A Few Things

Design

Reusing this material

Looking at Design

Motivation

- Is there a ‘Software Crisis’?
- Many software projects are unsatisfactory
  - lots fail to meet their design goals
  - lots exceed budget or time constraints significantly
  - some are total disasters and are abandoned at huge cost
  - see Computer Weekly for regular examples of software disasters
    - often paid for by the taxpayer!
    - e.g. air traffic control, health software, passport office
- Many reasons for software project failure
  - but good software design is a critical weapon against such problems

Design Goals

- Functional goals
  - ‘what it does’
  - e.g. the item must transport at least one person
  - e.g. the item must allow someone to stay warm in winter
- Performance goals
  - ‘how well it does it’
  - e.g. the item must have a top speed of at least 30 mph
  - e.g. the item must not be heavier than 0.25 kg
- A ‘good’ final item is one which satisfies the design goals

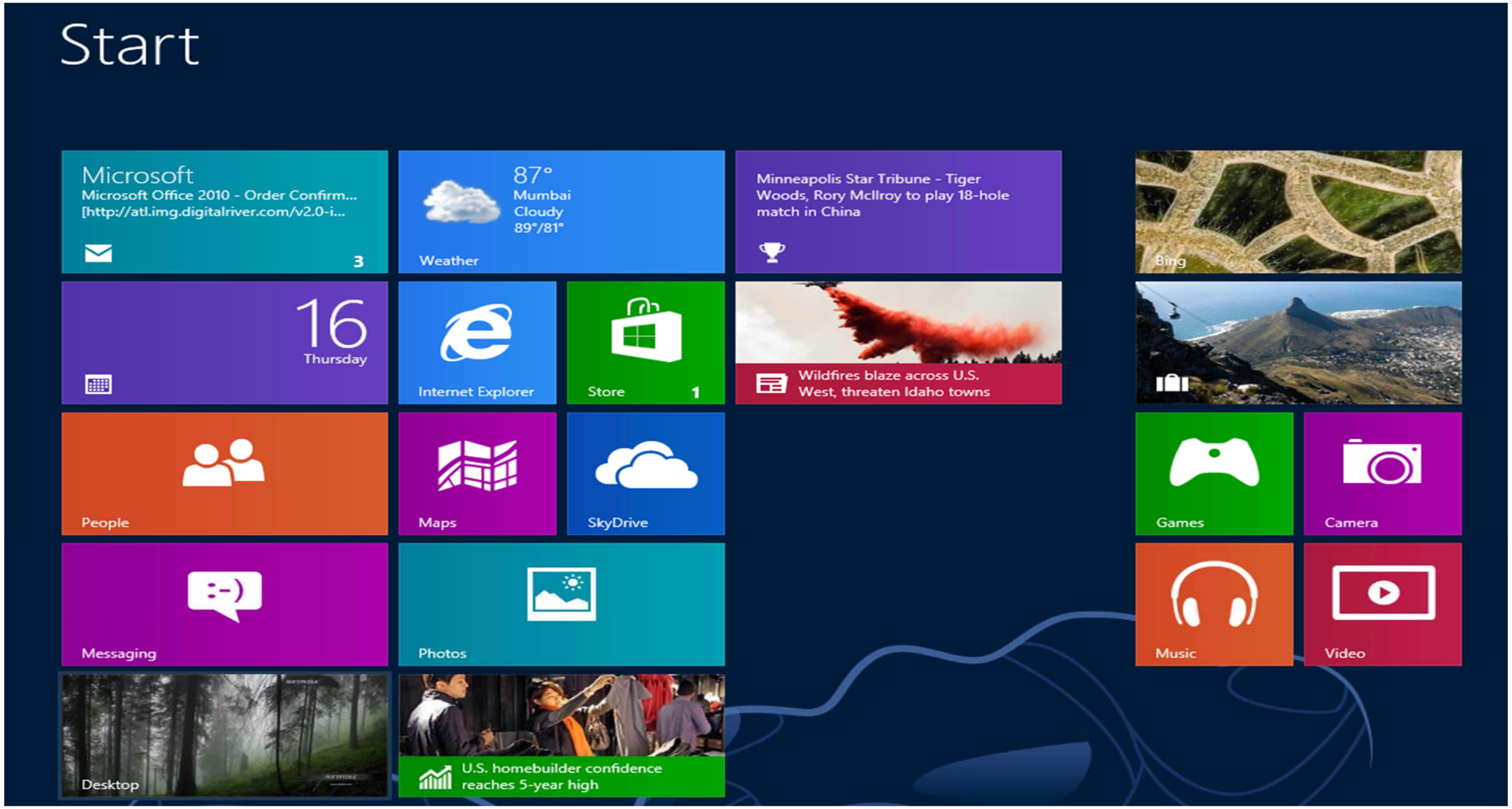
The Big 3 Design Criteria

- 1. Detail
  - how approximate is the design?
- 2. Intersection
  - how much common ground is there between the design and a good final item?
- 3. Merit
  - how many desirable properties does the design have?

Peelers

Picture by scoutjacobus – Creative Commons - Attribution-NonCommercial 2.0 Generic

Windows OS



Microsoft
Microsoft Office 2010 - Order Confirm...
[http://atl.img.digitalriver.com/v2.0-i...

87°
Mumbai
Cloudy
89°/81°

Minneapolis Star Tribune - Tiger Woods, Rory McIlroy to play 18-hole match in China

Wildfires blaze across U.S. West, threaten Idaho towns

U.S. homebuilder confidence reaches 5-year high

Design Merit

- Pick any 3 items (‘real-world’ or software) which have impressed you in some way
- What desirable properties did they have?

Design Merit

- Now pick any 3 items (‘real-world’ or software) which did *not* impress you
- What didn’t you like about them?

Early Design

- What happens if the last design before coding is poor or non-existent?
  - you' ll be coding without a clear idea of what you' re trying to achieve and why and how
  - you' ll be moving to fiddly detail before getting the basics sorted
    - You need to walk before you run
  - you' ll hit problems continuously, and fixing them will be costly
    - The later a change is, the more expensive
  - everything will take longer and the outcome will be poorer
  - A big reason for failure and overspending

Early Design

- 1. Requirements Capture
  - “what exactly is the problem we’re trying to solve?”
  - analyse the problem and establish the design goals
  - results in a requirements document
- 2. Functionality Design
  - “what’s the solution going to do?”
  - functionality and user interface
