Module 6: Architectural Design

Software Engineering

School of Computing, Faculty of Engineering

Universiti Teknologi Malaysia

Outline

- Architectural Design and Detailed Design
- Architectural Design Decision
- Architectural View
- Architectural Pattern
  - Model-View-Controller (MVC)
  - Layered
  - Repository
  - Client-Server
  - Pipe and Filter
- Application Architectures

Note: The overall contents of the slide are based on the main reference that is Sommerville (2016) with other references specified directly in respective slides (if any)

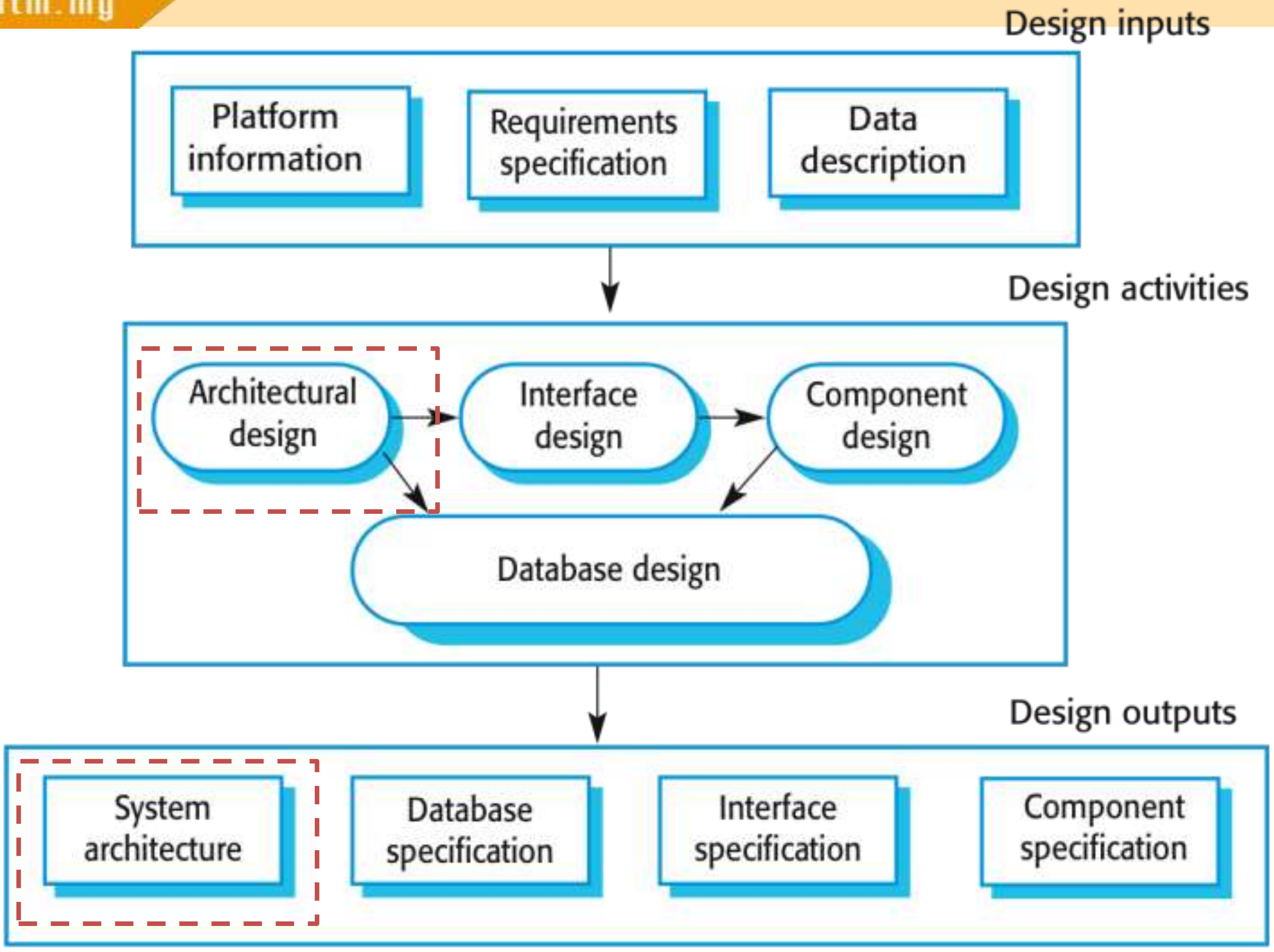
Objectives

- To know the purpose of design and the difference between architectural and detailed design (use case realization)
- To understand the architectural design decision
- To know differences in architectural views and representation using UML diagrams (deployment diagram and details of component diagram)
- To identify the use of different architectural patterns

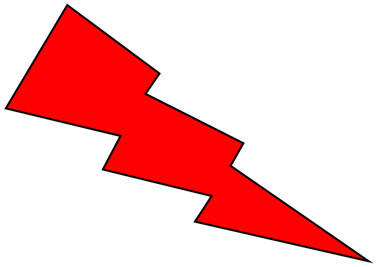
Overview

ARCHITECTURAL DESIGN AND DETAILED DESIGN

Recap on Design Stage

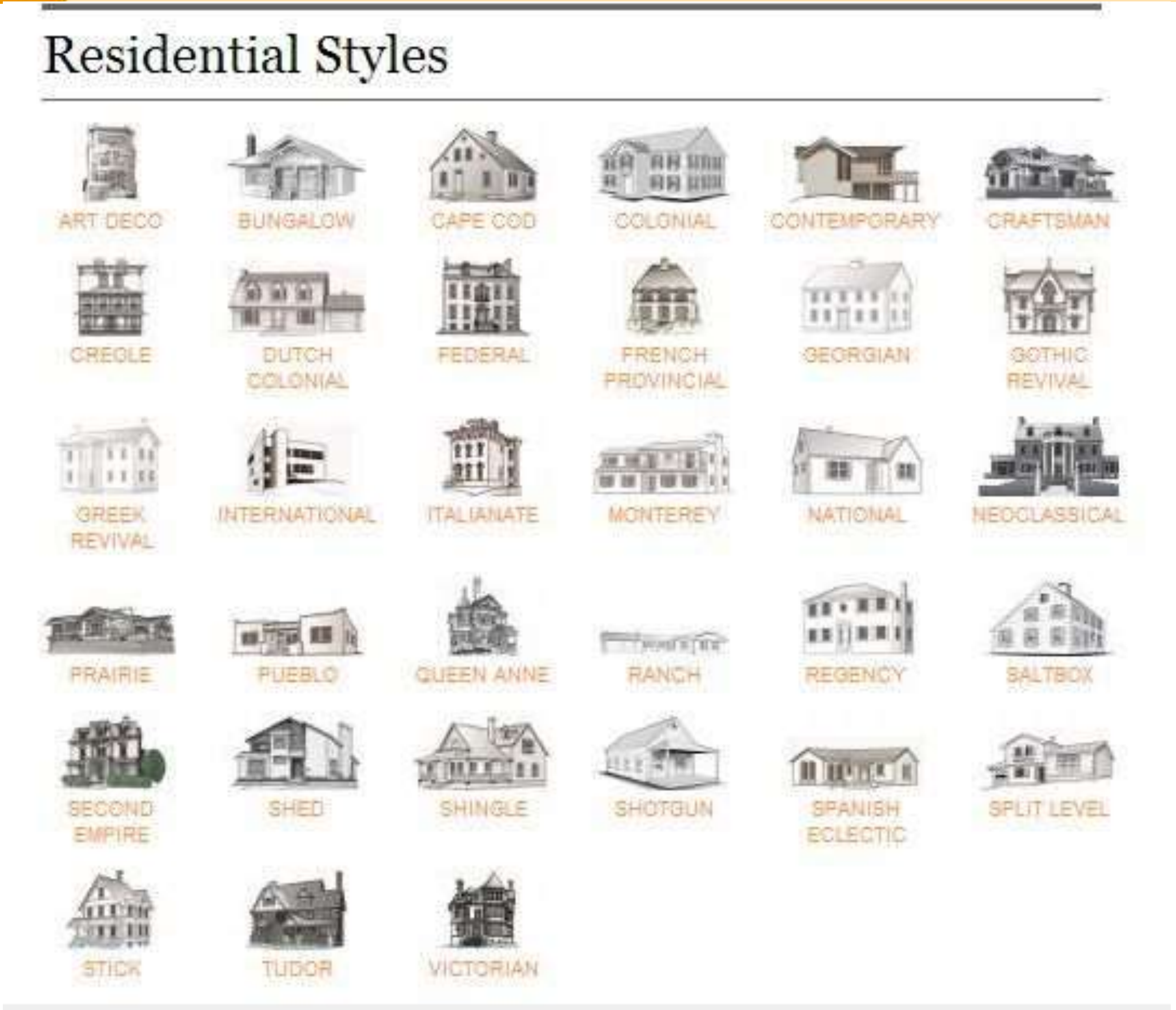


Architecture Analogy for Software vs. House?



Patio
12-8 x 10
Bedroom #1
11-6 x 13
9' Ceiling
Laun.
Breakfast Area
12 x 8
9' Ceiling
Kitchen
12 x 10-4
Bath
Clos.
Bath
Hall
Living Room
17-6 x 12-11
(Clear)
9' Ceiling
Gas Logs
Clos.
Bedroom #2
11-6 x 13
9' Ceiling
Front Porch
17-10 x 5-0

Analogy: Residential Styles

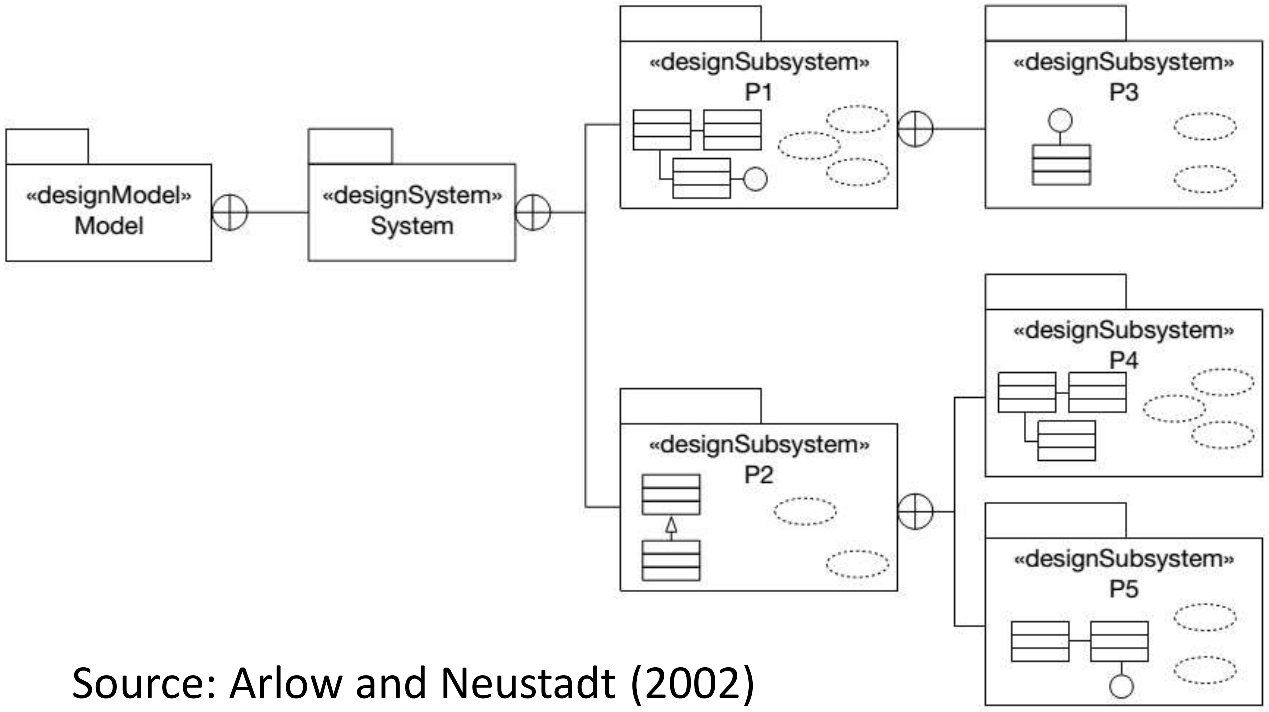


Design Discipline Activities

- Segmented into 6 major activities:
  - Design the support services architecture and deployment environment → (network course)
  - Design the software architecture → **focus of SE lecture**
  - Design the use case realizations → **focus of SE lecture**
  - Design the database → (database course)
  - Design the system and user interfaces (HCI course)
  - Design the system security and controls (network security; database security)

From Analysis to Design: Design Artefacts Metamodel

- Design model contains exactly **one design system** that contains **many design subsystems** (also known as “package” that can also be introduced in the analysis)

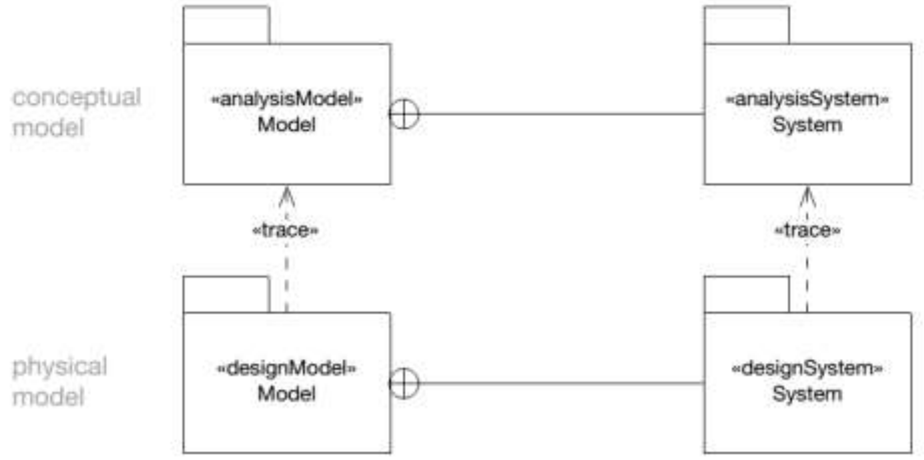


A subsystem is a part of the physical system.

