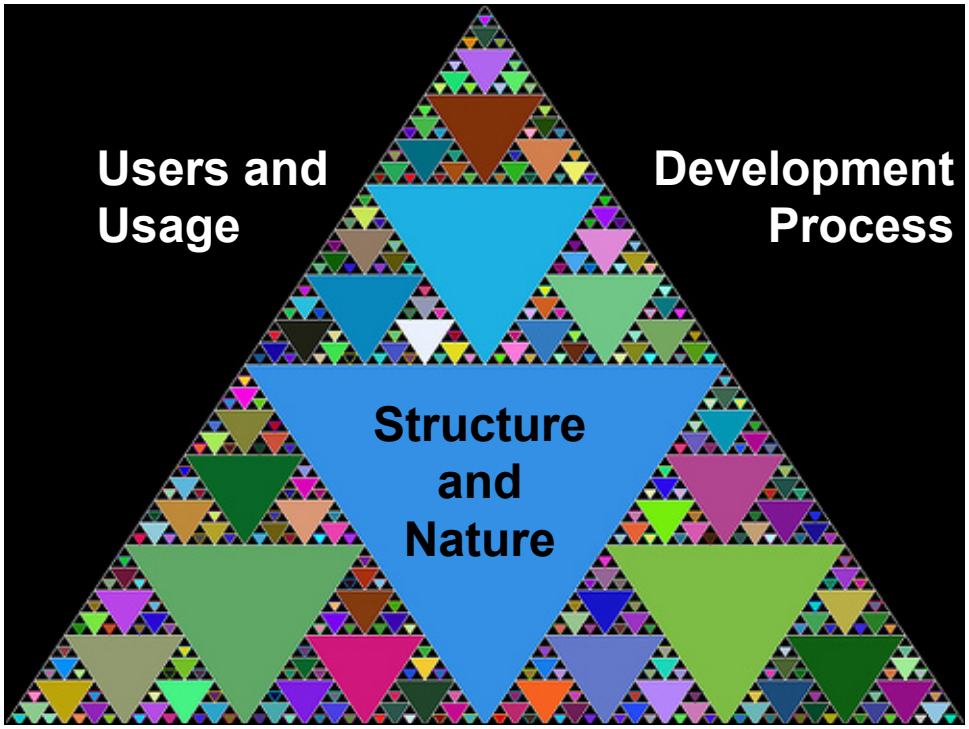
System Architectures

Web Application Development 2



Web Complexities

- **The structure and nature of the application**
  - The hypertext structure
  - The presentation and layout
  - Content usage and management
  - Service usage
- **The usage of the application**
  - Instant online access, with permanent availability
  - A wide variety of usually anonymous users
  - On a variety of devices
- **The development process behind the application**
  - by a multidisciplinary team including experts and novices
  - In a state of continuous development
  - A mixture of legacy technologies and immature technologies

Structure & Nature

- The **hypertext structure** means there is a need to
  - avoid user disorientation and cognitive overload
  - provide multiple paths to support users with different requirements
  - provide a good superstructure include search facilities, site map, and guided tours
- The **presentation or user interface** must be
  - self-explanatory with no user manuals for web sites!
  - aesthetically pleasing and adaptable to different contexts
    - ideally self-adapting, but somehow there must be a version of the user interface which works in each context
- **Content** delivery must be
  - fast, up-to-date, consistent and reliable
  - secure - particularly for financial transactions
  - adaptable to different contexts in terms of how much can be delivered

