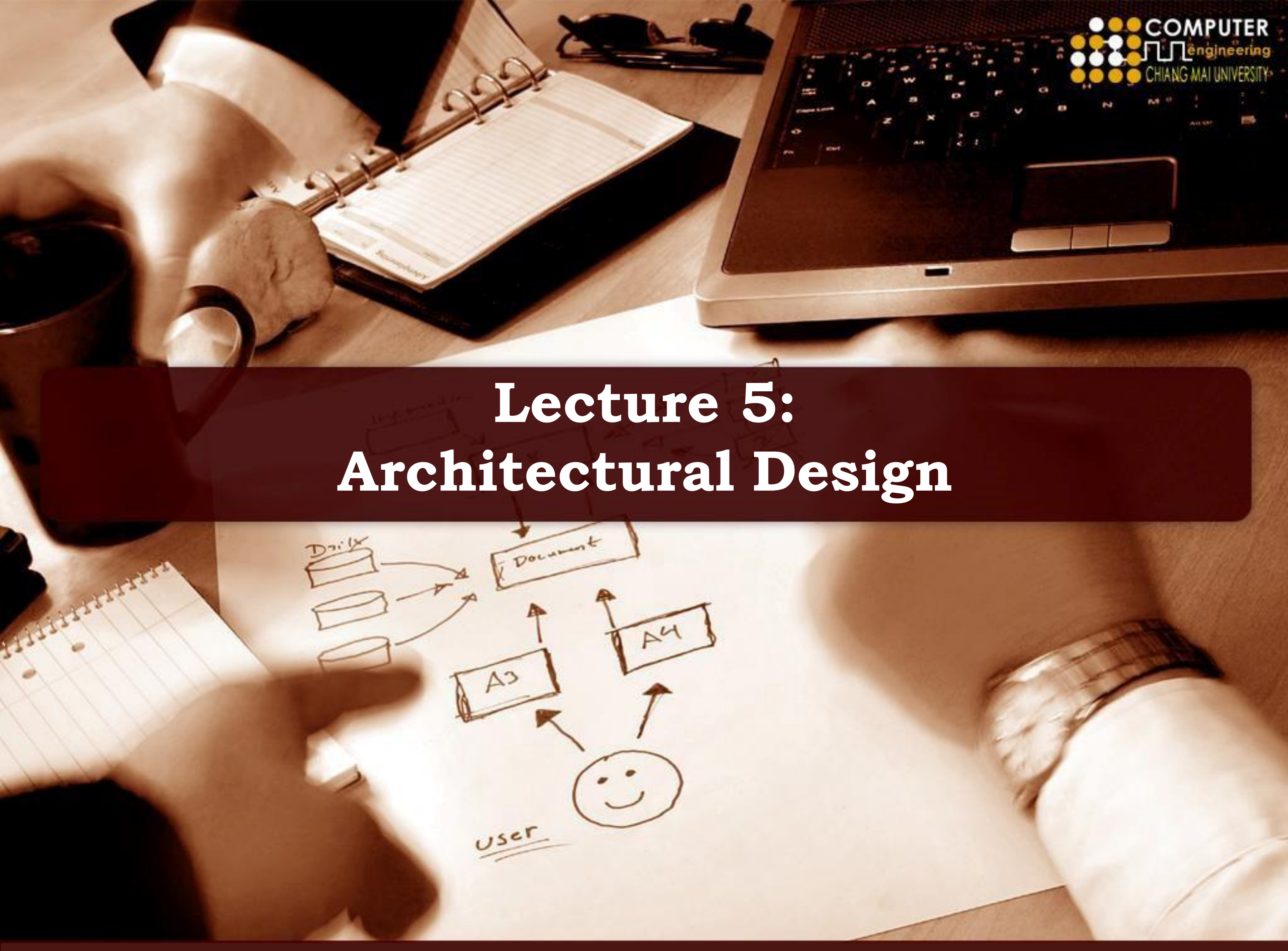


Lecture 5: Architectural Design

Class Review

- System modeling aids in representing and understanding of the system.
- Requirements specification may be overlapped with Designing activities.
- Designing starts from Architectural design which involves identifying sub-systems and its framework for controlling and communication.
- Software architecture is the output of the Architectural design process.