Source: Arlow and Neustadt (2002)

Relationships between Analysis and Design (Model and System)

- There is a simple **«trace» relationship** between the analysis and design models – the design model is based on the analysis model, and can be considered to be just a refinement and elaboration thereof
- Similarly, there is a simple, one-to-one «trace» relationship between the analysis system and the design system



Source: Arlow and Neustadt (2002)

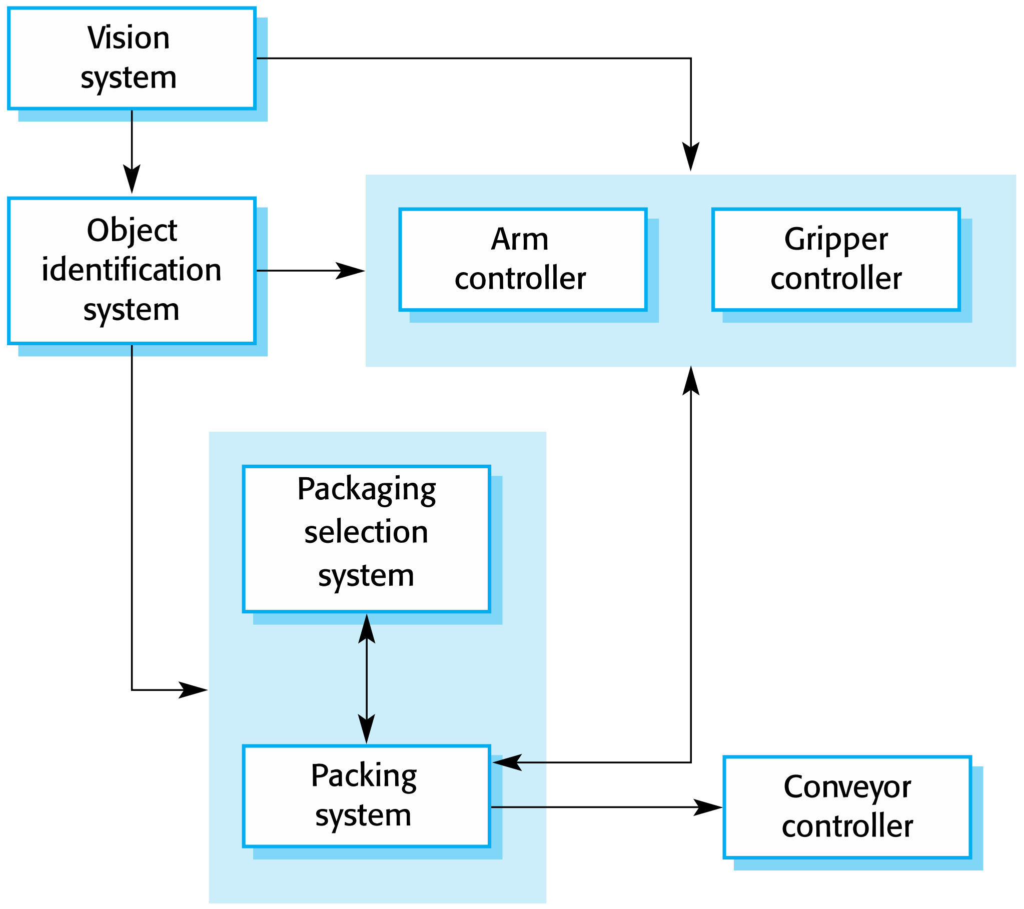
Architectural Design: Design the Software Architecture

- Architectural design is concerned with understanding how a software system should be organized and designing the **overall structure** of that system
- Architectural design is the critical link between design and requirements engineering, as it identifies the **main structural components in a system** and the relationships between them
- The output of the architectural design process is an **architectural model** that describes how the system is organized as a set of communicating components

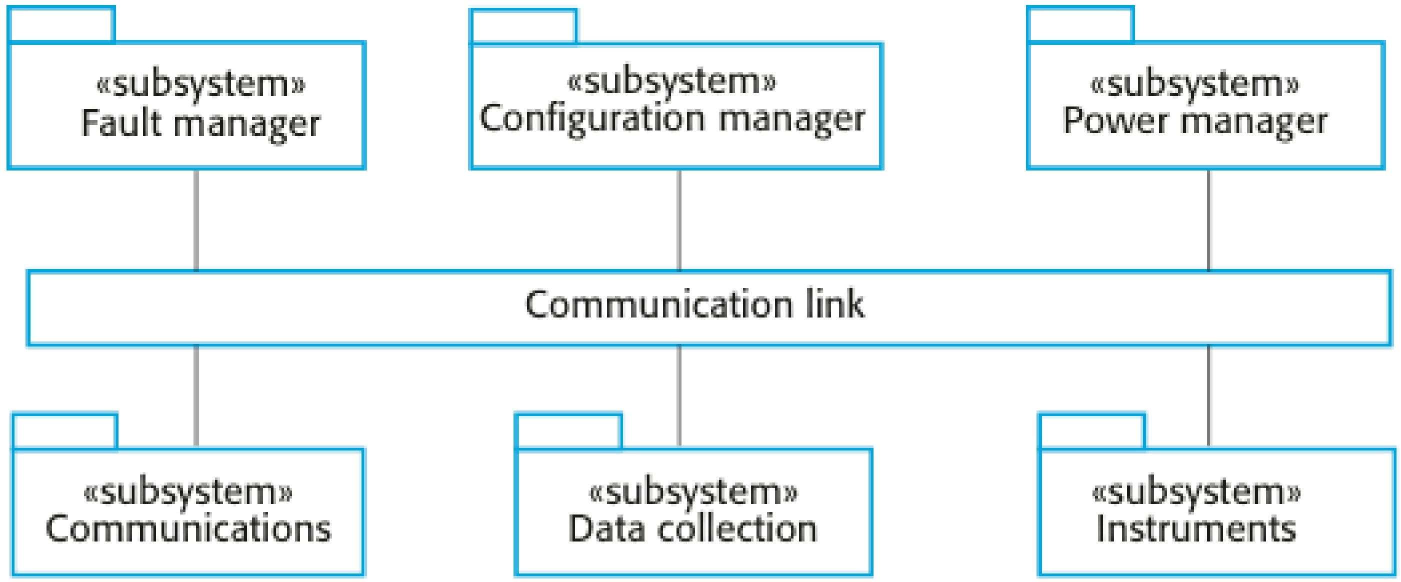
Design Use Case Realizations

- Use case realizations offer a lower-level view
- Two-tiered focus:
  - Class interactions supporting a particular use case
  - Interactions among software, users, and external systems
- Design typically spread over many iterations
- UML **design class diagrams** and **interaction diagrams** (sequence diagram) ➔ document design [to be elaborated in Module 7]

Example: Architecture of a Packing Robot Control System



Example: High Level Architecture of Weather Station System


«subsystem» Fault manager
«subsystem» Configuration manager
«subsystem» Power manager
Communication link
«subsystem» Communications
«subsystem» Data collection
«subsystem» Instruments

Architectural Abstraction

- Architecture in the **small** is concerned with the architecture of individual programs
  - Concerned with the way that an individual program is **decomposed into components**
- Architecture in the **large** is concerned with the architecture of complex enterprise systems that include **other systems, programs, and program components**
  - These enterprise systems are distributed over different computers, which may be owned and managed by different companies

Advantages of Explicit Architecture

- Stakeholder communication
  - Architecture may be used as a **focus of discussion** by system stakeholders
- System analysis
  - Means that analysis of whether the system can **meet its non-functional requirements** is possible
- Large-scale reuse
  - The architecture may be **reusable** across a range of systems
  - Product-line architectures may be developed

Architectural Representations

- Simple, informal **block diagrams** showing entities and relationships are the most frequently used method for documenting software architectures
- But these have been criticised because they **lack semantics**, do not show the types of relationships between entities nor the visible properties of entities in the architecture
- Depends on the use of architectural models, the requirements for **model semantics depends on how the models are used**

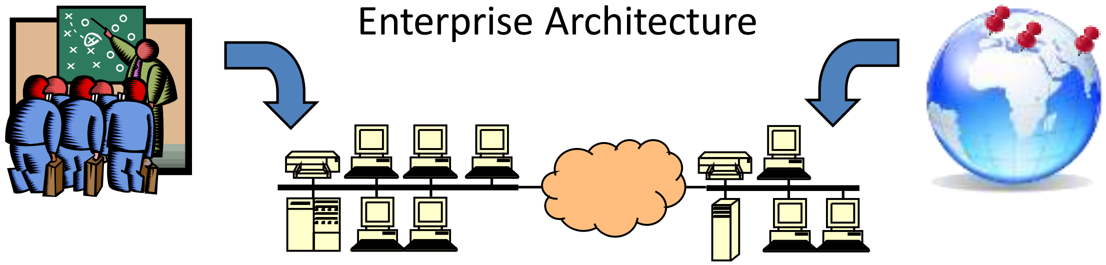
Box and Line Diagrams

- Very **abstract** - they do not show the nature of component relationships nor the externally visible properties of the sub-systems
- However, **useful for communication** with stakeholders and for project planning

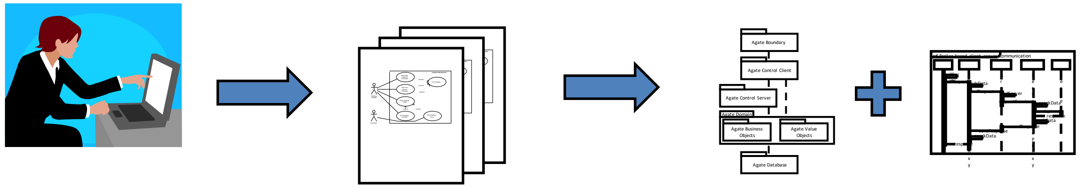
Use of Architectural Models

- As a way of **facilitating discussion** about the system design
  - A high-level architectural view of a system is useful for communication with system stakeholders and project planning because it is **not cluttered with detail**
  - Stakeholders can relate to it and understand an abstract view of the system, then discuss the system as a whole without being confused by detail
- As a way of **documenting an architecture** that has been designed
  - The aim here is to produce a complete system model that shows the different components in a system, their interfaces and their connections

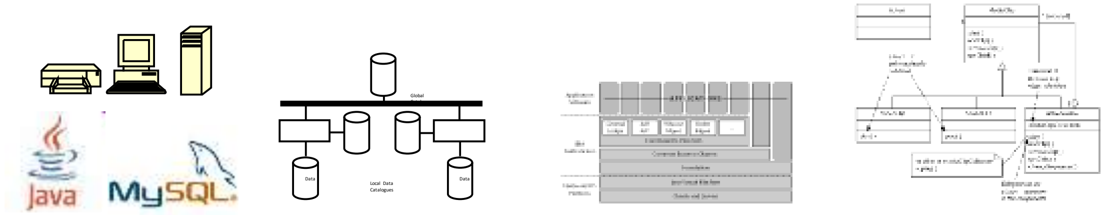
Enterprise Architecture



Alignment of IT architecture to business strategy and structure, and existing IT

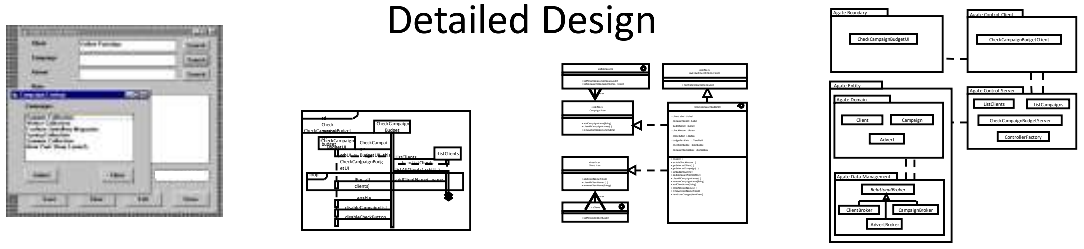


Meeting users’ requirements and determining the high-level structure of the system