  - results in a functional specification document
- 3. System Design
  - “how’s it going to do it?”
  - system architecture and detailed design to some level
  - results in a system design document

Requirements Capture

- “What exactly is the problem we’re trying to solve?”
- Aim to produce a Requirements document including:
  - Problem Statement
  - Functional Goals
    - Basic
    - Secondary
    - Enhancements
  - Performance Goals
  - Non-functional Requirements

Obtaining Information

- Who is your ‘customer’?
  - external organisation
  - internal department
  - funding body
  - research colleague
  - focus on whoever gives the ‘thumbs-up’ at the end

- Use all appropriate means to probe for accurate detailed information about the problem
  - face to face discussions
  - observation of existing system (if any)
  - study of existing documentation (if any)
  - questionnaires

The Unreliable Narrator

- Customers are like unreliable narrators in novels
  - you may get a mixture of truths, half-truths and outright falsehoods!
  - you may get conflicting information
    - particularly when several people have a say
  - information may be withheld (inadvertently or otherwise)
- But if the software solves the wrong problem, the customer will blame *you*!
- So try to untangle the requirements mess as early as possible
  - probe into the dark corners
  - overturn the stones

Cans of Worms

- Retailer: “I want a simple program to print out reports of all my current stock”
  - what's the input data?
  - how's it going to be entered? manually? bar-code swiping?
  - how is the stock data to be stored?
  - what sort of reports do you want? sorted? grouped?
  - how often do you want them generated?
  - what if it takes 5 minutes to generate? is that too long?
  - do you really mean print to a printer or to the screen?
  - what if there are reams and reams of it?

The Underlying Problem

- The customer’s perception of the problem may not reflect the real underlying problem!
  - what the retailer really wanted to know was "Do I have a TX354-2 out the back?"
  - he was going to manually scan through the list of stock until he came to TX354-2 in the part number column
  - the underlying problem was the ability to query a stock database
- Need to understand the underlying business or technical problem that needs to be solved

Task

- You are asked to create a player trade analysis program from scouts, coaches and front office staff in basketball.
- The users want to be able to look at any data for a player including motion data and try to predict how they would fit in with the plans for the current team.
- This has to make use of NBA statistics available to teams and public.
- What questions do you ask?
- What else might you do?
- Who do you look at?

