
# Topic 1: Intro to Software Architecture

ICT373: Software Architectures

# Overview

- Introduction to simple software architectures
  - Design and Software Development
  - Software Architectures
  - Pipe and Filter Architecture
- Design in software development
- What is architecture?
- The software process models
- What is software architecture?

Reference: Garlan and Shaw, "An Introduction to Software Architecture"
http://www.cs.cmu.edu/afs/cs/project/vit/ftp/pdf/intro_softarch.pdf

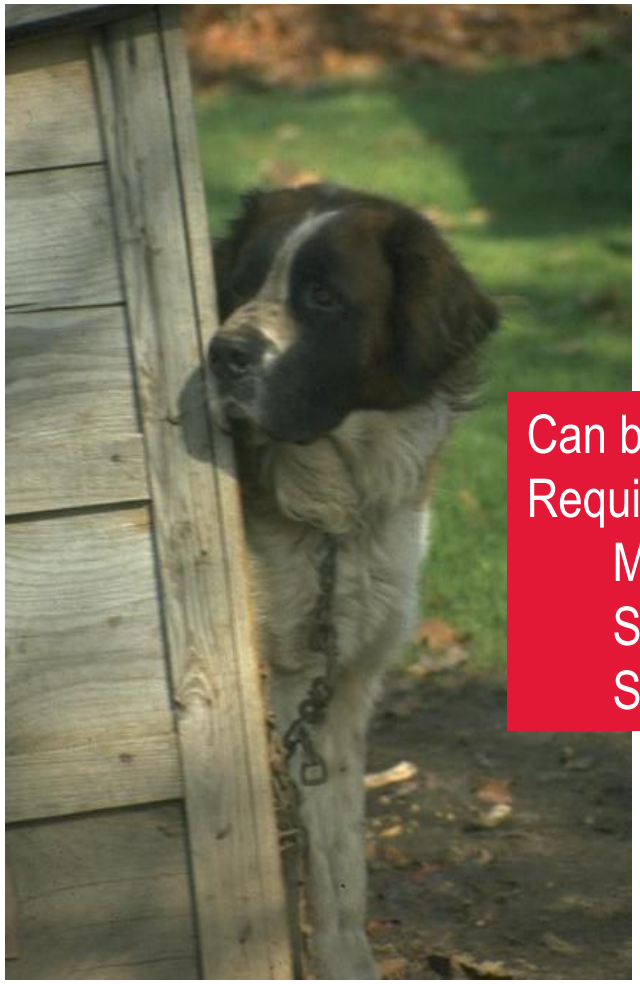
Murdoch UNIVERSITY

# Learning objectives

- Understand the software development lifecycle (SDLC).
  - Understand the software process models.
  - Distinguish between predictive lifecycle models and adaptive lifecycle models.
- Role of **design** in software development.
  - Factors considered in the design phase.
- Define software architecture and list some architectural issues.
- Learn the basics about Pipe and Filter architecture.

Murdoch UNIVERSITY

# Architecting a dog house



Can be built by one person
Requires
- Minimal modeling
- Simple process
- Simple tools

Murdoch UNIVERSITY

# Architecting a house

Built most efficiently and timely by a team

Requires

- Modeling
- Well-defined process
- Power tools

Murdoch UNIVERSITY

# Architecting a high rise

Murdoch UNIVERSITY

# Differences

- Scale
- Process
- Cost
- Schedule
- Skills and development teams
- Materials and technologies
- Stakeholders
- Risks