Choosing technologies and frameworks, setting standards, and applying patterns

Detailed Design



Designing user interfaces, interactions, classes, data storage

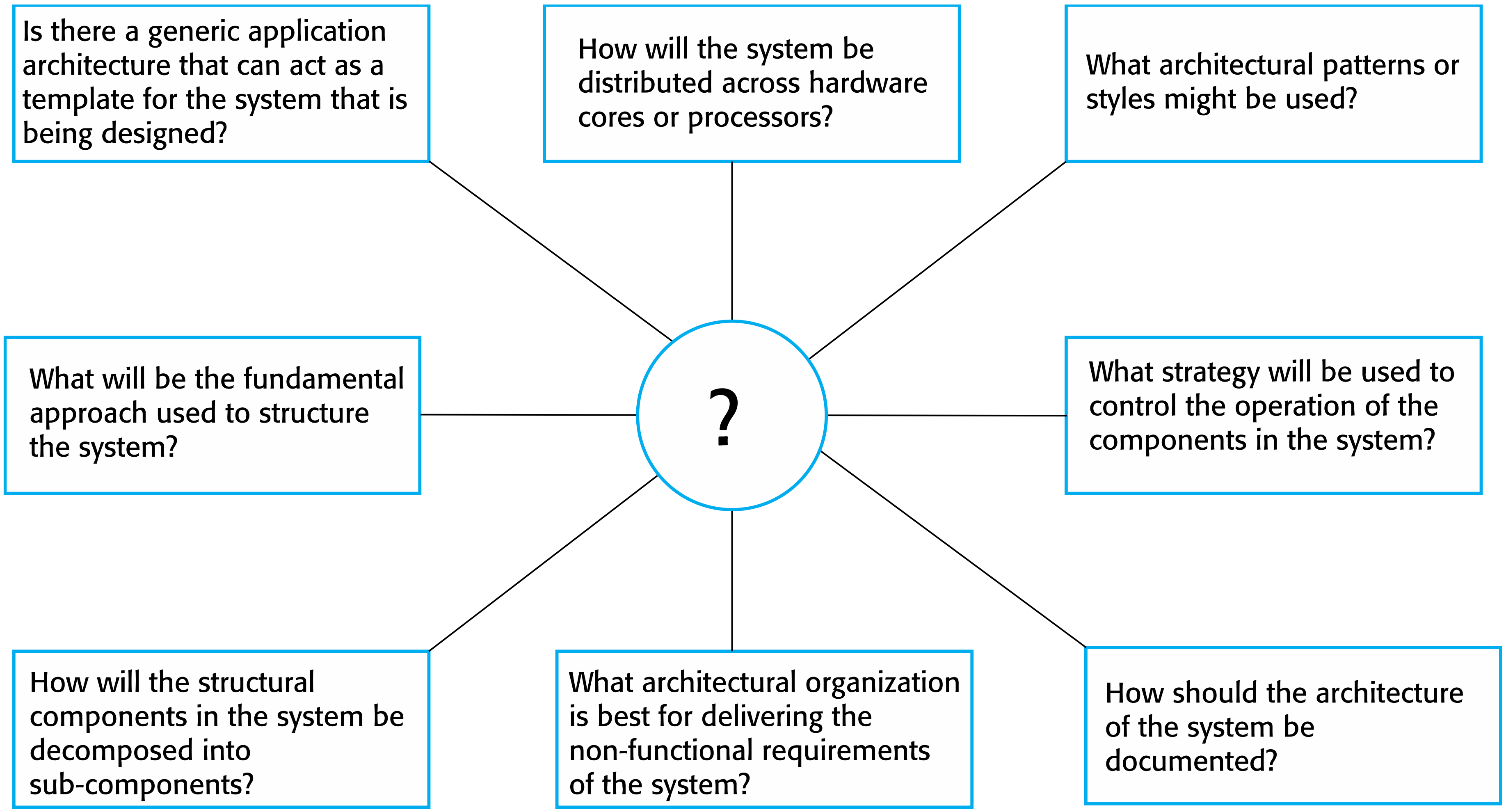
© 2010 Bennett, McRobb and Farmer

ARCHITECTURAL DESIGN DECISION

Architectural Design Decision

- Architectural design is a **creative process,** so the process differs depending on the type of system being developed
- However, a number of common decisions span all design processes and these decisions **affect the non-functional characteristics** of the system

Architectural Design Decision


Is there a generic application architecture that can act as a template for the system that is being designed?
How will the system be distributed across hardware cores or processors?
What architectural patterns or styles might be used?
What will be the fundamental approach used to structure the system?
?
What strategy will be used to control the operation of the components in the system?
How will the structural components in the system be decomposed into sub-components?
What architectural organization is best for delivering the non-functional requirements of the system?
How should the architecture of the system be documented?

Architecture Reuse

- Systems in the **same domain often have similar architectures** that reflect domain concepts
- Application **product lines** are built around a core architecture with variants that satisfy particular customer requirements
- The architecture of a system may be designed around one of more **architectural patterns or 'styles'** to capture the essence of an architecture and can be instantiated in different ways

Architecture and System Characteristics (Non-Functional Requirements)

- **Performance**
  - Localise critical operations and minimise communications
  - Use large rather than fine-grain components
- **Security**
  - Use a layered architecture with critical assets in the inner layers
- **Safety**
  - Localise safety-critical features in a small number of sub-systems
- **Availability**
  - Include redundant components and mechanisms for fault tolerance
- **Maintainability**
  - Use fine-grain, replaceable components

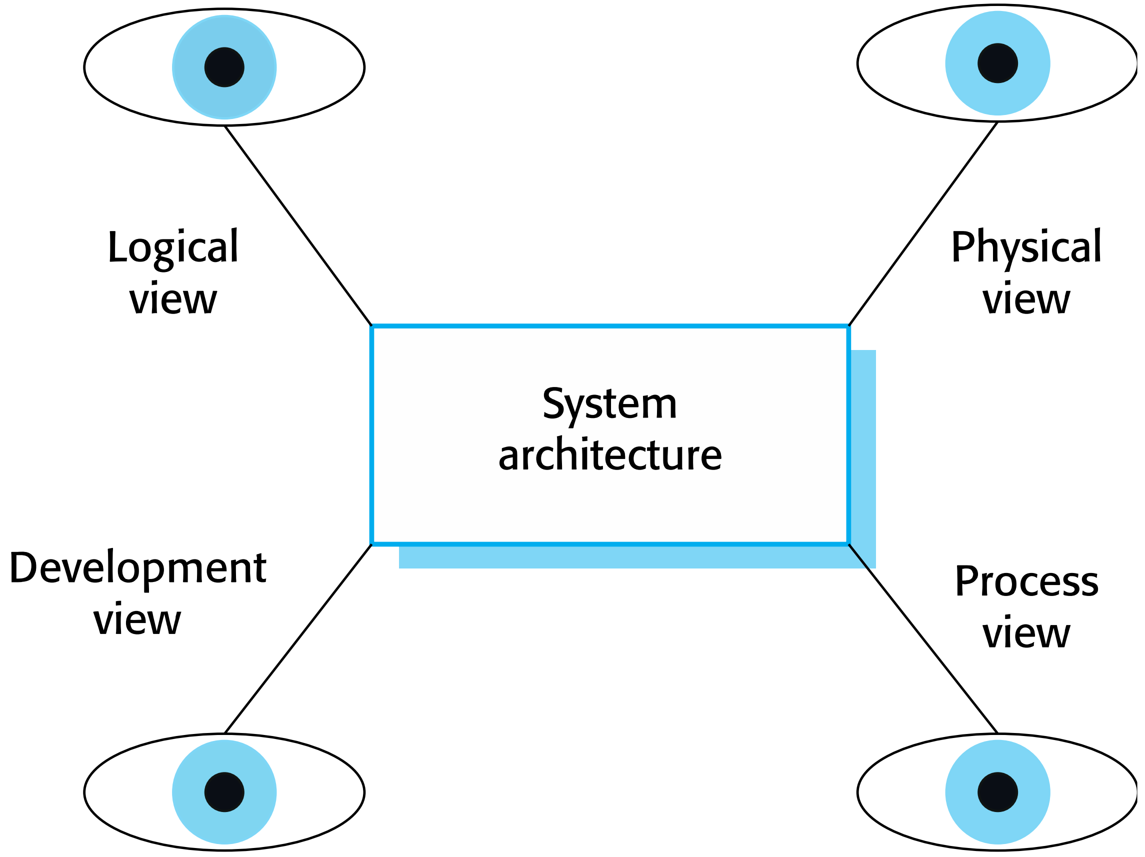
4+1 View Model

ARCHITECTURAL VIEW

Architectural Views

- What **views or perspectives** are useful when designing and documenting a system's architecture?
- What **notations** should be used for describing architectural models?
- Each architectural model only shows one view or perspective of the system
  - It might show how a system is **decomposed into modules**, how the run-time **processes interact** or the different ways in which system **components are distributed** across a network
  - For both design and documentation, we usually need to present multiple views of the software architecture

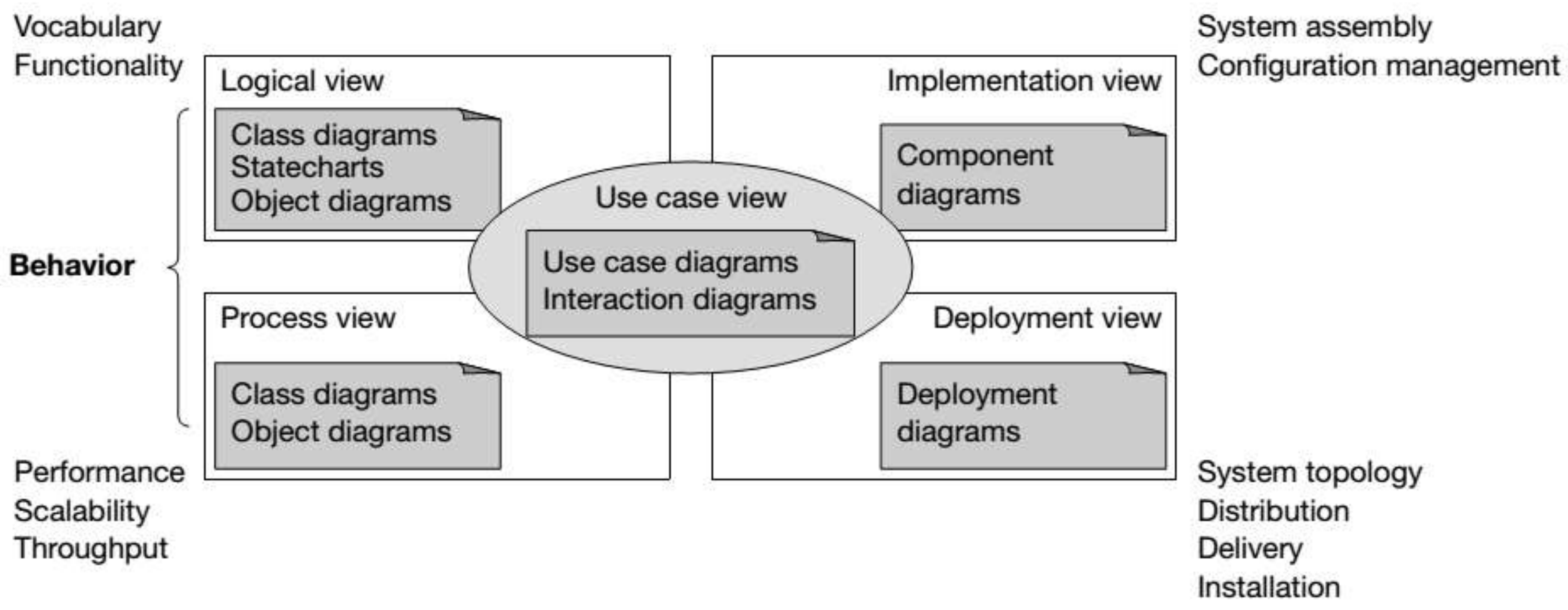
Architectural Views



4 + 1 View Model of Software Architecture

- **Logical view** shows the key abstractions in the system as objects or object classes
- **Process view** shows how, at run-time, the system is composed of interacting processes
- **Development (or implementation) view** shows how the software is decomposed for development
- **Physical (or deployment) view** shows the system hardware and how software components are distributed across the processors in the system
- Related using **use cases or scenarios** (+1)

UML Diagrams to Represent Architectural Views (4+1 View Model)



Source: Kruchten (2000) in Arlow and Neustadt (2002)

Note: Sommerville (2016) states implementation view as development view, while deployment view as physical view