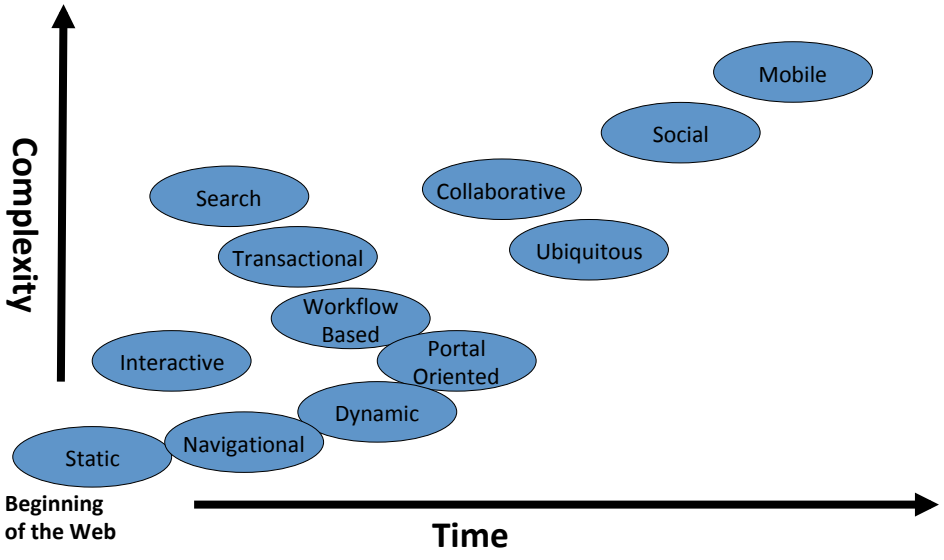
Users and Usage

- Users expect **immediate** and **fast** availability
- They also expect **permanent** availability
- They also expect access on a **variety of devices**
  – quality of service becomes an issue
- They have a very **diverse background** culturally and linguistically
- They have a low tolerance threshold for slow or hard to use sites
- There may be lots of them!

Development Process

- The nature of the **developers**
  - Professional development is by a multidisciplinary team
    - programmers, graphic designers, domain experts…
  - Much development is by amateurs or inexperienced programmers
  - Development can be communal by a geographically distributed group
- The **development environment** consists of
  - a wide variety of technologies, each programmed differently
  - many of the technologies are immature but some are legacy
  - each of the technologies has multiple competing products and upgrades to the site often mean a change of product
- The **development process**
  - follows no accepted internet application development methodologies
  - must be flexible and not rigid
  - Parallel development of components (and even versions) is necessary
  - Agile processes seem valuable here

Web App Evolution

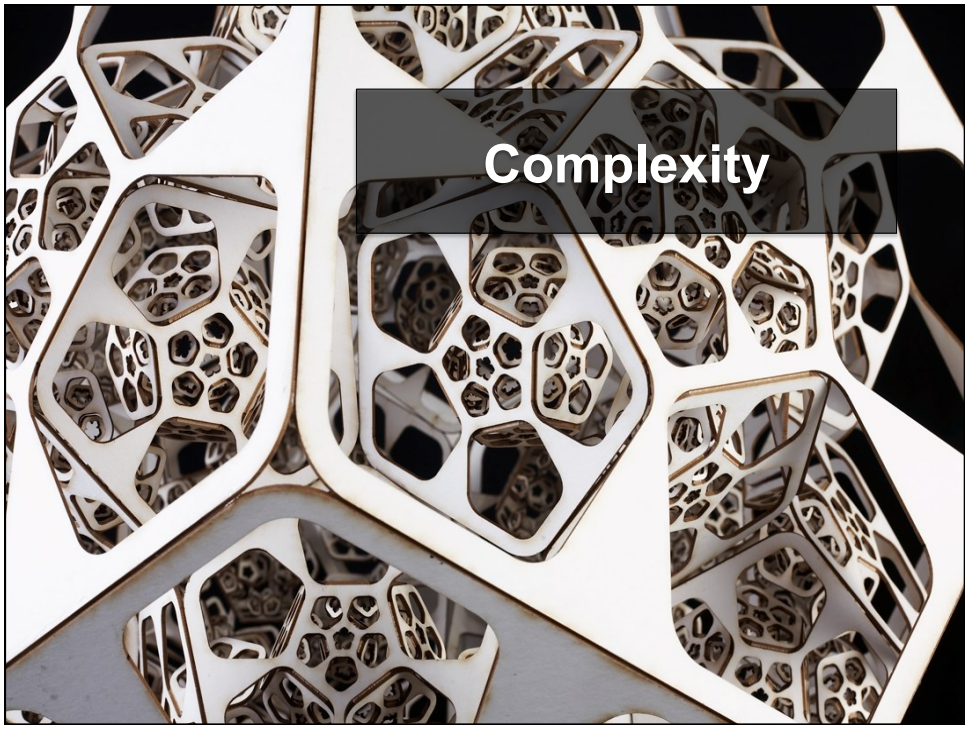


Types of Web App

- **Static**
  - Originally web applications were just collections of hand built HTML pages
  - These required manual update and introduced the possibility of inconsistency
- **Interactive**
  - Forms, selection menus and radio buttons on web pages offer the possibility of selectively chosen pages generated by server-side programs
  - CGI was the first technology but has been superseded by others (ASP, JSP, PHP, Cold Fusion, etc.)
- **Transactional**
  - Forms also offer the capture of data and database storage at the server
  - Although the data management may be separated from the internet server
- **Workflow-based**
  - Support for functionality expressed as a sequence of pages reflecting a business process
  - For instance, booking a flight