Murdoch UNIVERSITY

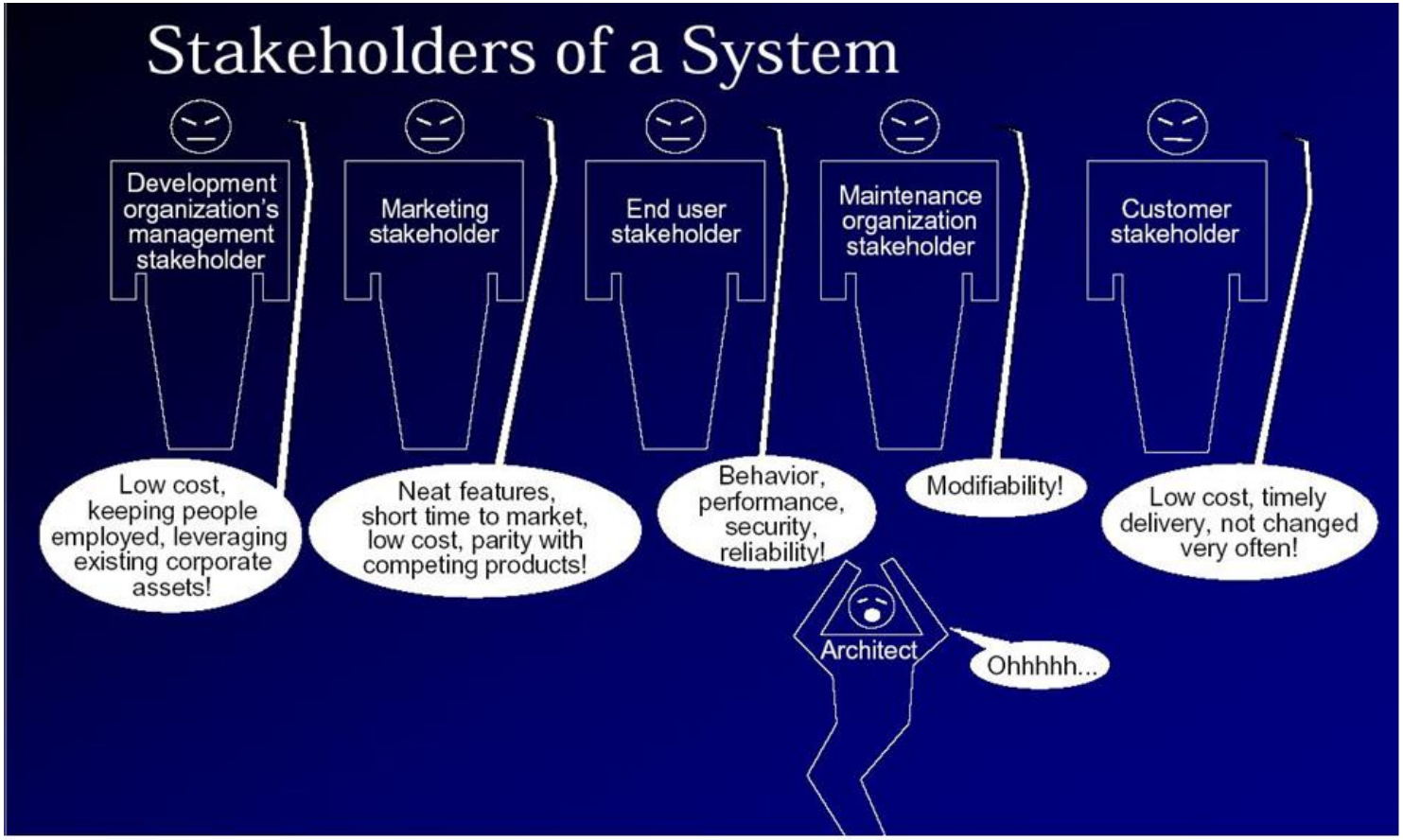
# Definition: Software Architectures

IEEE 1471-2000

> Software architecture is the **fundamental organization** of a system, embodied in its **components**, their **relationships** to each other and the environment, and the **principles** governing its design and evolution

Murdoch UNIVERSITY

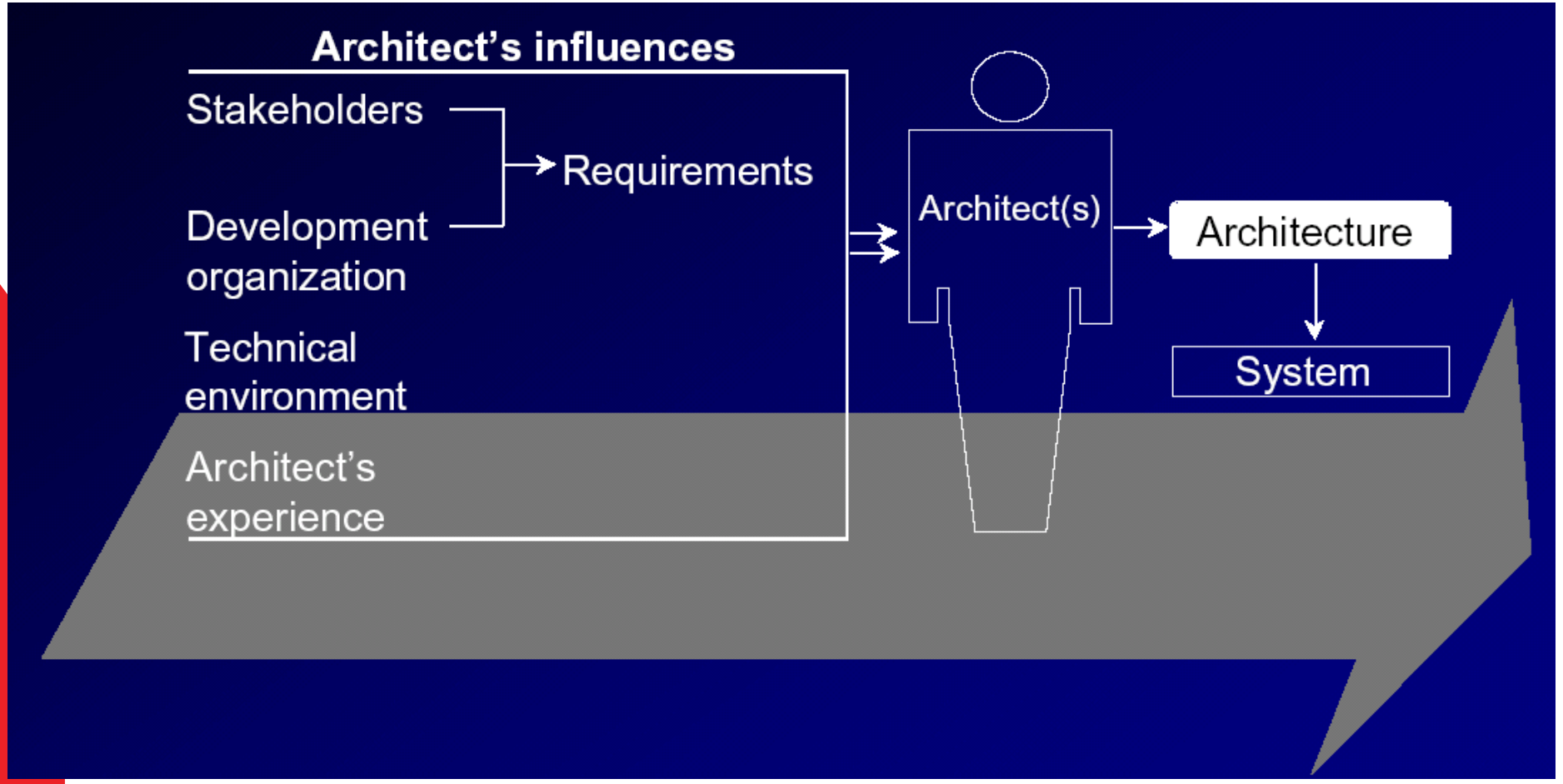
# Who influences SA?




