CS 5150 Software Engineering

System Architecture: Introduction

William Y. Arms
Design

The **requirements** describe the function of a system as seen by the client.

Given a set of requirements, the software development team must **design** a system that will meet those requirements.

In this course, we look at the following aspects of design:

- system architecture
- program design
- usability
- security
- performance

In practice these aspects are interrelated and many aspects of the design emerge during the requirements phase of a project. This is a particular strength of the iterative and incremental methods of software development.
Creativity and Design

Software development

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.

Creativity and design

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**

**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

The slides provide diagrams that give an outline of the systems, without the supporting specifications.

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.

For example, to specify a web plug-in, at the very least, the specification should include the version of the protocols to be supported at the interfaces, options (if any), and implementation restrictions.
Subsystems

Subsystem

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

*Note:* Some authors draw packages with a different shaped box:
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 server
- All interaction requires communication with server
Deployment Diagram

nodes

components

PersonalComputer

WebBrowser

WebServer

DeptServer
Component Diagram: Interfaces

WebBrowser

HTTP

WebServer

dependency

realization

interface
An **API** is an interface that is realized by one or more components.
An **architectural style** is system architecture that recurs in many different applications.

Architectural Style: Pipe

Example: A three-pass compiler

Lexical analysis → Parser → Code generation

Output from one subsystem is the input to the next.
Architectural Style: Client/Server

Example: A mail system

![Diagram showing a client/server architecture with mail client and mail server components.]

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.
Architectural Style: Repository

Advantages: Flexible architecture for data-intensive systems.

Disadvantages: Difficult to modify repository since all other components are coupled to it.
Architectural Style: Repository with Storage Access Layer

**Advantages:** Data Store subsystem can be changed without modifying any component except the Storage Access.
**Architectural Style: Model/View/Controller**

**Example: Control of a unmanned model aircraft**

Controller: Receives instrument readings from the aircraft 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.
Model/View/Controller: Autonomous Land Vehicle

Sensors
- GPS
- Sonar
- Laser

Signal processing

Model

View

Controller
- Steer
- Throttle
- Controls

Control signals
Model/View/Controller for Web Applications

1. User interacts with the user interface (e.g., presses a mouse button).
2. **Controller** handles input event from the user interface, (e.g., via a registered handler or callback) and converts the event into appropriate user action.
3. **Controller** notifies the **model** of user action, possibly resulting in a change in the **model's** state (e.g., update shopping cart).
4. **View** interacts with the **model** to generate an appropriate user interface response (e.g., list shopping cart's contents).
5. User interface waits for further user interactions.

*from Wikipedia 10/18/2009*
Model/View/Controller for Web Applications

WebBrowser control functions

HTTP

Input events

Controller

Model

WebBrowser
view functions

Response

HTML

View
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
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 process to handle the message
- returns to listening at port 80
**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**.

**Example:** The Domain Name System

For details of the protocol read:


When an application wants a continuous stream of data from a source that delivers data in bursts (e.g., over a network or from a disk), the software reads the bursts of data into a buffer and the application draws data from the buffer.
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.

- PersonalComputer
  - WebBrowser

- ShoppingServer
  - Product DB
  - Ordering system
  - Customer DB
(a) For the system architecture of Phase 1:
ii. Draw a UML interface diagram.
(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.
(b) For Phase 1:

iii Draw an UML diagram for this architectural style showing its use in this application.
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 server and client implement the HTTP interface.

**Problem**

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
Architectural Style: Three Tier Architecture

Each of the tiers can be replaced by other components that implement the same interfaces.
These components might be located on a single node.
Three Tier Architecture: Broadcast Searching

User

User interface service

Databases

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).
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.

**Problem 2**

Every interaction with the user requires communication between the client and the server.

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 protocols.
In this example, each package represents a related set of classes.
System Design Study 1 (continued)
Extending the Architecture of the Web

Using a three tier architecture with downloadable scripts, the web has:

- a server that delivers dynamic pages in HTML format
- a client (known as a browser) that renders HTML pages and executes scripts

Both server and client implement the HTTP interface.

**Problem 3**

Every interaction between the client and a server uses the HTTP protocol.