What is Software Design?

- Definition by ISO/IEC/IEEE 12207-2:2020(E)
- Activities
  - UX Design (Customer Journey Map: CJM)
  - Architectural design specification (Software Architecture Document: SAD)
  - Abstract specification (ASD)
  - Interface design specification (ISD)
  - Component design specification (CSD)
  - Data structure design specification (DSDS)
  - Algorithm design specification (ADS)

Definitions

- “Software design is usually concurrent with software implementation, integration, verification, and validation.” – 12207
- “The software design strategy can include initial or incremental decomposition into system elements; creation of various views of automated procedures, data structures and control systems; selection of design patterns, or progressively more detailed definition of objects and their relationships.” -- 12207
- See Annex H for Application of Agile to software devesign.

Software Architecture

- Software Architecture is a high-level design of an overall structure of a software system showing main structural components and relationships between them
- Concerns with organization of a system
- Often overlapped with Requirements Engineering
- The systems can comprise more than one structure.

Definition

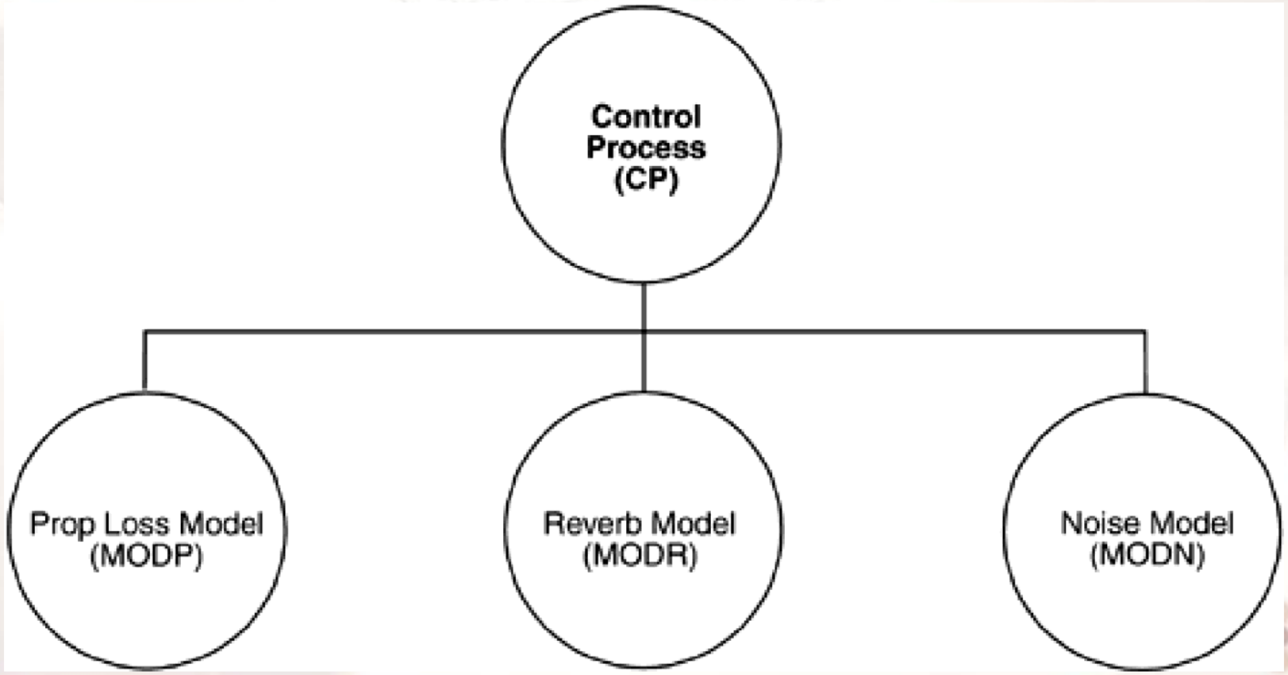
- As many definitions do, architecture is a set of components and their connections.
- Software architecture definition is much more precise.