http://www.ece.ubc.ca/~matei/EECE417/BASS/ch01lev1sec1.html#ch01fig02

Murdoch UNIVERSITY

# Influences on the Architect

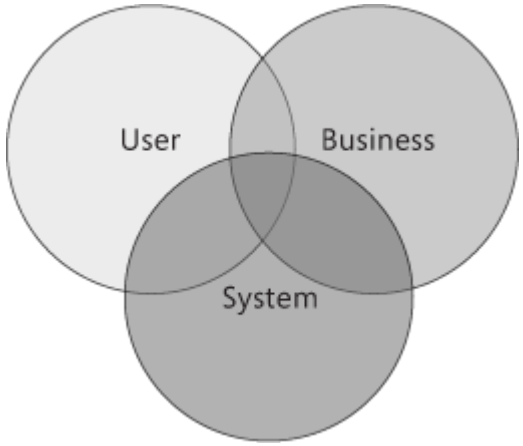



http://www.ece.ubc.ca/~matei/EECE417/BASS/ch01lev1sec1.html#ch01fig02

Murdoch UNIVERSITY

# Influences on the Architect

- Systems should be designed with consideration for the user, the system (the IT infrastructure), and the business goals.




Murdoch UNIVERSITY

# Why is architecture important?

- Handling complexity
- Communication among stakeholders
- Early Design Decisions
- SA is a transferable, reusable model.

Murdoch UNIVERSITY

# Complexity of Software Systems

- Software systems are some of the most complex systems that humans have developed.
- Is this complexity accidental or inherent?
- Most people think it is **inherent** because of the combination of its elements which are individually complex:
  - complexity of problem domain - aircraft control software, phone switching networks, defence systems, etc. are examples of complex domains with many, and conflicting, requirements
  - complexity of development process - the size of such systems is beyond a single human's comprehension, thus the work must be done in teams.

# Complexity of Software Systems

- flexibility possible in using software - there are no building codes and standards for software as there are for the building industry and little repositories of reusable software.
  - but this is changing as the discipline matures.
- complexity of characterising behaviour - an executing software system moves from one discrete state to another. Each state usually contains a very large number of variables.
  - there is no software physics to predict accurately and reliably the stability or otherwise of software systems.

# Managing Complexity

**Structure**

- Master complexity of software systems by imposing structure on them.
- Decomposition - breaking a system into a set of smaller, relatively independent parts that are hopefully easier to understand.
    - This process is repeated until the components are each small enough to understand completely.

**Cognitive limits**

- Miller's discovery: that the human brain can only handle about 7 (±2) items at any one time.

# Managing Complexity

**Decomposition**

- The classical approach to software decomposition is **top-down** structured design.
- An alternative is the **object-oriented** approach, where we identify the important abstractions of the problem and model them as objects.

**Abstraction and Hierarchy**

- Abstraction: involves ignoring everything that is not absolutely essential to the problem. Art is in choosing the best abstractions to use and what details have to be kept and modelled.
- Hierarchy: organise abstractions into hierarchies - that impose a certain logical structure which simplifies the problem.

# Managing Complexity

- We can apply *hierarchy* to the **software development process** itself,
- we can impose a structure on the development of a software product by identifying the components of the development process and dealing with them independently - this led to the concept of **SDLC** or **software process models** (such as the Waterfall model, the Spiral model, Prototyping model, Incremental development).

Murdoch
UNIVERSITY

# Design

# “Design” in Software Development

- This unit is about how to go about building a software system.
- We are interested in medium to large complex systems of software
- We are mainly interested in the **architecture** or **overall form of design**.

Murdoch UNIVERSITY

# What is design? General concepts

- For a consumer product, we can identify three important stages:
- Concept formation
  - The need for the product is identified and its functions are defined.
- Design
  - Methods of achieving the required functions are invented.
  - The product is a set of parts that work together to achieve the functions.
  - Designing the parts and the way they interact is the essence of the design process.
- Manufacture
  - The design is turned into a physical product.

Murdoch UNIVERSITY

# Definition of DESIGN

- Conscious, methodical and planned approach for solving problems to meet certain goals.
- Also the more 'artistic' use that emphasises the aesthetic nature - fashion design etc.
- The Collins Concise Dictionary gives the definition as:
- **Design** *vb.*
  - to work out the structure or form of (something) as by making a sketch or plans.
  - to plan and make (something) artistically or skilfully.
- (tr.) to invent ~n
  - a plan or preliminary drawing.
  - the arrangement, elements, or features of an artistic or decorative creation.
  - see http://www.thefreedictionary.com/design

Murdoch UNIVERSITY

# Software Architecture Vs Design

**Architecture** is the bigger picture:

- the choice of frameworks, languages, scope, goals, and high-level methodologies
- The architecture of a system is its 'skeleton'.
- It is the highest level of abstraction of a system.