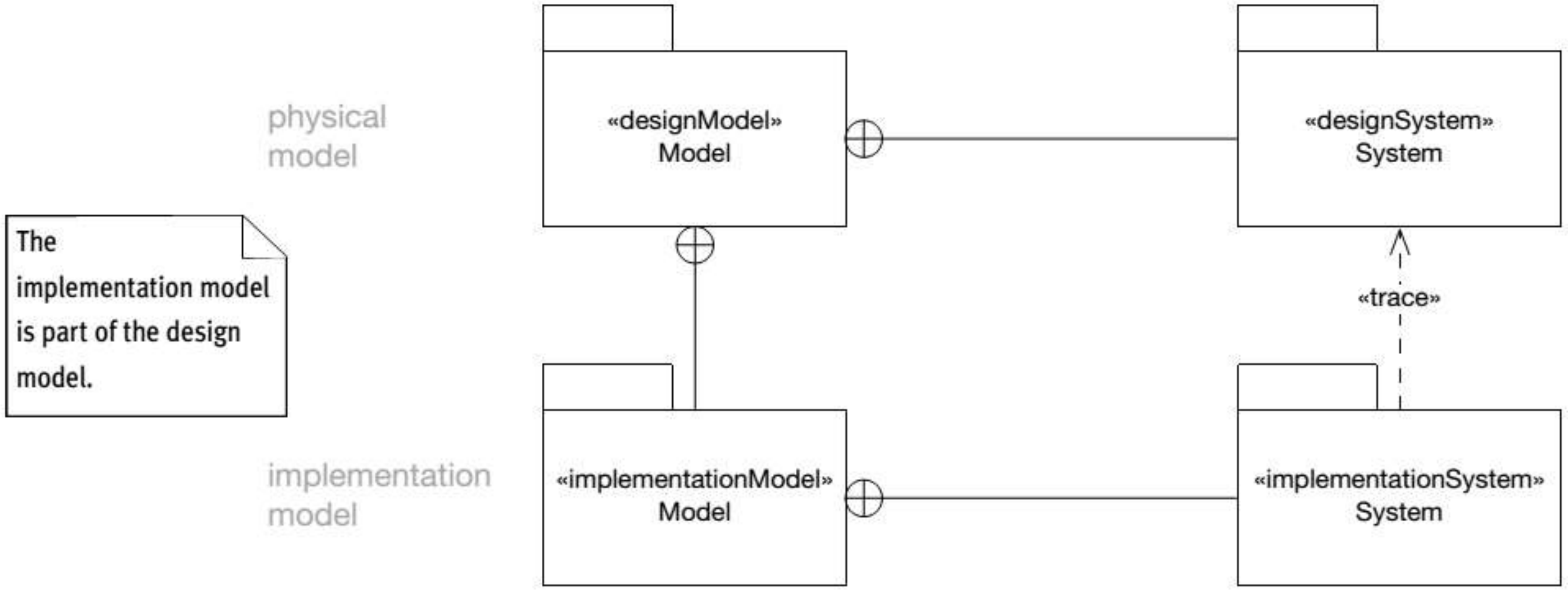
From Design to Implementation

- The key artefact of the implementation workflow from the point of view of the OO analyst or designer is the implementation model
- This model consists of **2 types of diagram**:
  - **Component diagram**: models the dependencies between the software components that constitute the system
  - **Deployment diagram**: models the physical computational nodes on which the software will be deployed, and the relationships between those nodes

Source: Arlow and Neustadt (2002)

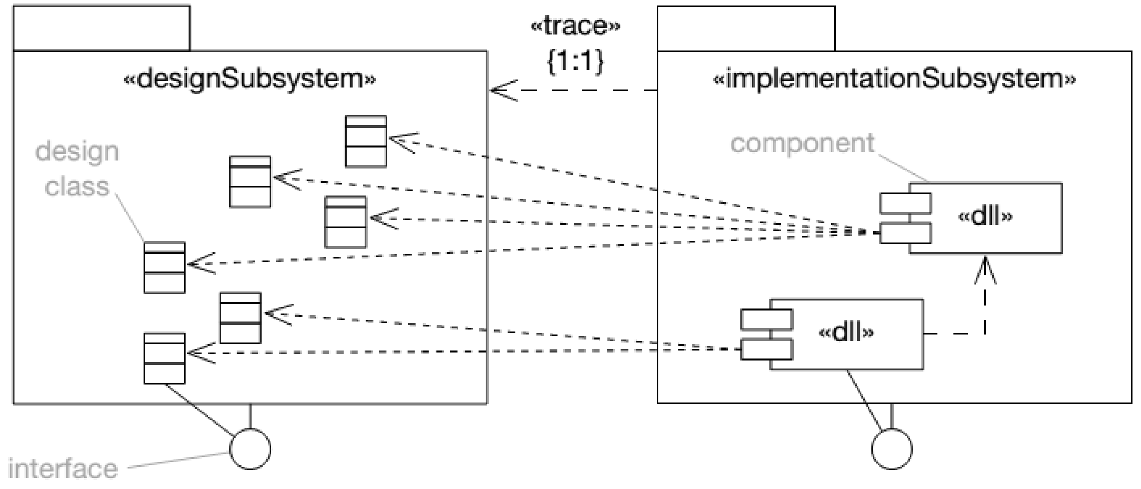
Relationship between Implementation Model and Design Model

- Implementation model is really just the implementation view of a design model i.e. it is part of the design model, see the relationship <<trace>>


physical model
«designModel» Model
«designSystem» System
The implementation model is part of the design model.
«trace»
implementation model
«implementationModel» Model
«implementationSystem» System

Relationship between Design Subsystems and Implementation Subsystems

- There is a one-to-one «trace» relationship between design subsystems and implementation subsystems
- Design subsystems contain **design classes** but implementation subsystems contain **components that package those classes**



Model-View-Controller (MVC), Layered, Repository, Client-Server, Pipe and Filter

ARCHITECTURAL PATTERN

Architectural Patterns

- Patterns (also known as styles) are a means of representing, sharing and reusing knowledge
- An architectural pattern is a **stylized description of good design practice**, which has been tried and tested in different environments
- Patterns should include information about when they are and when they are not useful
- Patterns may be represented using tabular and graphical descriptions

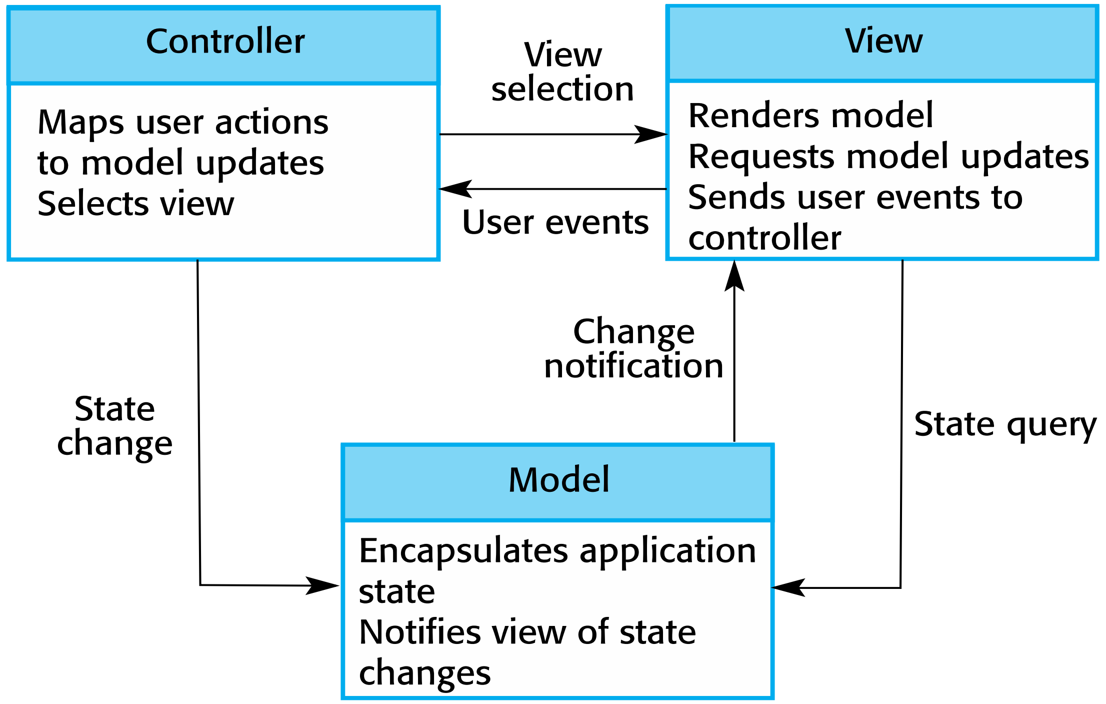
Common Architectural Patterns

- Model-View-Controller (MVC)
- Layered
- Repository
- Client-Server
- Pipe and Filter

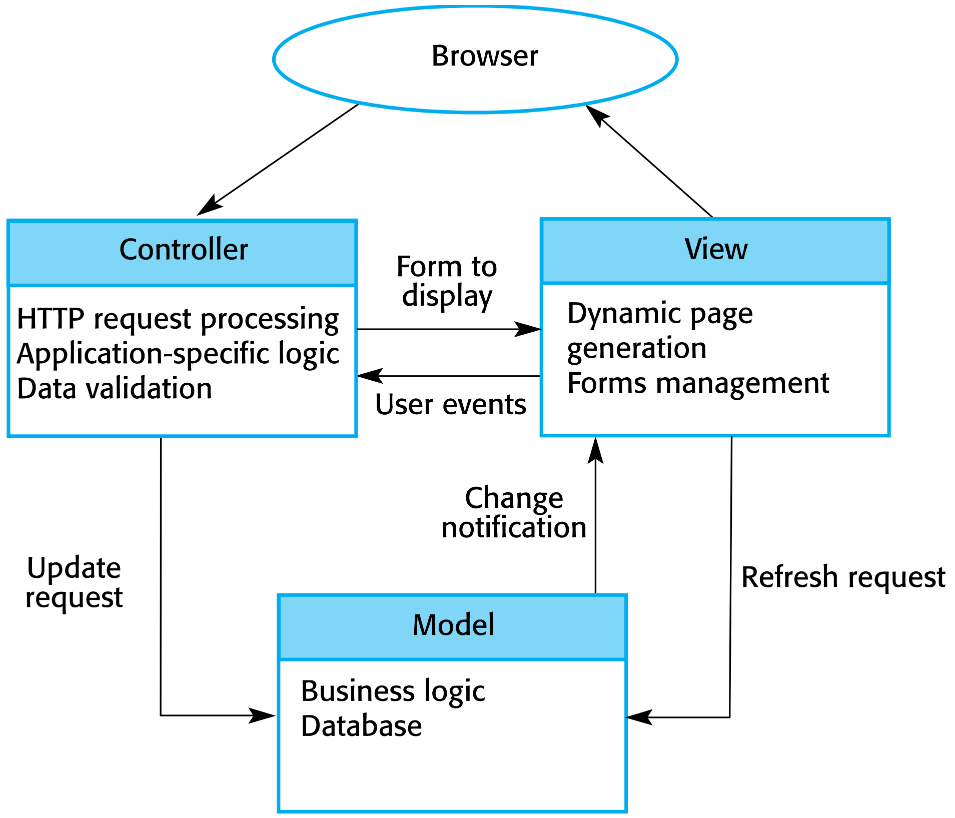
Model-View-Controller (MVC) Pattern

Name	MVC (Model-View-Controller)
Description	Separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other. The Model component manages the system data and associated operations on that data. The View component defines and manages how the data is presented to the user. The Controller component manages user interaction (e.g., key presses, mouse clicks, etc.) and passes these interactions to the View and the Model.
Example	The architecture of a web-based application system organized using the MVC pattern (see the example slide).
When used	Used when there are multiple ways to view and interact with data. Also used when the future requirements for interaction and presentation of data are unknown.
Advantages	Allows the data to change independently of its representation and vice versa. Supports presentation of the same data in different ways with changes made in one representation shown in all of them.
Disadvantages	Can involve additional code and code complexity when the data model and interactions are simple.

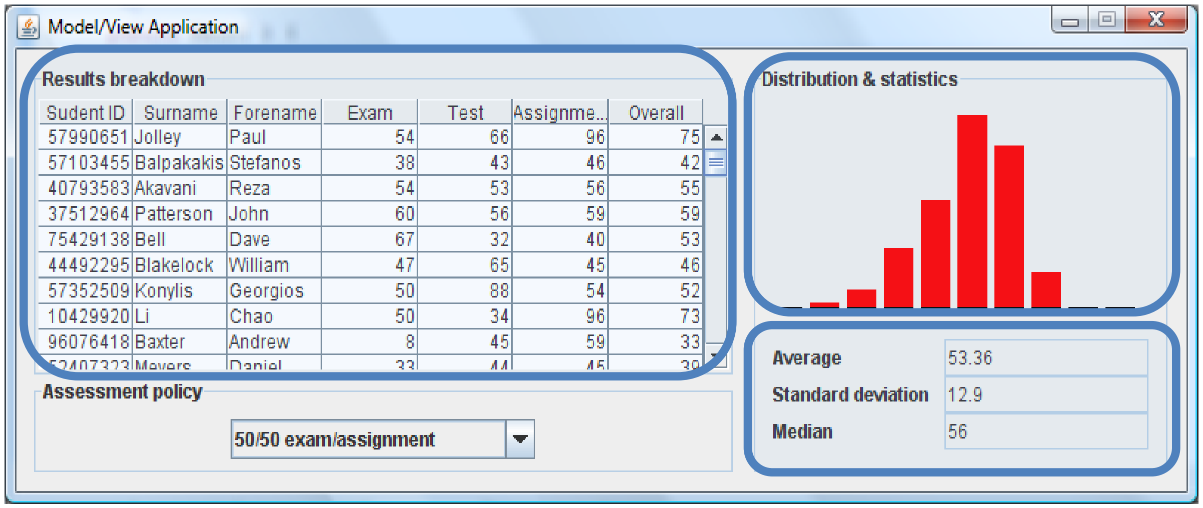
Organization of MVC



Web Application Architecture Using MVC



Example: MVC Architecture Used for a Dashboard



Model/View Application

Results breakdown

Sudent ID	Surname	Forename	Exam	Test	Assignme...	Overall
57990651	Jolley	Paul	54	66	96	75
57103455	Balpakakis	Stefanos	38	43	46	42
40793583	Akavani	Reza	54	53	56	55
37512964	Patterson	John	60	56	59	59
75429138	Bell	Dave	67	32	40	53
44492295	Blakelock	William	47	65	45	46
57352509	Konylis	Georgios	50	88	54	52
10429920	Li	Chao	50	34	96	73
96076418	Baxter	Andrew	8	45	59	33
52407323	Meyers	Daniel	33	44	45	39

Assessment policy

50/50 exam/assignment

Distribution & statistics

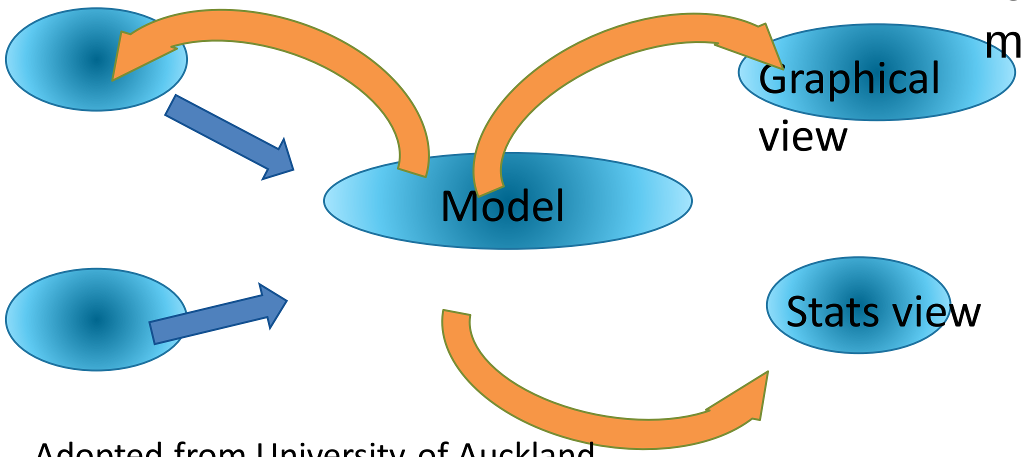
Average	53.36
Standard deviation	12.9
Median	56

Many desktop applications provide multiple views of some data model.