"The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them."[1]

1. Software Architecture in Practice, 2nd Edition.

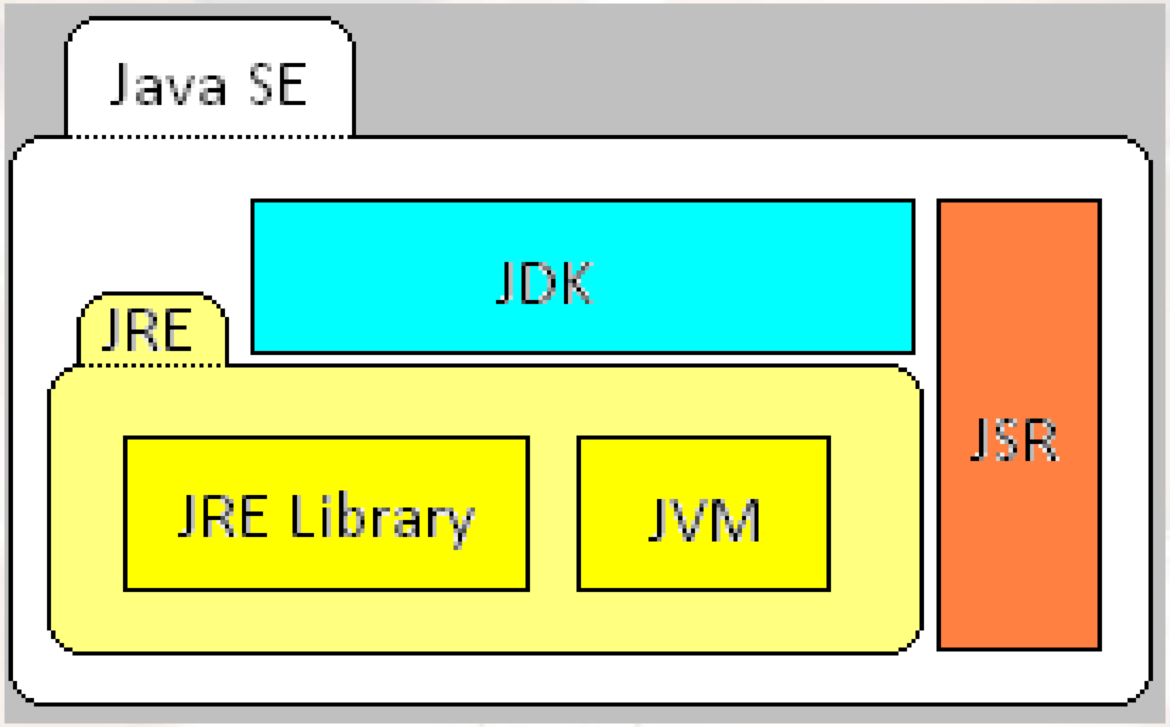
An Example

- Underwater acoustic simulation
- Informative or uninformative?
- What can we tell from it?
- What can we *not* tell from the diagram?