**Design** is the smaller picture:

- the plan for how code will be organized;
- how the contracts between different parts of the system will look;
- the ongoing *implementation* of the project's methodologies and goals.
- Specification are written during this stage.

- Software architecture is more about the design of the entire system, while software design emphasizes on module / component / class level.
- An architecture determines how components interact, not how they are implemented.

Murdoch UNIVERSITY

# What is design? Software development

- Process that occurs between analysis and implementation

**What factors are important?**

- structure of the eventual code of the solution (Organisation)
- activities that are to occur (Function) and their ordering (Control)
- how activities fit into the structure (Packaging)
- organisation of data: data structures, files, databases,... (Data)

# What is design? Software development

**Constraints**

- timing/prerequisites (activities may have to complete within a certain time, or follow each other within a certain time)
- capacity - amount of data, data transfer, storage mechanisms

Design must provide solution to all aspects of the specification and make sure that the eventual implementation meets its constraints!

**Design is an ESSENTIAL skill for Software Engineers, Programmers, etc.**

Murdoch UNIVERSITY

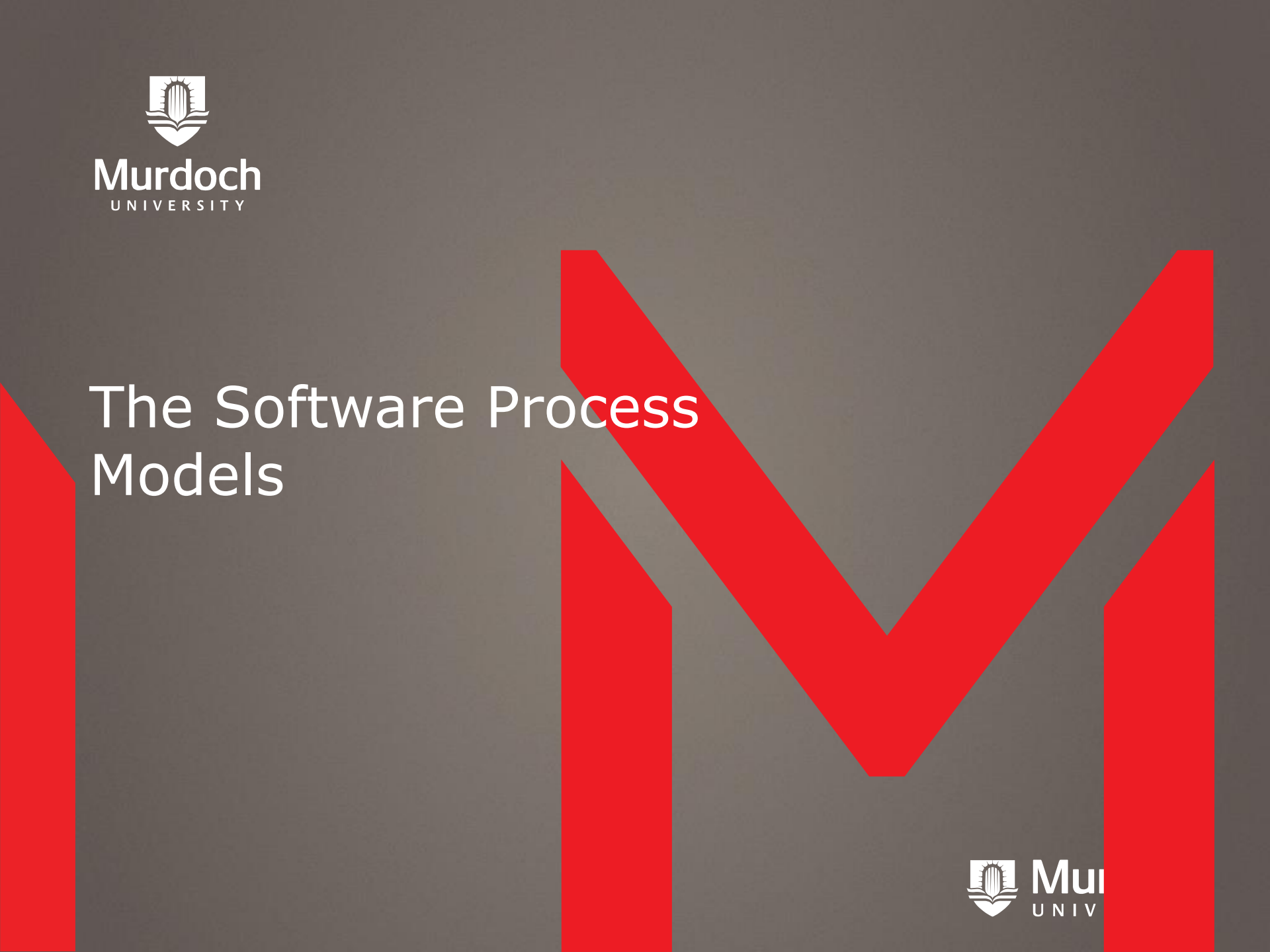
# More on Design and Architecture

When we have to design a system involving many components then the arrangements of algorithms and data structures are no longer the main design problem. Instead, the organization of the overall system – the *software architecture* – must be addressed first.