Requirements Summary

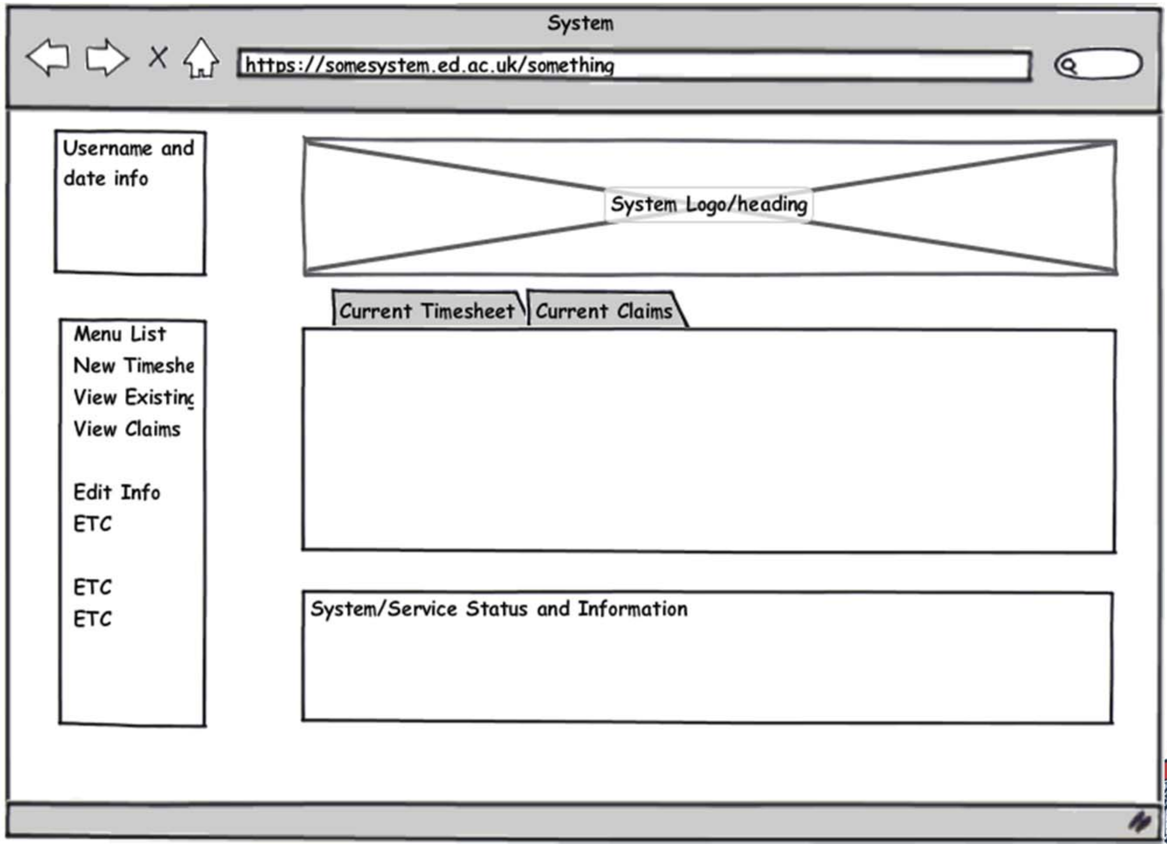
- Gather the information you need
- Resolve conflicts and inconsistencies
- Write a clear and concise Requirements document
- Seek the customer’s approval of the document before proceeding

Functionality Design

- “What’s the solution going to do?”
  - design the *behaviour* of a system which would satisfy the requirements
  - propose a software solution without worrying unduly (yet) about how to build it
- Aim to produce a Functional Specification document including:
  - the main features of the user interface
    - and how the user will interact with the UI to achieve their tasks (use model)
  - the input data
    - and how the system will modify it
  - the main functionality
    - and how it will operate on the data in order to satisfy the requirements

