A given model may support a choice of alternative views.

# **Controller**

* The controller is the part of the user interface that manages user input and navigation within the application.
  + defines the application behavior
  + maps user actions to changes in the state of the model
  + interacts with external services via APIs
  + may be responsible for validation of information
* Different frameworks handle controllers in different ways. In particular there are several ways to divide responsibilities between the model and the controller, e.g., data validation, external APIs.

# **External Services for Mobile Apps**

* Mobile apps often make extensive use of cloud-based external services, each with an API (e.g., location, validation). These are usually managed by the controller.

****

# **Apple’s Version of MVC**

* The diagram shows the model, view, and controller as components. In practice the Model-View-Controller is a program design with three major classes.



# **An Exam Question**

* A company that makes sports equipment decides to create a system for selling sports equipment online. The company already has a product database with description, marketing information, and prices of the equipment that it manufactures.
* To sell equipment online the company will need to create: a customer database, and an ordering system for online customers.
* The plan is to develop the system in two phases. During Phase 1, simple versions of the customer database and ordering system will be brought into production. In Phase 2, major enhancements will be made to these components.
* a) For the system architecture of Phase 1:
  + i). Draw a UML deployment diagram.



* a) For the system architecture of Phase 1:
  + i). Draw a UML deployment diagram.

A diagram of a product

Description automatically generated

* (b) For Phase 1:
* i). What architectural style would you use for the customer database? Repository with Storage Access Layer
* ii). Why would you choose this style? It allows the database to be replaced without changing the applications that use the database.