**[From Garlan and Shaw]…** Architectural issues include

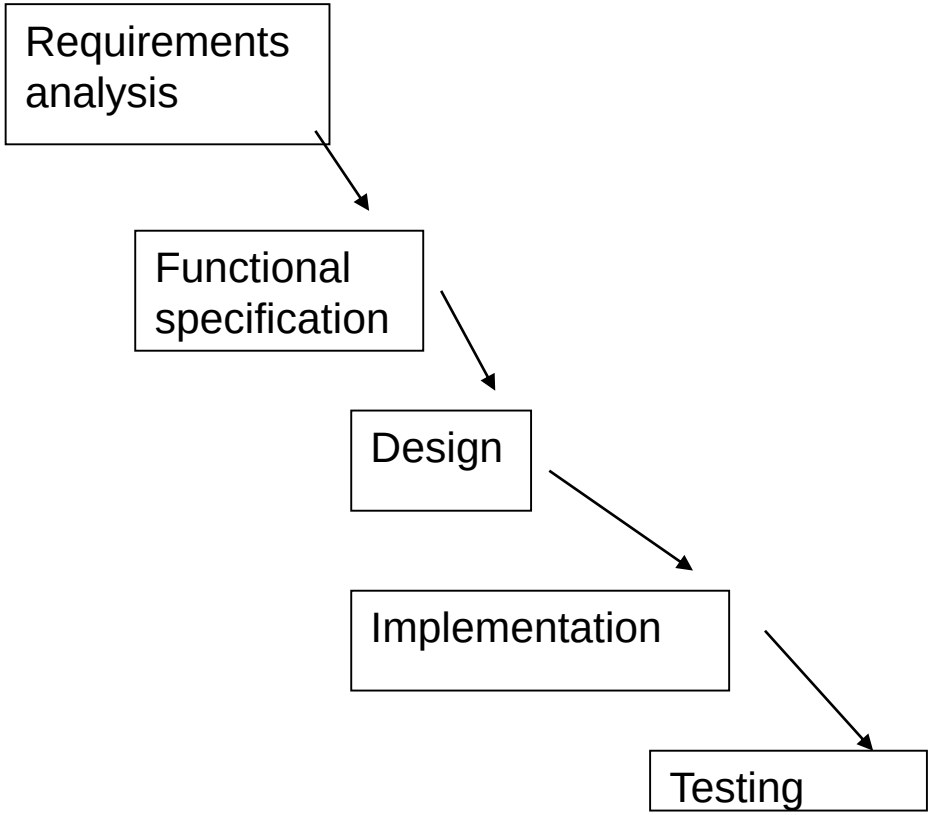
- gross organization and global control structure
- protocols for communication, synchronization and data access
- assignment of functionality to design elements
- physical distribution
- composition of design elements
- scaling and performance
- selection among design alternatives

Murdoch UNIVERSITY


Murdoch
UNIVERSITY
The Software Process
Models
Mu
UNIV

# The classic *Waterfall* lifecycle model

- No phase is considered complete until all documents have been delivered and accepted.
- You can add 'deployment and maintenance'




Murdoch UNIVERSITY

# Prototyping Model

- Involves building a model (a working replica) of the system that is to be constructed but the model is missing some of the essential functionality.
- It requires heavy user involvement to clarify user requirements for operational software.
- Developers can throw away or keep prototypes depending on the project.
- The role of prototyping is to reduce the risk of producing a system which does not meet user requirements or is too expensive or unreliable.

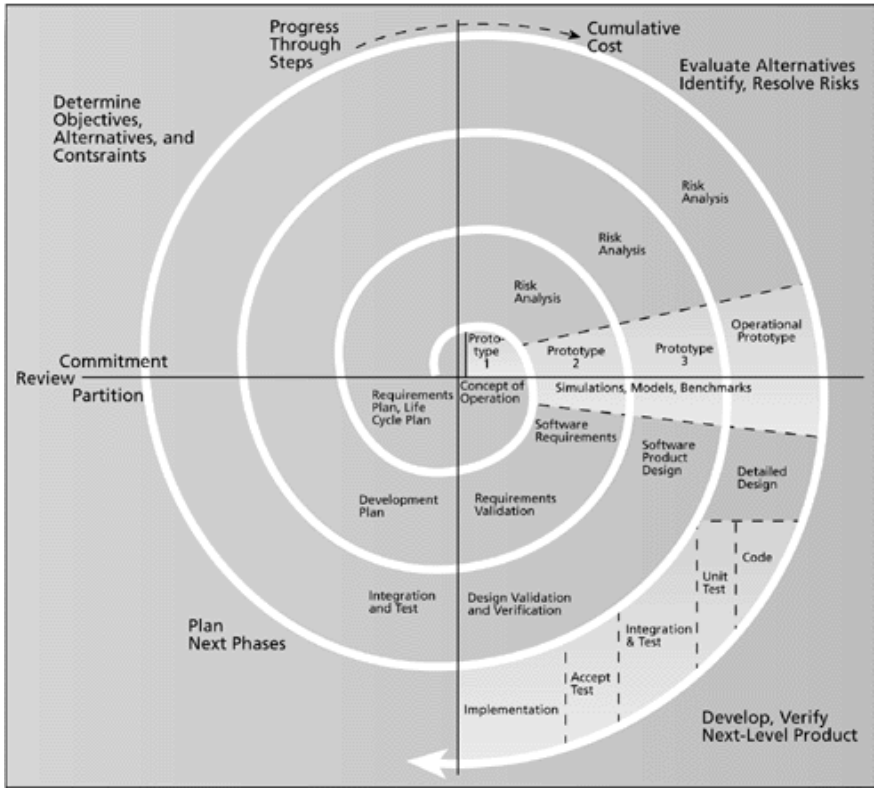
# Prototyping Model

Three common prototyping approaches:

- the Human-Computer Interaction (HCI) prototype
- a working, minimal prototype that executes a subset of the functions
- a complete working system that is unfinished in terms of either final implementation platform (hardware and/or language) or algorithm (not yet optimised for efficiency of execution or storage)

# Spiral Model of Software Development (Boehm, 1988)

- The spiral model is a risk-driven approach to software development
- combines the best features of both classical lifecycles and prototyping
- works well where risk, technical challenges, rapid market changes, or other factors might warrant project cancellation




Murdoch UNIVERSITY

# Incremental build model

- Incremental model provides for progressive development of operational software.
  - First, develop and deliver a system with a subset of the required final functionality.
  - then extend in the next system, and so on until the final system is delivered.
- It involves prioritizing requirements of the system and then implementing them in groups.
- It produces operational system more quickly (than the waterfall model) by emphasizing a building-block approach that helps control the impact of changing requirements (including technology).
- It works well on large as well as on small projects.

# The Rapid Application Development (RAD) model

- It is a type of incremental model.
- In RAD model the components or functions are developed in parallel as if they were mini projects.
- The developments are time boxed, delivered and then assembled into a working prototype.
- RAD is ideal for developing software using 4GL and DBMS technology.
- The model was introduced at IBM by James Martin
- "RAD (Rapid Application Development) refers to a development life-cycle designed *to give much faster development and higher-quality results than those achieved with the traditional life-cycle*"

  - James Martin - "Rapid Application Development" – Macmillan Publishing

Murdoch UNIVERSITY

# RAD Models

- This model involves the use of CASE tools and code generators to facilitate rapid prototyping and code generation.
  - 4GLs and development environments in which design tools, GUI generators, report generators etc are used to develop large systems.

Examples of such tools include

- Powerbuilder,
- Oracle Design/2000,
- Visual C++,
- Visual Basic.

**Active user involvement is essential as developers work with an evolving prototype.**

Murdoch UNIVERSITY

# Predictive models Vs Adaptive models


- **Predictive models (Process-oriented)**:
- The scope of the software project can be clearly articulated and the schedule and cost can be predicted.
  - The models we discussed are examples of predictive models.
- **Adaptive models (People-oriented)**:
- When requirements cannot be expressed early in the lifecycle.
  - Requirements are developed using an iterative approach, and development is risk-driven and change tolerant (responds to change over following a plan).
  - Emphasis is on producing working software than comprehensive documentation.
  - **Agile Software development**, examples of which include Extreme Programming (XP) and Scrum.

Murdoch UNIVERSITY

# Predictive models Vs Adaptive models

**Extreme Programming (XP):**

- Programmers program in pairs and must write the tests for their own code.
    - XP teams include programmers, architects, managers, and users. These people must work together daily throughout the project. It focuses on whole team approach and collective code ownership.
    - Software development is test driven.

**Scrum:** (holistic approach)

- Based on rugby's scrums, to get out-of-play ball back into play.
    - Repetitions of iterative development
    - Daily goals
    - Works best for object-oriented software development and requires strong leadership to coordinate the work.
    - Has a project management flavour compared to the programming flavour in XP.

# Predictive models Vs Adaptive models

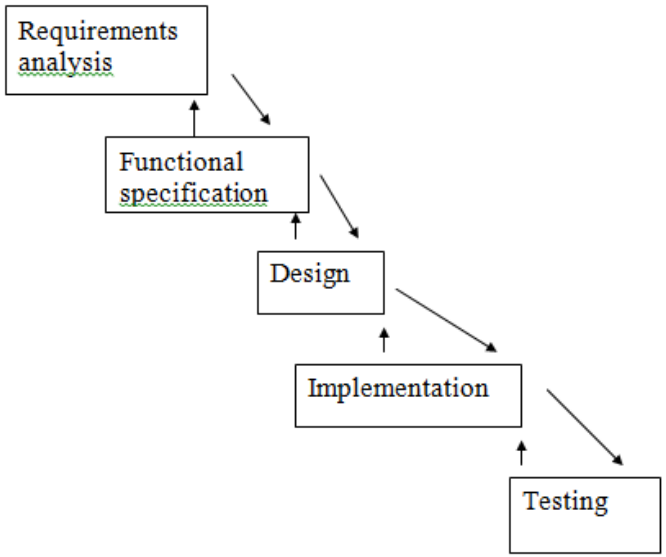
When to adopt agile methods?

- Volatile and uncertain environments
- Motivated people, adaptable people
- Small to medium size teams
- Flexible, involved client.

Murdoch UNIVERSITY

# The Role of Design in SDLC

- the process of design is an important stage of the SDLC, for example, in the *waterfall model* the phases in the process do not actually proceed in a linear order but that there is back-flow.
- However, the idealised order is appropriate for structuring an account of the development process.