Types of Web Apps

- **Portal Oriented**
  - One point of access is given to multiple web sites
    - e.g., Virtual Shopping malls
- **Collaborative**
  - Web sites which permit multiple users to share information management
    - e.g., Wikis such as Wikipedia
- **Social Web**
  - Many sites provide a focal point for communities
    - e.g., Photo sharing sites, Facebook, etc.
- **Mobile and Ubiquitous**
  - "Web" applications increasingly provide access by small, mobile and non-visual devices (i.e., phones) as well as data capture by sensors



DIFFERENT SYSTEM ARCHITECTURES

System Architectures

- Every **component** of a system must be **designed**
- The **architecture** of the **system** shows the **blueprint** or plans of **how each component fits together**.
- Architectural diagrams can be at various levels
  - Data structure / algorithm / object level
  - Component / library level
  - Application level
- Various diagrams are needed
- Here we focus on the application level or the **high level system architecture**

Monolithic Programs

- The main tasks that any application must support:
  - **user interface** management
  - the implementation of **algorithms** – the business logic
  - **information manipulation**
  - **data storage**
- A monolithic program in Java does all of these:
  - user interface with Swing or JavaFX
  - algorithms in methods
  - information manipulation in methods, e.g. sort methods
  - data storage using File I/O

Single Tier Architecture

Tiered Architectures

- The **structure** of **applications** have changed from:
  - **Monolithic** programs as a single unit in which every aspect of the application is coded
  - To **tiered** structures in which different aspects are separated into different levels
- The **advantage** of the **tiered architectures** are
  - Each tier can be coded **separately** without one programmer having to deal with everything
  - The different tiers can be **distributed over the network** leading to increased efficiency
- But the **tiers** must **interact effectively**
  - There must be well defined interface between adjacent tiers
  - Internet protocols and database connection software do this well

TWO TIER ARCHITECTURES

Data Management

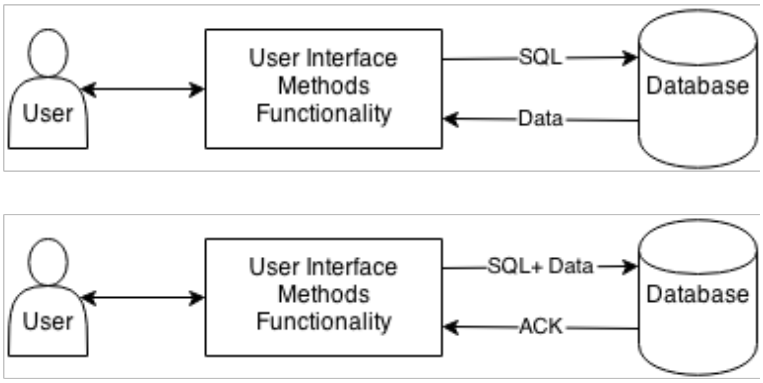
- Data and Information Management is common to many applications
  - Handling and dealing with large amounts of data is required by many applications
  - Access to a shared repository of data is highly beneficial to facilitate information flows between actors
  - Storing, retrieving, modifying, securing is common to many application
  - Database Systems and Information Retrieval Systems provide ways to manage this data and information (efficiently)

Data Management

- It is **difficult/time-consuming** to write the code which accesses large amounts of data efficiently
- **Solution: Separate the concerns**
  - Let a **Database System** handle the data management
  - And then use a **Client application** to interact with the database:
    - The client acts like a database user sending in queries and updates and making use of the result
    - Simple way to do this by coding **SQL statements** as strings inside the program

Client-Server Architecture

Fat Client: This then is the standard architecture for getting an application (e.g. Java-based) to work on top of a database.