Invariant : all views should offer a mutually consistent representation of the model.

JTable component instance

JCombobox component instance



Change request

Update notification

Adopted from University of Auckland

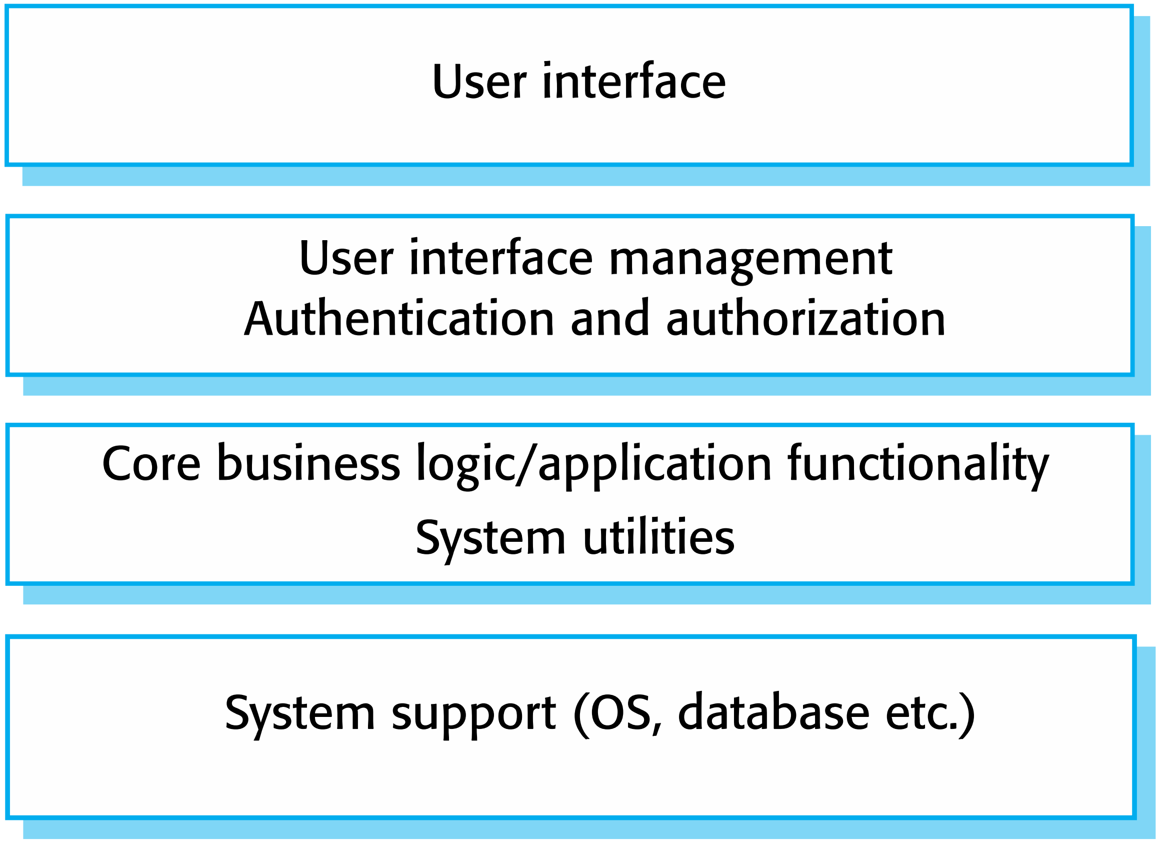
Layered Architecture

- Used to model the interfacing of sub-systems
- Organises the system into **a set of layers** (or abstract machines) each of which provide a set of services
- Supports the **incremental development** of sub-systems in different layers
- When a layer interface changes, only the adjacent layer is affected
- As layered systems localize machine dependencies, this makes it **easier to provide multi-platform implementations** of an application system

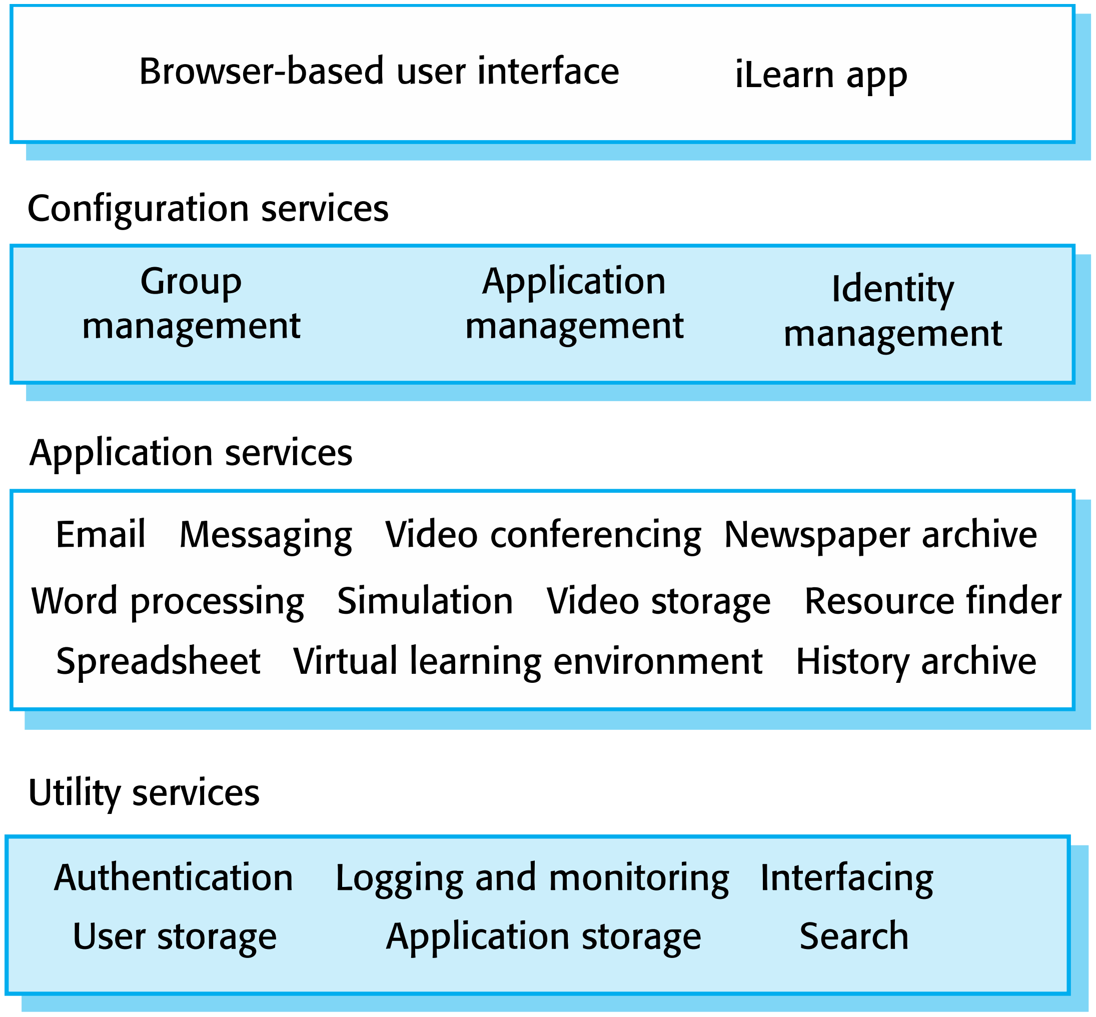
Layered Architecture Pattern

Name	Layered architecture
Description	Organizes the system into layers with related functionality associated with each layer. A layer provides services to the layer above it so the lowest-level layers represent core services that are likely to be used throughout the system. See next slide.
Example	A layered model of a system for sharing copyright documents held in different libraries, as shown in the iLearn example slide.
When used	Used when building new facilities on top of existing systems; when the development is spread across several teams with each team responsibility for a layer of functionality; when there is a requirement for multi-level security.
Advantages	Allows replacement of entire layers so long as the interface is maintained. Redundant facilities (e.g., authentication) can be provided in each layer to increase the dependability of the system.
Disadvantages	In practice, providing a clean separation between layers is often difficult and a high-level layer may have to interact directly with lower-level layers rather than through the layer immediately below it. Performance can be a problem because of multiple levels of interpretation of a service request as it is processed at each layer.

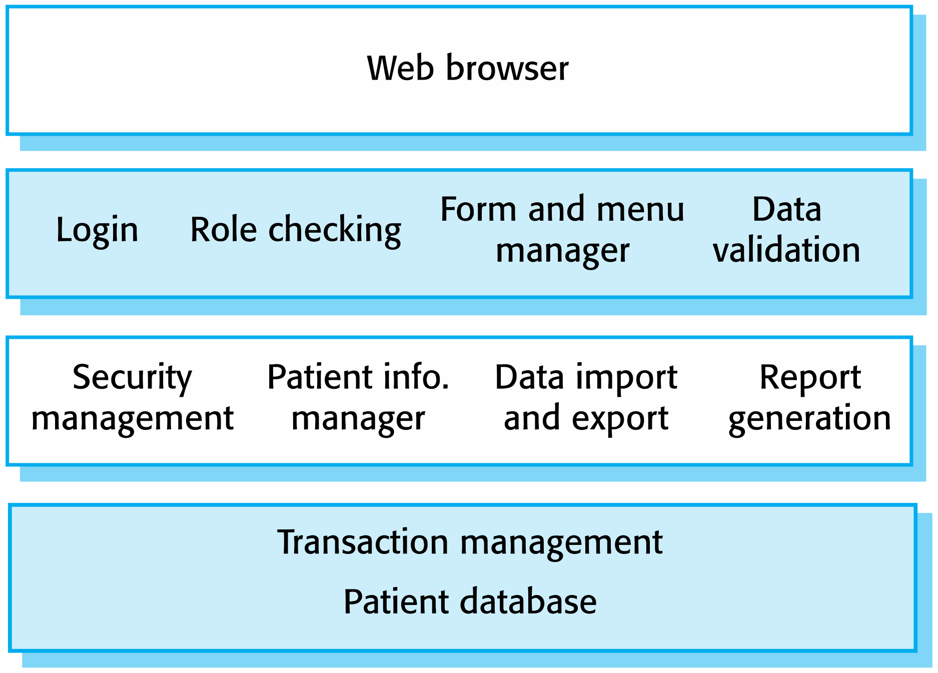
Generic Layered Architecture


User interface
User interface management
Authentication and authorization
Core business logic/application functionality
System utilities
System support (OS, database etc.)