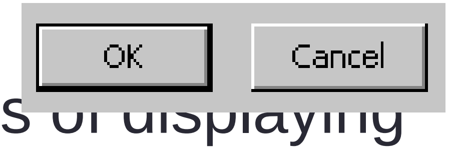
User Interface Design

- Different user interfaces for different applications
- Designing the main features of the UI early on is highly recommended
- UI prototyping
  - Balsamiq
  - Lumzy
  - Pencil
  - Pen and Paper



User Interface Design

- May have to cater for different types of users
  - novice users may want to be hand-held through it
  - expert users usually want to whiz through it with as few mouse clicks as possible
- UI conventions have evolved over the years



- Save your originality for devising intuitive ways of displaying data specific to your application domain

Use Model

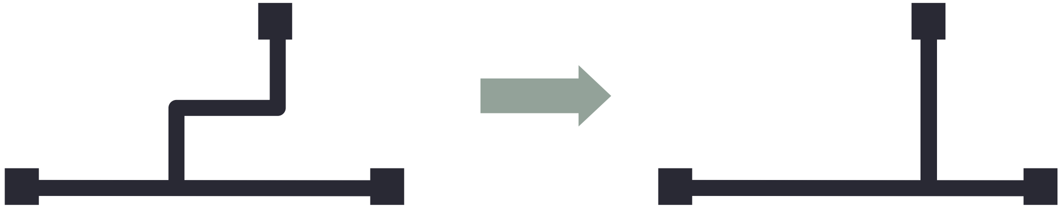
- How will the user accomplish their tasks through the user interface?
    - consider the various ‘flows’ through the software
    - document the sequences of UI interactions necessary
    - show what happens to the user’s data (files) on the way
- Can be very helpful
    - for clarifying your own ideas about how the system will behave
    - for describing to the customer how it will behave

Emphasis on Data

- In general, customers understand their data
  - it’ s important and precious to them
- So communicate with them in terms of things they understand
- Show them:
  - what you think their data is
  - what you’ re going to do to their data
  - what new data you’ ll leave them with at the end of the day
  - what hoops they’ ll have to jump through to get it
- And they’ ll tell you if it’ s a system they want

Main Functions

- What are the main functions of the system?
- For each main function describe the following:
  - its behaviour
  - its input and output data
  - how the data is modified by the function
- Use pictures and examples wherever possible
  - saves lots of typing, aids understanding
  - e.g. a “dog-leg removal” function in a chip layout program



Design Evolution

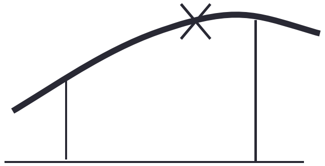
- Designing involves two main things:
  - 1. having ideas
  - 2. realising they’re rubbish (and why they’re rubbish)
- Iterative refinement
  - try not to fall in love with your first idea
  - through perseverance and cunning you may come up with a valuable simplification
- Encourage ‘off the wall’ thinking

Design Evaluation

- Detail
  - have you described the functionality in sufficient detail for it to be meaningful?
  - "and there will be a graphical user interface" is not sufficient detail!
- Intersection
  - will the functionality that you've described satisfy the design goals?
  - does your functionality solve the right problem?
  - is the functionality consistent and coherent?
- Merit
  - does the behaviour you've described have desirable properties?
  - is the system as simple as possible?
  - is it intuitive?

Design Evaluation

- Probing to the next level of detail beyond the level you' re documenting can be very useful
  - helps establish the quality of what you *are* documenting
- E.g. "the interpolation facility will operate on the curve data which is passed in"
  - sounds fine
  - but on probing to the next level of detail you discover that the interpolation facility also needs a point at which to interpolate the curve
  - where' s this point going to come from?
  - oops - inconsistency exposed
  - much cheaper to fix it sooner than later



Functional Design Summary

- Design the behaviour of your solution
  - and prototype the user interface if possible
- Iteratively evolve and improve the design
  - focus on the critical features first
- Try to gain confidence in its quality
  - evaluate it and get someone else to review it
- Write a clear and concise Functional Specification document describing it
- Again seek document approval from the customer

System Design

- “How’s the solution going to work?”
  - how will the documented behaviour be realised in software?
- Aim to produce a System Design document including:
  - system architecture
    - how the system will be composed of smaller components or modules
  - component descriptions
    - responsibilities and interfaces
    - where will the main functions reside?
    - main data structures and algorithms
  - solutions to key technical problems
  - enough detail that moving to code doesn’t seem like a huge step!