Murdoch UNIVERSITY

# The Role of Design in SDLC

The phases are: (recall from waterfall …)

- requirements analysis: determine what is needed from the system
- specification: precise detailed statement of what the system should do
- design: describe how the system will do it
- implementation: code the design
- testing: show that it works as required
- installation (including user training)
- maintenance

# The Role of Design in SDLC

Exactly what should be done during the design phase depends on:

- the size and complexity of the system
- the size and arrangement of the design and implementation teams
- the design method
- any requirements for documentation
- any management and institutional requirements

# The Role of Design in SDLC

The design phase will produce a plan for the product (the final system):

- the static structure of the system (e.g., hierarchies of sub-programs)
- data objects
- algorithms
- packaging (into compilation units)
- interaction between components
- A description of the planned design may thus have several views. There are several formalized design representations such as:
  - entity-relationship diagrams (ERDs),
  - data-flow diagrams (DFDs),
  - state transition diagrams (STDs) and
  - UML diagrams.

# The Role of Design in SDLC

The design process might include producing a plan for the process of developing the product:

- the arrangement of personnel and equipment
- the order of development of parts of the system
- the order of assembly of the parts (integration)
- any requirements for recording the decisions taken.

There are several recognized methods or strategies for the design process such as top-down, re-use, the SSADM or O-O design methods.

# Software Architecture: Designers' attributes

- The designer needs to be competent at architectural design because:
  - they need to recognize common paradigms so that new systems can be built as variations on old ones
  - they need to get the right architecture
  - they need to know the details of an architecture to choose the right design methods
  - they need to be able to analyse, describe and represent the high-level properties of  a complex system.

# Software Architectures: examples

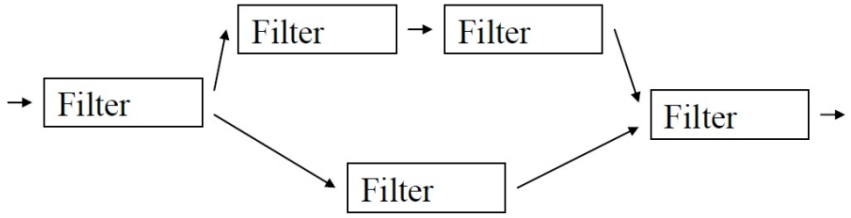
There is not one choice of architecture or even only one approach to classifying architectures.

Examples of architectures (architectural styles) include:

- **pipes and filters**
- **data abstraction and O-O organization**
- event based systems
- layered systems
- repositories
- virtual machines
- distributed processes including client-server architectures
- state transition systems
- **client-server architectures**

Murdoch UNIVERSITY

# Pipe and Filter Architecture




**Filter:**

- stream(s) of data in
- stream(s) of data out
- usually local transformation of input(s)
- incremental computation: output commences before input is consumed
- All data do not need to be processed for next filter to start working.

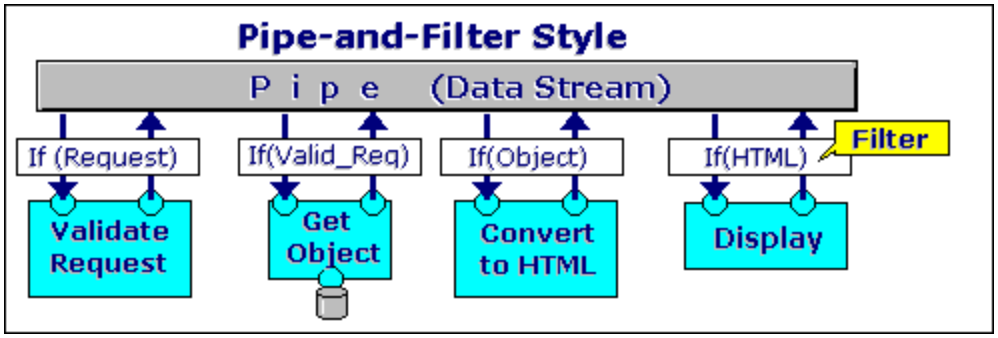
**Pipe:**

- connector of filters
- transfers output of one filter to inputs of another (without change)
- may be bounded in capacity
- may be type restricted

Murdoch UNIVERSITY

# Pipe and Filter Architecture


- **Variants/specializations:**
- restricted topology: i.e. arrangement of components
  - e.g., "pipelines" - linear sequences of filters




- special case "batch sequential style"

  each filter consumes whole input before producing output

Murdoch UNIVERSITY

# Pipe and Filter Architecture

**Important properties:**

- filters are independent entities (no sharing of state)
- identities of filters not known to each other
- correctness of overall output of a pipe and filter network should not depend on the order of processing in individual filters.

**Advantages:**

- understanding: the overall input/output function of a system is a simple composition of the behaviours of the individual filters (and the way they are put together)
- reuse: just pipe the filters together for all sorts of different applications
- maintenance and enhancement: easy to add new filters or replace old ones with better versions
- analysis: amenable to rigorous formal analysis of certain properties such as throughput and deadlock
- supports concurrency

# Pipe and Filter Architecture

**Disadvantages:**

- Designer may be forced into batch processing design,
- No cooperation between filters, hard to handle reactive and interactive problems and hard to maintain correspondence between several similar streams.
- Filters often force the data to be represented in the lowest common denominator, typically byte or character streams. This means that if processing must be based on information tokens (words, lines, records), every filter may introduce overhead for parsing and unparsing the data stream.
- If a filter cannot produce any output until it has received all of its input, the filter will require a buffer of unlimited size. If fixed size buffers are used, the system could deadlock. A sort filter has this problem.