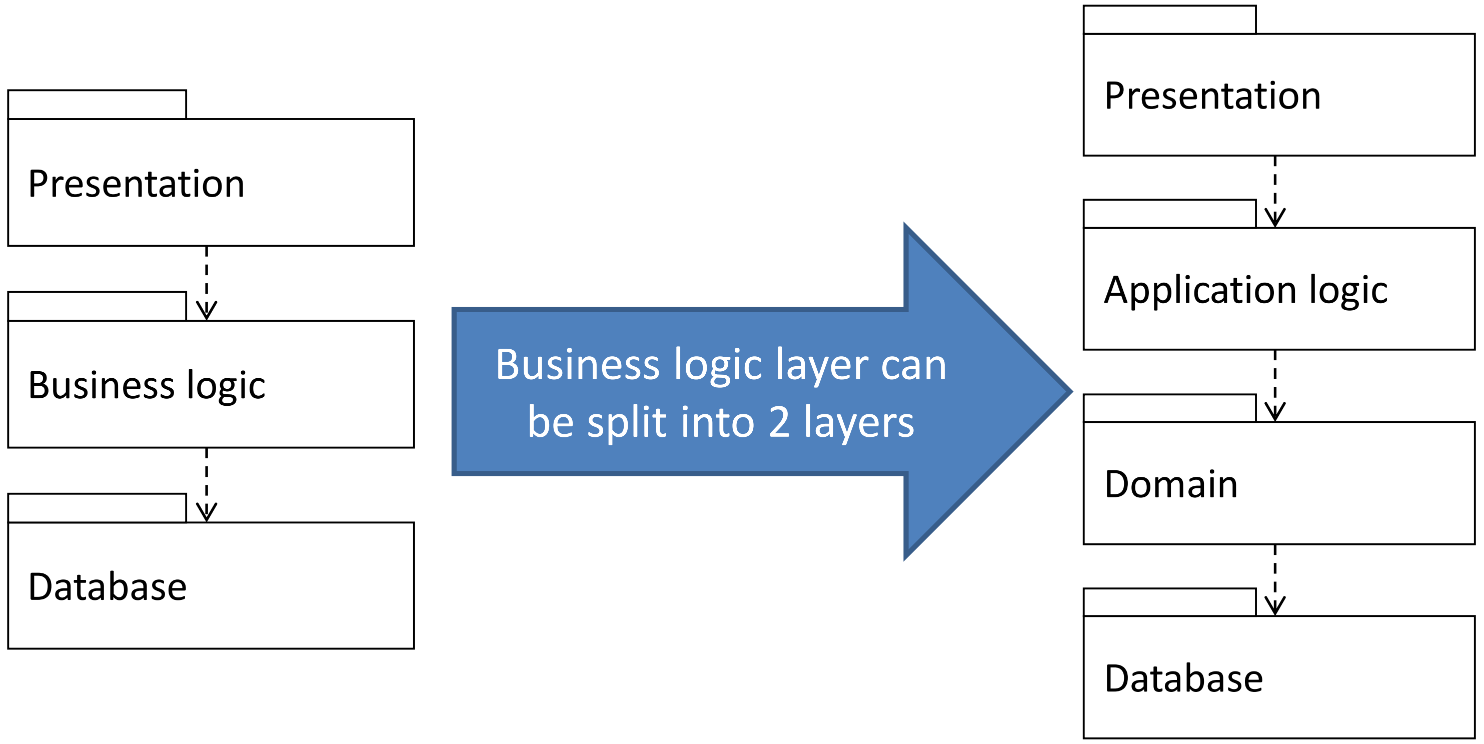
Example: Architecture of iLearn System



Example: Architecture of Mentcare System



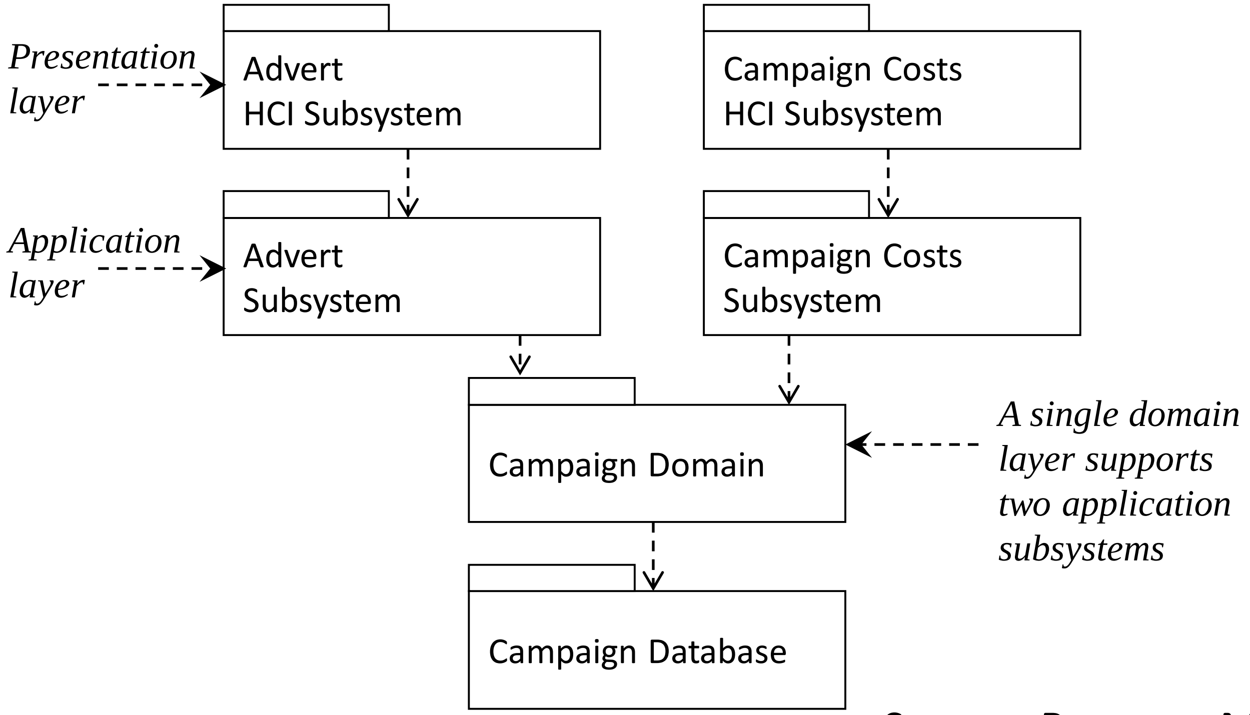
Three and Four Layer Architectures



Source: Bennett, McRobb and Farmer (2010)

Example: Partitioned Subsystems in Four Layer Architecture

- Loosely coupled subsystems, each delivering a single service or coherent group of services



E.g. Agate campaign management system

Source: Bennett, McRobb and Farmer (2010)

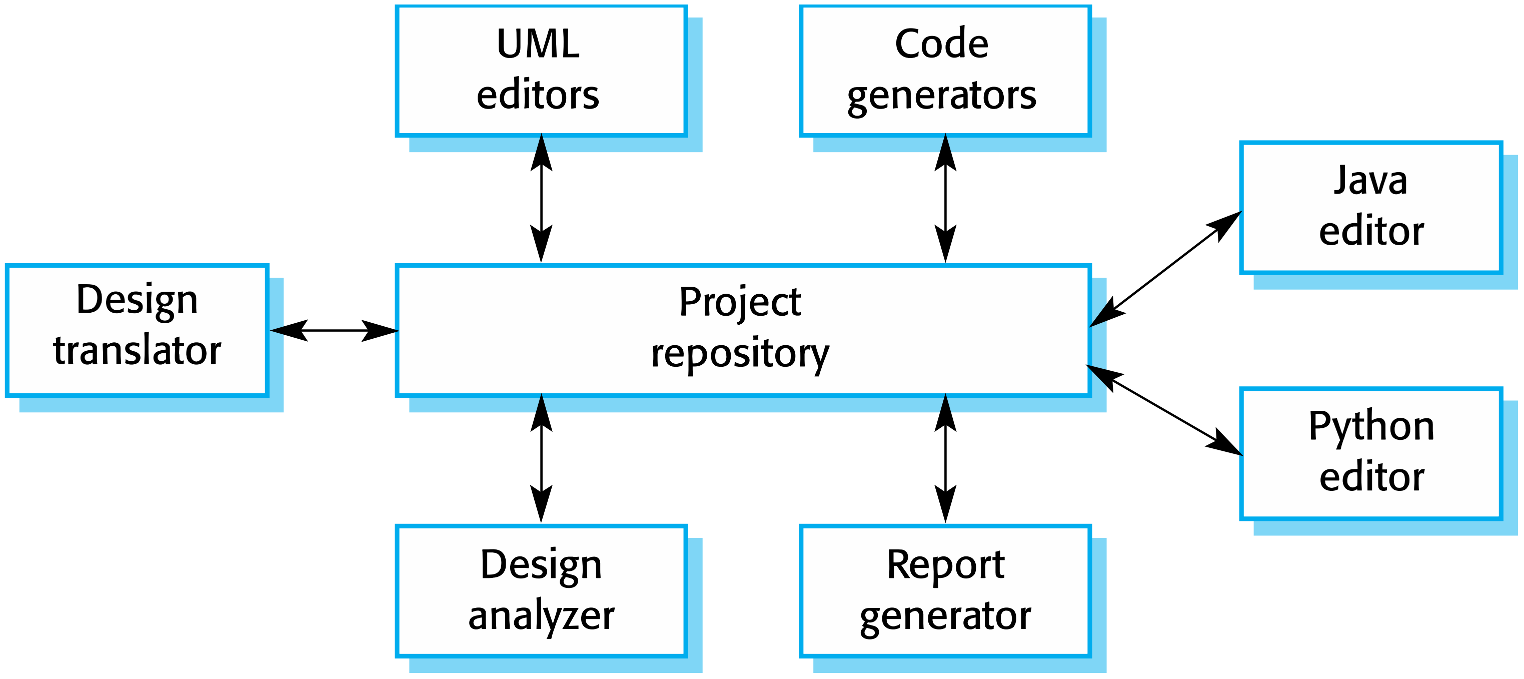
Repository Architecture

- Sub-systems must **exchange data** that may be done in two ways:
  - – Shared data is held in a central database or repository and may be accessed by all sub-systems
  - – Each sub-system maintains its own database and passes data explicitly to other sub-systems
- When **large amounts of data are to be shared**, the repository model of sharing is most commonly used a this is an efficient data sharing mechanism

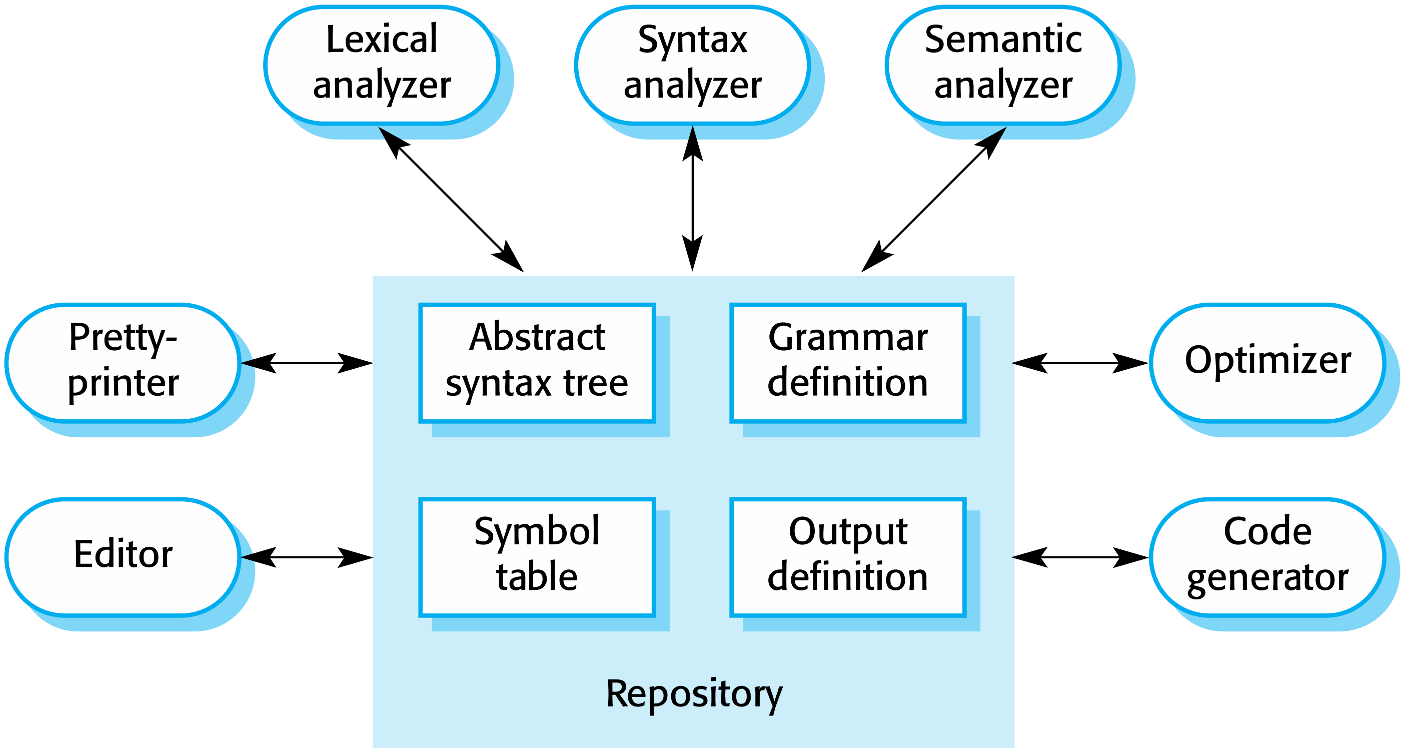
Repository Pattern

Name	Repository
Description	All data in a system is managed in a central repository that is accessible to all system components. Components do not interact directly, only through the repository.
Example	An example of an IDE (see next slide) where the components use a repository of system design information. Each software tool generates information which is then available for use by other tools.
When used	Should use this pattern when we have a system in which large volumes of information are generated that has to be stored for a long time. May also use it in data-driven systems where the inclusion of data in the repository triggers an action or tool.
Advantages	Components can be independent—they do not need to know of the existence of other components. Changes made by one component can be propagated to all components. All data can be managed consistently (e.g., backups done at the same time) as it is all in one place.
Disadvantages	The repository is a single point of failure so problems in the repository affect the whole system. May be inefficiencies in organizing all communication through the repository. Distributing the repository across several computers may be difficult.