Components

- Why has this component been defined?
    - what’s its purpose?
    - ensure the component has clear goals and responsibilities
- Which of the component’s functions will form the interface to the outside world?
- Which of the main functions will reside in this component?

Components

- How will the component’s data be modeled in software?
  - arrays, records, structs, objects?
  - what will they contain?
  - what’s the lifetime of the data?
  - who’s responsible for the creation / destruction of which data?
- What are the main algorithms and how will they be implemented?
  - give pseudo-code if appropriate
  - pseudo-code shouldn’t just be verbose normal code!
  - Pseudo-code should help not hinder

Pseudo-Code Example

- Graphics update algorithm

```
for each open window, w {
   for each of w’s objects, obj {
      if obj has been modified since last redraw then {
         redraw obj
         clear obj’s modified flag
      }
   }
}
```

When to Stop?

- When should you stop documenting the system design and actually start coding?
  - tricky matter of judgement
- Things to ask yourself to see if you’re ready
  - are there any parts of the design I’m particularly nervous about?
  - is my vision of the system the same as that of my co-developers?
  - is there enough design detail for coding to be an orderly guided activity?
  - do I think I’m close enough to the top-right corner of the design cube?
- Often worth going to pseudo-code detail for trickier areas first
  - quicker than writing and compiling real code

Write a bit of pseudocode

- Write a piece of pseudocode or other none code based description of an algorithm
- Choose one that you know for example, binary search, quick sort, how to get the mode of an array or other
- Try not to fall into using programming language

System Design Summary

- Design the architecture of the system
- Design the components and their interactions
- Evolve and improve the design
- Check it relates closely to the Functional Spec
- Write a clear and concise System Design document
- Unlike the Requirements and Functionality documents, this is an internal document
  - for the benefit of the developers when they start coding in earnest
  - customer doesn’t care how it works as long as it does work

Interim Key Points

- Key Points
  - Design is important and ongoing
  - Only as document heavy as needed
  - Constant Evaluation
  - Talk to Clients
  - Do not overcomplicate

What is a System

- Definitions:
  - A set of things working together as parts of a mechanism or an interconnecting network.
  - A set of principles or procedures according to which something is done

- What do this means for us?
  - Its not enough to work inside the boxes and wires
  - Impact on and of the wider world

Examples of Systems

- A front door
- A calculator
- An air-conditioning system for a house
- A word processor
- An operating system
- Government
- The Internet
- Taxes
- Humans

Characteristics of Systems

- Interconnections
- Complexity
- Size
- Procedures
- Redundancy?
- Safety?

A Task

In small groups, take five minutes to do the following:

- Think of a system
- Describe why it is a system?
- What are its characteristics?
- Which are innate to the system?
- Which are outside influences?

Boundaries and Interactions

- A key characteristic of a system is that it will have input and output beyond itself
- A system is not an isolated thing – it needs other things to work
- Where does one system end and another pick up?
    - When the processing becomes the responsibility of another
    - Transference of control
    - Organisation
- What are the interactions of a system?

Interactions

- In System
  - Electronic
  - In Control
- External to System
  - Status
  - Input and Output
  - Interfacing

Interactions

- Consider these systems for what you would think of as interactions
  - World of Warcraft
  - ATM
  - Traffic Monitoring
  - Restaurant
- Discuss one of these in small groups and then summarise the interactions for the rest of the groups

Common High Level Architectures

- layered architectures
- pipe and filter
- shared-data
- Client server
- model-view-controller