Extend the architecture so that other protocols can be used.
Web User Interface: Applet

- Any executable code can run on client
- Client can connect to any server
- Functions are constrained by capabilities of browser
Applet Interfaces

XYZInterface

WebBrowser

HTTP

WebServer
These examples (three tier architecture, downloadable scripts, and applets) are just some of the ways in which the basic architecture has been extended. Here are some more:

**Protocols:**
- HTTP, FTP, etc., proxies

**Data types:**
- helper applications, plug-ins, etc.

**Executable code:**
- Server-side code, e.g., servlets, CGI

**Style sheets:**
- CSS, etc.
Examples

- Electricity utility customer billing (e.g., NYSEG)
- Telephone call recording and billing (e.g., Verizon)
- Car rental reservations (e.g., Hertz)
- Stock market brokerage (e.g., Charles Schwab)
- E-commerce (e.g., Amazon.com)
- University grade registration (e.g., Cornell)
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.,
System Design Study 2 (continued)
First Attempt

Each transaction is handled as it arrives.
Criticalisms 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
System Design Study 2 (continued)
Batch Processing: Edit and Validation

Batches of incoming transactions → Data input

Edit & validation

errors

Batches of validated transactions → Master file

read only
Deployment Diagram: Validation

- **RawData**
  - **DataInput**
    - **EditCheck**
      - **ValidData**

- **MasterFile Check**
System Design Study 2 (continued)
Batch Processing: Master File Update

Validated transactions in batches → Sort by account → Batches of input data → Master file update

Errors

Checkpoints and audit trail

Bills

Reports
System Design Study 2 (continued)
Benefits of Batch Processing with Master File Update

- 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: Master File Update (basic)

Advantages:

Efficient way to process batches of transactions.

Disadvantages:

Information in master file is not updated immediately. No good way to answer customer inquiries.

Example: billing system for electric utility
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.
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.
Architectural Style: Master File Update (full)

Advantage:
Efficient way to answer customer inquiries.

Disadvantage:
Information in master file is not updated immediately.

Example: billing system for electric utility
Example: A small bank

- Transactions are received by customer in person, over the Internet, by mail or by telephone.
- Some transactions must be processed immediately (e.g., cash withdrawal), others are suitable for overnight processing (e.g., check clearing).
- Complex customer inquiries.
- Highly competitive market.
Real-time Transactions & Batch Processing

This is a combination of the Repository style and the Master File Update style.

Real-time transactions

Batch data input

Customer & account database
System Design Study 2 (continued)
Practical Consideration

- Can real-time service during scheduled hours be combined with batch processing overnight?
- How will the system guarantee database consistency after any type of failure?
  - reload from checkpoint + log
detailed audit trail
- How will transaction errors be avoided and identified?
- How will transaction errors be corrected?
- How will staff dishonesty be controlled?

These practical considerations may be major factors in the choice of architecture.
Many data intensive systems, e.g., those used by banks, universities, etc. are legacy systems. They may have been developed forty years ago as batch processing master file update systems and been continually modified.

- Recent modifications might include customer interfaces for the web, smartphones, etc.
- The systems will have migrated from computer to computer, across operating systems, to different database systems, etc.
- The organizations may have changed through mergers, etc.

Maintaining a coherent system architecture for such legacy systems is an enormous challenge, yet the complexity of building new systems is so great that it is rarely attempted.
In some types of system architecture, non-functional requirements of the system may dictate the software design and development process.
Continuous Operation

Many systems must operate continuously

- Software update while operating
- Hardware monitoring and repair
- Alternative power supplies, networks, etc.
- Remote operation

These functions must be designed into the fundamental architecture.
Example: Testing multi-threaded and parallel systems

Several similar threads operating concurrently:

- Re-entrant code -- separation of pure code from data for each thread
- May be real-time (e.g., telephone switch) or non-time critical

The difficult of testing real-time, multi-threaded systems may determine the entire software architecture.

- Division into components, each with its own acceptance test.
Developers of advanced time-critical software spend much of their effort developing the software environment:

- Monitoring and testing -- debuggers
- Crash restart -- component and system-wide
- Downloading and updating
- Hardware troubleshooting and reconfiguration
  etc., etc., etc.