Example: Repository Architecture for an IDE



Example: Repository Architecture for Language Processing System



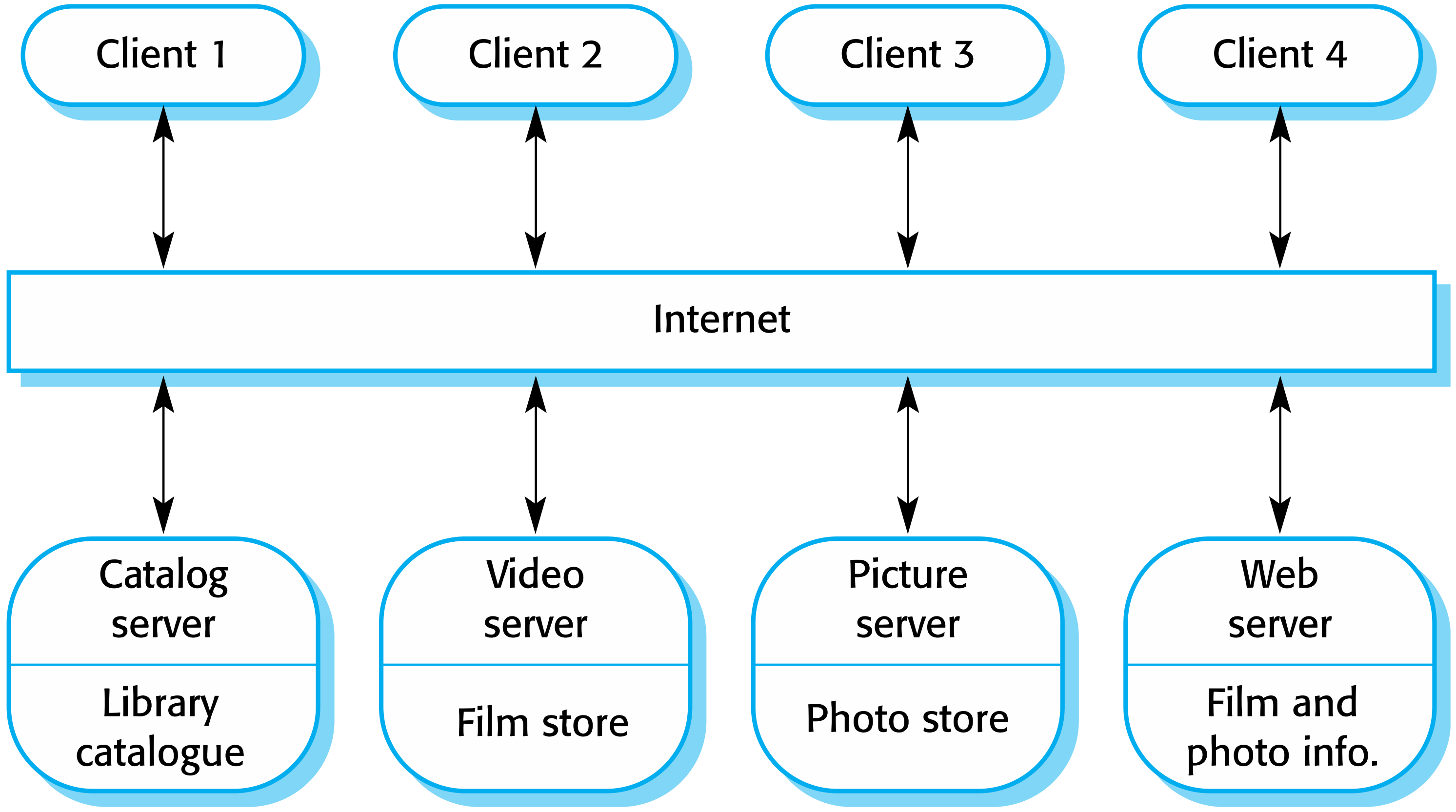
Client-Server Architecture

- Distributed system model which shows how data and processing is **distributed** across a range of components
  - Can be implemented on a single computer
- Set of stand-alone **servers** which provide specific services such as printing, data management, etc.
- Set of **clients** which call on these services
- Network which allows clients to access servers

Client-Server Pattern

Name	Client-server
Description	In a client–server architecture, the functionality of the system is organized into services, with each service delivered from a separate server. Clients are users of these services and access servers to make use of them.
Example	Example of a film and video/DVD library organized as a client–server system (see next slide).
When used	Used when data in a shared database has to be accessed from a range of locations. Because servers can be replicated, may also be used when the load on a system is variable.
Advantages	The principal advantage of this model is that servers can be distributed across a network. General functionality (e.g. a printing service) can be available to all clients and does not need to be implemented by all services.
Disadvantages	Each service is a single point of failure so susceptible to denial of service attacks or server failure. Performance may be unpredictable because it depends on the network as well as the system. May be management problems if servers are owned by different organizations.

Example: Client-Server Architecture for Film Library



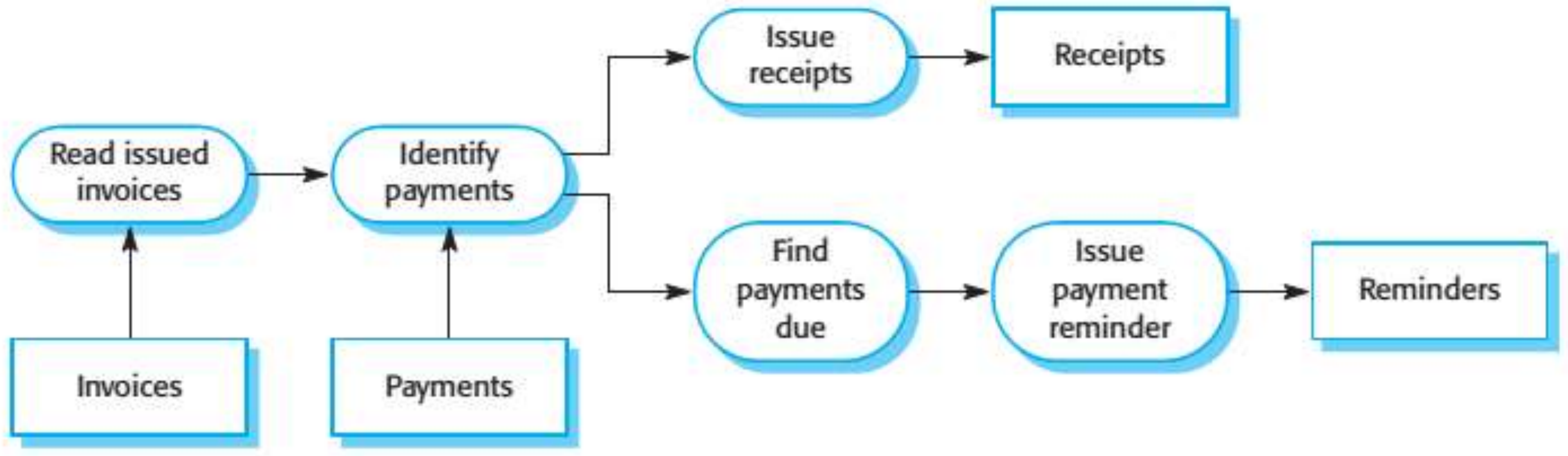
Pipe and Filter Architecture

- Functional transformations **process their inputs to produce outputs**
- May be referred to as a pipe and filter model (as in UNIX shell)
- Variants of this approach are very common
- When transformations are sequential, this is a **batch sequential model** which is extensively used in data processing systems
- Not really suitable for interactive systems

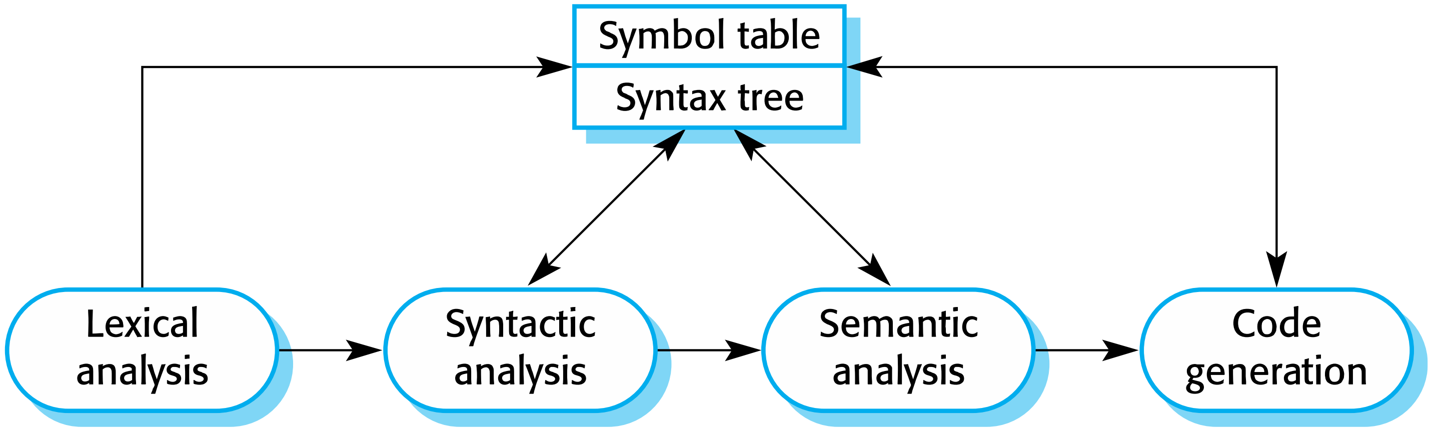
Pipe and Filter Pattern

Name	Pipe and filter
Description	The processing of the data in a system is organized so that each processing component (filter) is discrete and carries out one type of data transformation. The data flows (as in a pipe) from one component to another for processing.
Example	Example of a pipe and filter system used for processing invoices (see next slide).
When used	Commonly used in data processing applications (both batch- and transaction-based) where inputs are processed in separate stages to generate related outputs.
Advantages	Easy to understand and supports transformation reuse. Workflow style matches the structure of many business processes. Evolution by adding transformations is straightforward. Can be implemented as either a sequential or concurrent system.
Disadvantages	The format for data transfer has to be agreed upon between communicating transformations. Each transformation must parse its input and unparse its output to the agreed form. This increases system overhead and may mean that it is impossible to reuse functional transformations that use incompatible data structures.

Example: Pipe and Filter Architecture Used in Payments System



Example: Pipe and Filter Used for Compiler Architecture



Transaction Processing System, Information Systems, Language processing system,

APPLICATION ARCHITECTURES

Application Architectures

- Application systems are intended to meet a business or an organizational need.
- Therefore, as well as general business functions, all phone companies need systems to connect and meter calls, manage their network and issue bills to customers.
- Consequently, the **application systems** used by these businesses also **have much in common.**
- The **application architecture may be reimplemented** when developing new systems.
- However, for many business systems, application architecture reuse is implicit when generic application systems are configured to create a new application.

Used of Model Application Architectures

- As a starting point for the architectural design process If you are unfamiliar with the type of application that you are developing, you can base your initial design on a generic application architecture. You then specialize this for the specific system that is being developed.
- As a design checklist If you have developed an architectural design for an application system, you can compare this with the generic application architecture. You can check that your design is consistent with the generic architecture.
- As a way of organizing the work of the development team The application architectures identify stable structural features of the system architectures, and in many cases, it is possible to develop these in parallel.

Used of Model Application Architectures…cont

- As a means of assessing components for reuse If you have components you might be able to reuse, you can compare these with the generic structures to see whether there are comparable components in the application architecture.
- As a vocabulary for talking about applications If you are discussing a specific application or trying to compare applications, then you can use the concepts
- Example of application architecture: Transaction processing system, information system, and language processing system.

Transaction Processing Systems

- Transaction processing systems are designed to **process user requests** for information from a database, or requests to update a database.
- Technically, a database transaction is **part of a sequence of operations** and is treated as a single unit (an atomic unit).
- All of the operations in a transaction have to be completed before the database changes are made permanent. This ensures that failure of operations within a transaction does not lead to inconsistencies in the database.

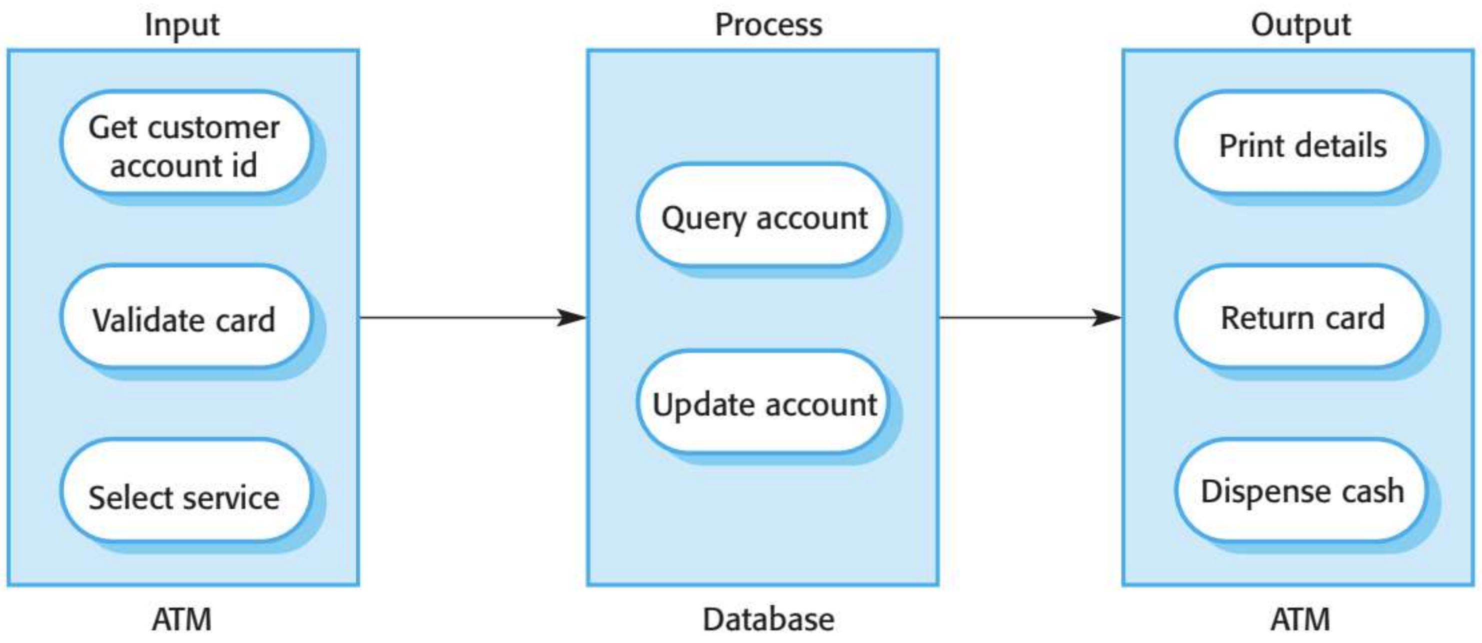
Transaction Processing Systems..cont

- An **example of a database transaction** is a customer request to withdraw money from a bank account using an ATM. This involves checking the customer account balance to see if sufficient funds are available, modifying the balance by the amount withdrawn and sending commands to the ATM to deliver the cash. Until all of these steps have been completed, the transaction is incomplete and the customer accounts database is not changed.
- Transaction processing systems may be organized as a “pipe and filter” architecture, with system components responsible for input, processing, and output.