Layered Architecture

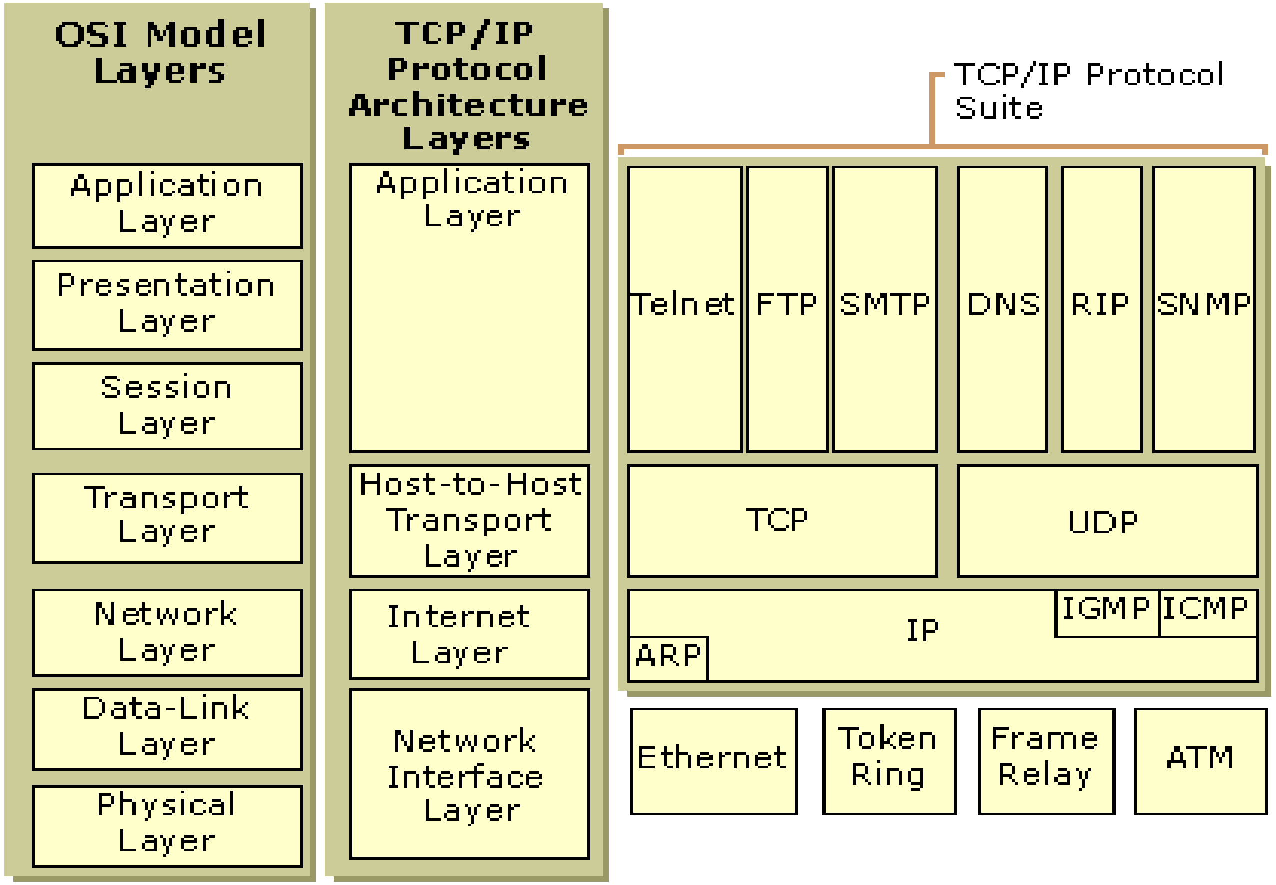
- System Divided into layers/modules
- Layers provide services to other layers
- Can be open or closed in operation
- Common Layered Architecture:

Presentation Layer
Application Layer
Storage Layer

Layers

- Cohesive definition
- Works well with OOP principles
- Useful for decomposing functionality
- Decrease coupling
- Can affect performance
- Debugging can be awkward
- Getting it right is hard

Another Layer Example



Pipe and Filter

- "The Pipes and Filters architectural pattern provides a structure for systems that process a stream of data. Each processing step is encapsulated in a filter component. Data [are] passed through pipes between adjacent filters. Recombining filters allows you to build families of related filters."

Pattern-Oriented Software Architecture: A System of Patterns,, Wiley, 1996.