Two Tier Issues

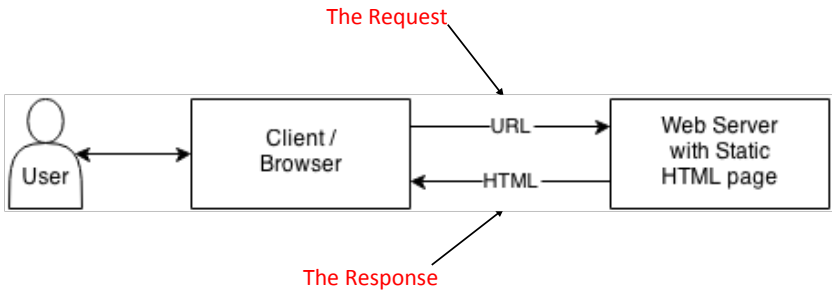
- Applications with a **two-tiered architecture** distinguish only:
  - **client** processes – control the application logic and the user interface
  - **server** processes – supply resources, such as data, to the clients
  - however this:
    - Puts a lot of the load on the client
    - Might tie the client software to the database software so changes in each affect the other

Static Web Application

- Static web sites consist of a set of hyper-linked HTML files
- Each HTML file describes one page of the web site
- Each page includes one or more links to other pages
- If a user clicks a link or enters a URL then a request is sent to the web server identifying the next page to load

Two Tier Architecture

Thin Client: This is the standard architecture for serving up static content on the web.

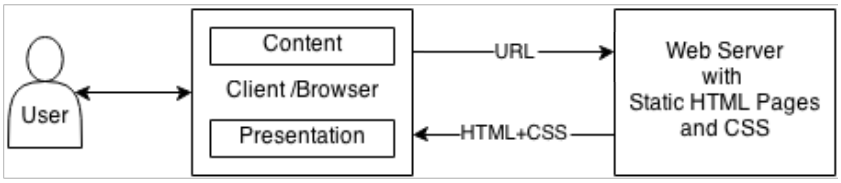


Style and Substance

- Initially, the HTML files contained all the information on how to render the page
  - The text/data (html) and the style (css)
- With lots of pages there is lots of overhead in creating and maintaining the site
- **Solution: Separate the concerns**
  - Use external Cascading Style Sheets which act upon the different elements in the html
  - The web server/application returns an html file and a css file
  - i.e. one to hold the Content and one to represent the Style

Two Tier with Layers

- **Extended Thin Client:** Layers within the client tier handle the content and look/feel



- The **content layer** represents/stores the data
- The **presentation layer** renders the data

Repetition of Structure

- On a site, many of the pages have a similar structure and a lot of repeated content
  - i.e. headers, footers, navigation, etc
- This leads to a maintenance nightmare
  - (or a job for life)
- What should we do when we encounter such a situation?

Repetition of Structure

- **Solution: use a template and decompose repeated elements, i.e., separate the concerns**
  - It is better to have a program which generates pages by merging a template with the specific page data
  - The appropriate data depends on the URL and the parameters in the URL
  - This enables the provision of dynamic content
    - but requires a more sophisticated architecture

Look for Commonalities

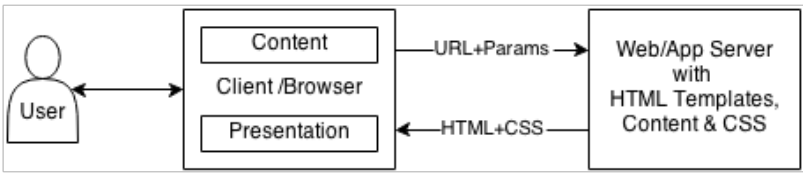
- **Anywhere, anytime, anyplace you see repeated code, look to refactor**
  - Through methods, objects, templates, applications
- **Train of thought**
  - **Extract** the common parts
  - **Parameterise** the parts that change
  - Create a **template**
  - Write a program to use the template given a set of parameters

Handling User Interaction

- Clicking links only provides the link information
  - Forms take input but what happens when you hit the submit button?
  - This enables the provision of Interaction
  - But this requires the server side program to be able to deal with the data from the form
- HTTP provides methods for sending the data entered by the user to the program
  - GET and POST

Two Tier with Layers & Servers

- **Extended Thin Client and Extended Server:** Layers within the client tier handle the content and look/feel. The server now houses a web server and an application server.



- **Web Server** handles incoming requests and sends outgoing responses, routing them to/from the correct application server
- **Application Server** is typically a script that dynamically generates a response given a request