Transaction Processing Systems..cont

- An **example of a database transaction** is a customer request to withdraw money from a bank account using an ATM. This involves checking the customer account balance to see if sufficient funds are available, modifying the balance by the amount withdrawn and sending commands to the ATM to deliver the cash. Until all of these steps have been completed, the transaction is incomplete and the customer accounts database is not changed.
- Transaction processing systems may be organized as a “pipe and filter” architecture, with system components responsible for input, processing, and output.

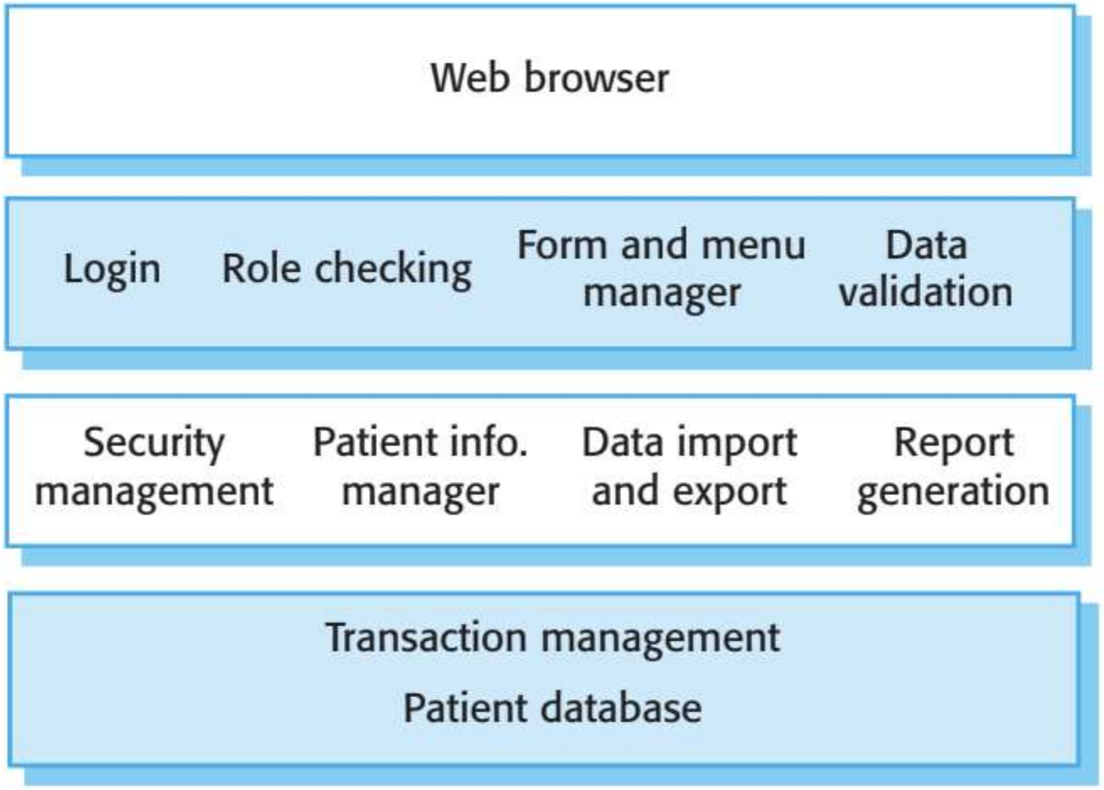
Example: Transaction Processing Systems



Information Systems

- All systems that involve interaction with a shared database can be considered to be transaction-based information systems.
- An information system allows controlled access to a large base of information, such as a library catalog, a flight timetable, or the records of patients in a hospital. Information systems are almost always web-based systems, where the user interface is implemented in a web browser.

Example: The architecture of MentCare System



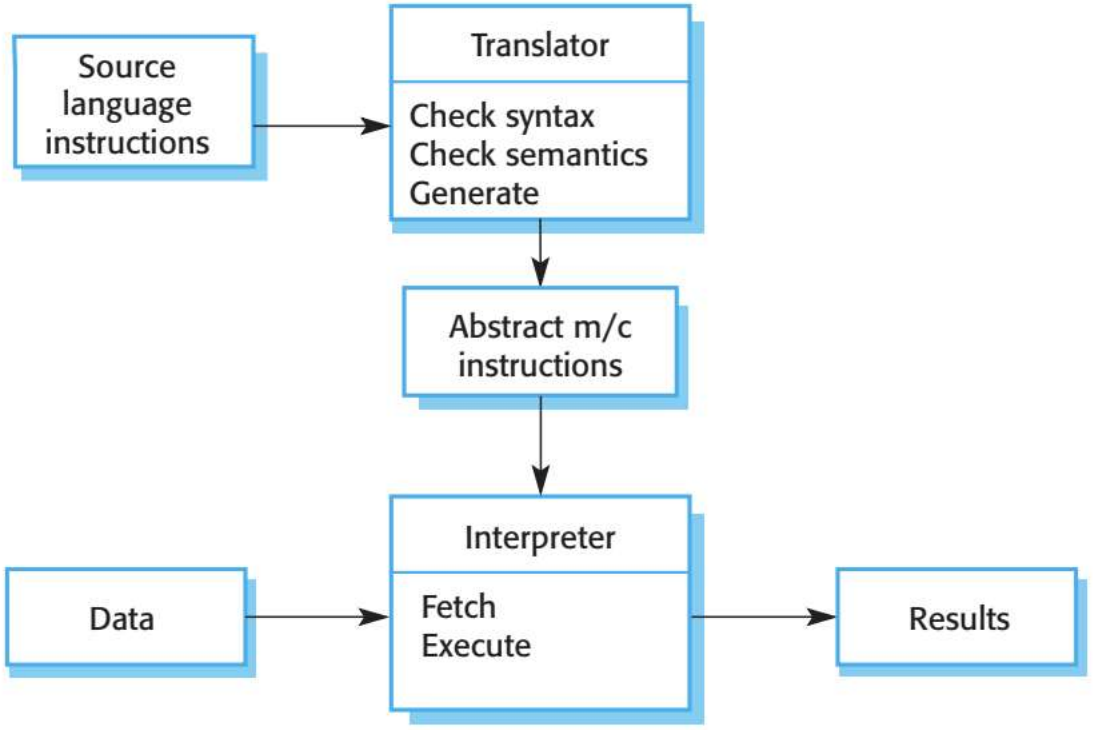
Language Processing System

- Language processing systems translate one language into an alternative representation of that language and, for programming languages, may also execute the resulting code.
- Compilers translate a programming language into machine code. Other language processing systems may translate an XML data description into commands to query a database or to an alternative XML representation.
- Natural language processing systems may translate one natural language to another, for example, French to Norwegian.

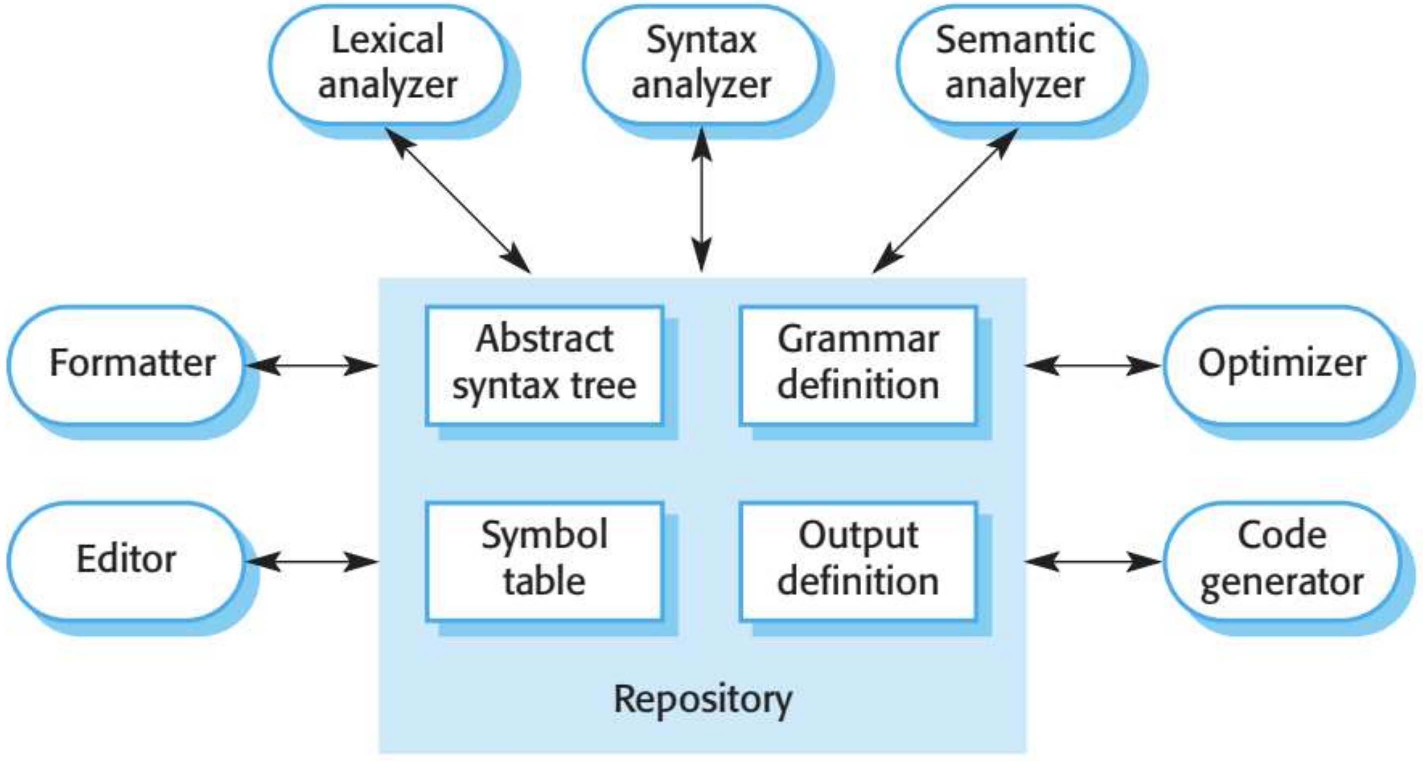
Language Processing System..cont

- Alternative Architectural patterns may be used in a language processing system (Garlan and Shaw 1993).
- Compilers can be implemented using a composite of a repository and a pipe and filter model. In a compiler architecture, the symbol table is a repository for shared data.
- This pipe and filter model of language compilation is effective in batch environments where programs are compiled and executed without user interaction; for example, in the translation of one XML document to another. It is less effective when a compiler is integrated with other language processing tools such as a structured editing system, an interactive debugger, or a program formatter.

Example: Architecture of Language Processing System



Example: Repository Architecture of Language Processing System



Example: Repository Architecture of Language Processing System

- Programming language compilers that are part of a more general programming environment have a generic architecture (shown in Slide 84) that includes the following components:
  1. **A lexical analyzer,** which takes input language tokens and converts them into an internal form.
  2. **A symbol table,** which holds information about the names of entities used in the text that is being translated.
  3. **A syntax analyzer,** which checks the syntax of the language being translated. It uses a defined grammar of the language and builds a syntax tree.
  4. **A syntax tree,** which is an internal structure representing the program being compiled.
  5. **A semantic analyzer,** which uses information from the syntax tree and the symbol table to check the semantic correctness of the input language text.
  6. **A code generator,** which “walks” the syntax tree and generates abstract machine code.

Key Points

- A software architecture is a description of how a software system is organized.
- Architectural design decisions include decisions on the type of application, the distribution of the system, the architectural styles to be used.
- Architectures may be documented from several different perspectives or views such as a conceptual view, a logical view, a process view, and a development view.
- Architectural patterns are a means of reusing knowledge about generic system architectures. They describe the architecture, explain when it may be used and describe its advantages and disadvantages.