# Pipe and Filter Architecture: examples

- compilers (lexical analysis, parsing, semantic analysis, code generation)
- signal processing
- functional programming
- distributed systems

UNIX operating system style:

- This is strongly based on the pipe and filter architecture.
- Programs such as cat, sort, grep act as filters.
- The UNIX pipe command '|' is a pipe.

```
cat fred.txt mary.txt | sort | grep ize
```

Murdoch UNIVERSITY

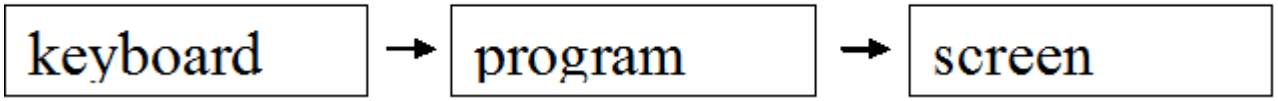
# Pipes and Filters in MS-DOS/UNIX

- I/O redirection in MS-DOS
- Pipes in MS-DOS
- Filters in MS-DOS
- GREP *regular expressions*

Murdoch
UNIVERSITY

# Examples

# Pipes and Filters in MS-DOS




**1. I/O redirection in MS-DOS**

- Redirect output to a file using '>', e.g.,

  dir > fred.txt

will create (or overwrite) the file 'fred.txt' to make it contain the output of the 'dir' program (the listing of the current directory).

- Appending to a file can be done by '>>', e.g.,

  dir >> fred.txt

- Redirect input to come from a file using '<', e.g.,

  more < fred.txt

  will display the contents of 'fred.txt' one screenful at a time.

# Pipes and Filters in MS-DOS

- combinations, eg,

    sort < fred.txt > mary.txt

sorts the contents of ‘fred.txt’ line by line and stores the result in ‘mary.txt’.

**2. Pipes in MS-DOS**

- The vertical bar ‘|’ represents a pipe connecting the output of one program to the input of another. Eg,

    dir | more

displays the contents of the current directory one screenful at a time.

- Eg, dir | sort > stdir.txt

sorts the listing of the current directory line by line and stores the result in ‘stdir.txt’.

Murdoch UNIVERSITY

# Pipes and Filters in MS-DOS

**3. Filters in MS-DOS**

- Filters available include FIND and MORE (?) and  GREP.

**Note:** Many MS-DOS versions do not have GREP. An alternative is FINDSTR command available in recent Windows versions. FINDSTR (sort of) emulates GREP, although it does not offer the same functionality.

Also free versions of GREP (for MS-DOS) are available on the web.

# GREP in MS-DOS

- Grep is used to find all lines in a specified file (or files) which contain some string (or pattern of characters).
  - can be used to look in files in the current directory, eg,

    grep Expn *.txt

    lists all the lines in any '.txt' file in the directory which contains the string (or pattern) "Expn".
  - can be used to look in any input file, eg,

    dir | grep Expn

    lists each of the files in the current directory whose name contains "Expn".
- the argument "Expn" to grep can be a simple string or a pattern described in a simple language. The patterns are called ***regular expressions***.

Murdoch UNIVERSITY

# GREP in MS-DOS

- details of regular expressions can be found, e.g.,

http://en.wikipedia.org/wiki/Regular_expression

**Note** that in DOS it is best to quote the expression inside double quotes to prevent special characters being interpreted by DOS.

Murdoch UNIVERSITY

# Regular expressions

A regular expressions (regex or regexp for short) is a string (pattern) consisting of **special** characters (meta-symbols) and **regular** characters. Regular chars are all those that are not special chars.

| Special chars in RE | Description |
|---|---|
| ^ (Caret) | Match expression at the beginning of a line, as in ^A. |
| $ | match expression at the end of a line, as in A$. |
| \ (Back Slash) | turn off the special meaning of the next character, as in \^ |
| . (Period) | match a single character of any value, except end of line |
| [ ] (Brackets) | match any one of the enclosed characters, as in [aeiou] |
| [A-D] | Match a range of characters or numbers, eg [A-Za-z] matches any uppercase or lowercase letters |
| [^ ] | match any single character except those enclosed in [ ], as in [^0-9] |
| * (Asterisk) | match zero or more of the preceding character or expression |

Murdoch UNIVERSITY

# Regular expressions: examples

- dir | grep ^”DIR”

  all the file names starting with DIR (^ means that this must be the start of the line)

- grep “g.*g” *.txt

  all lines in any ‘.txt’ file which contain two letter g’s. (the . matches any character and the .* matches any sequence of any characters)

- grep “g[a-z]*g” *.txt

  all lines in any ‘.txt’ file which contain two letter g’s separated only by lower case letters (the [a-z] matches any lower case)

Murdoch UNIVERSITY

# Acknowledgement

I inherited most of the materials of this unit from A/Prof Pyara Dhillon (who was the coordinator for ICT306).

Murdoch UNIVERSITY

Murdoch
UNIVERSITY

# Summary

Mu
UNIV

# Summary

- Basics of SW Architecture/Design
- SDLC
- Predictive vs Adaptive design
- Pipe and Filter method

Murdoch UNIVERSITY


Murdoch
UNIVERSITY