Pipes and Filters

- Intermediate files unnecessary, but possible
- Flexibility by filter exchange.
- Flexibility by recombination
- Reuse of filter elements.
- Rapid prototyping of pipelines.
- Efficiency by parallel processing.
- *Sharing state information is expensive or inflexible.*
- *Efficiency gain by parallel processing is often an illusion.*
- *Data transformation overhead*
- *Error handling.*

Shared Data

- Characterised by one or more shared-data stores used by one or more
- Shared-data accessors (i) store, delete, and modify data in shared-data stores and (ii) communicate through shared-data stores only
- Shared-data stores have no knowledge of accessors

Pros and Cons

Pros:

- Accessors, which only communicate through stores, can be independently changed, replaced, added, or deleted.
- The independence of accessors increases program robustness and fault tolerance
- Placing all data in the store makes it easier to secure data and to ensure its quality

Cons:

- Forcing all communication through the store may degrade performance
- If the store fails, the entire program is crippled; this may be a source of unreliability

Client Server

- A server subsystem instance provides services to instances of other subsystem instances (the clients), which are responsible for interacting with the user or other systems
- Client and Server communicate via a defined set of service interfaces
- Client and Server do not have to be implemented in same fashion
- The communication is handled by a protocol
- Service definition is key.

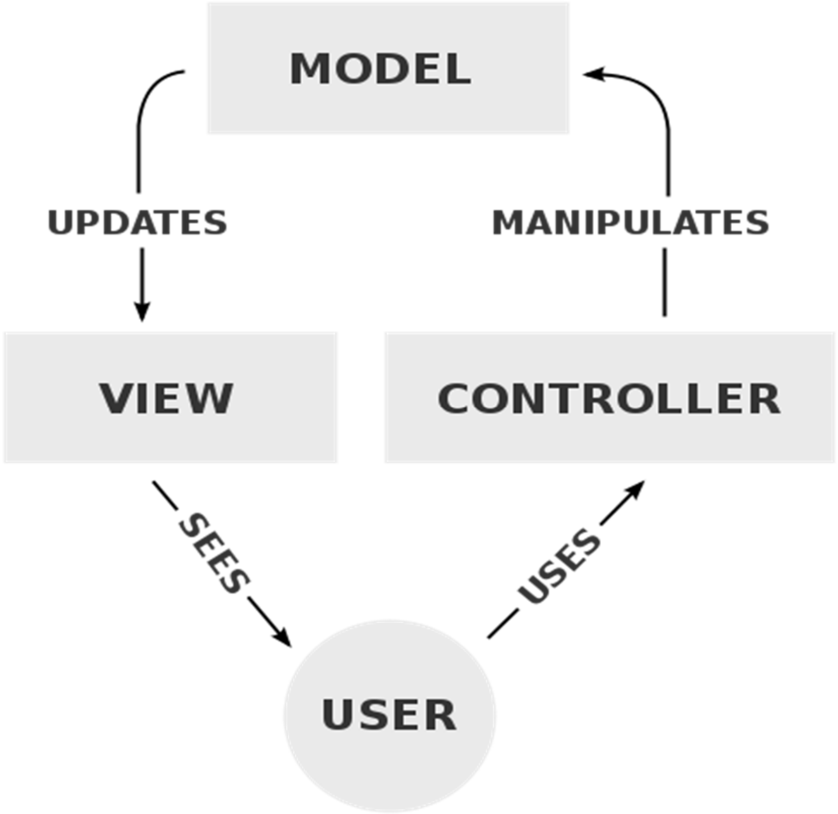
Client-Broker-Server

- Extension to the Client Server
- A Broker sits between the Client and the Server
- The Client only knows about the Broker
- The Server talks via the Broker
- Can be used for load balancing
- Additional Security
- Performance can be affected

Peer to Peer

- An extension to Client Server
- Clients can be Servers and vice versa
- Increases possibility of deadlocks
- complicates the control flow
- Allows greater flexibility

MVC



Outside the Computer – Still a System

- Systems are not all in one place
- Need to take into account:
  - Common Practice
  - Legal
  - Reporting
  - Human Factors
  - Non-computable elements
- Even if you don’t write/develop/suggest something, it can still be in the system

Where do you stop?

- A difficult question
  - Too soon – system is incomplete and useless
  - Too late – system is over-engineered, too expensive, corrupts work practices

- Stop once you get to where the client or user is telling you to stop
- Stop once you get to the point where you are making business decisions

A Task

- Create a high level design for one of the following:
  - Fantasy Sports Provider
  - House Automation
  - E-Learning Platform
  - Museum or Art Gallery Archive and Access
  - Online Shop (Amazon or similar)

- Do this in groups and be prepared to summarise for everyone your designs.