Another Example

- JavaSE Software Architecture

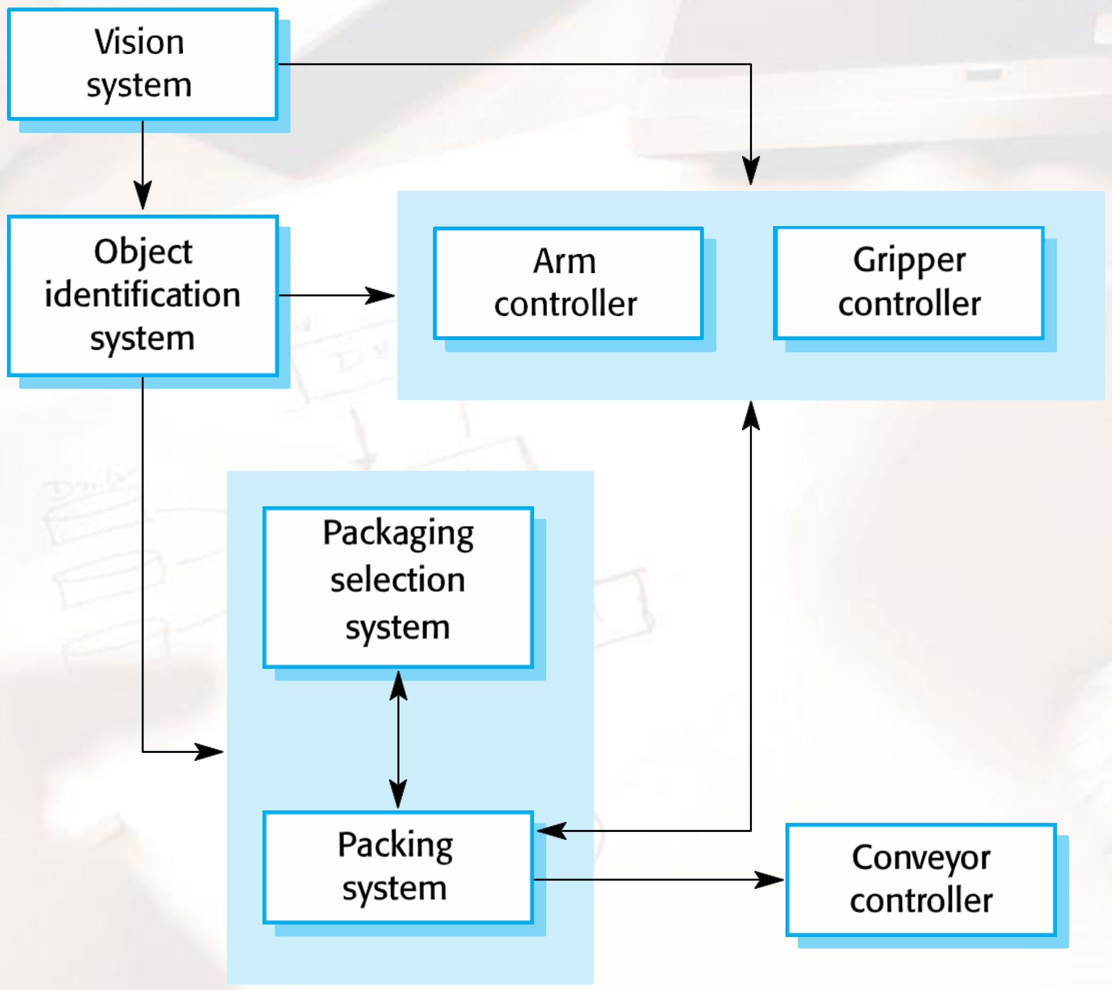


- Image taken from https://commons.wikimedia.org/wiki/File:GDK_software_architecture.svg

Agile and Architecture

- An overall systems architecture is generally designed at an early stage of agile processes.
- Refactoring the system architecture is impractical as it affects many components in the system

Packing Robot Control System: Architecture



Small vs. Large Architecture

- Small
  - Individual programs
  - decomposed into components.
- Large
  - Complex enterprise systems distributed over different computers, which may be owned and managed by different companies.

Architectural design decisions

- Is there a generic application architecture that can be used?
- How will the system be distributed?
- What architectural styles are appropriate?
- What approach will be used to structure the system?
- How will the system be decomposed into modules?
- What control strategy should be used?
- How will the architectural design be evaluated?
- How should the architecture be documented?

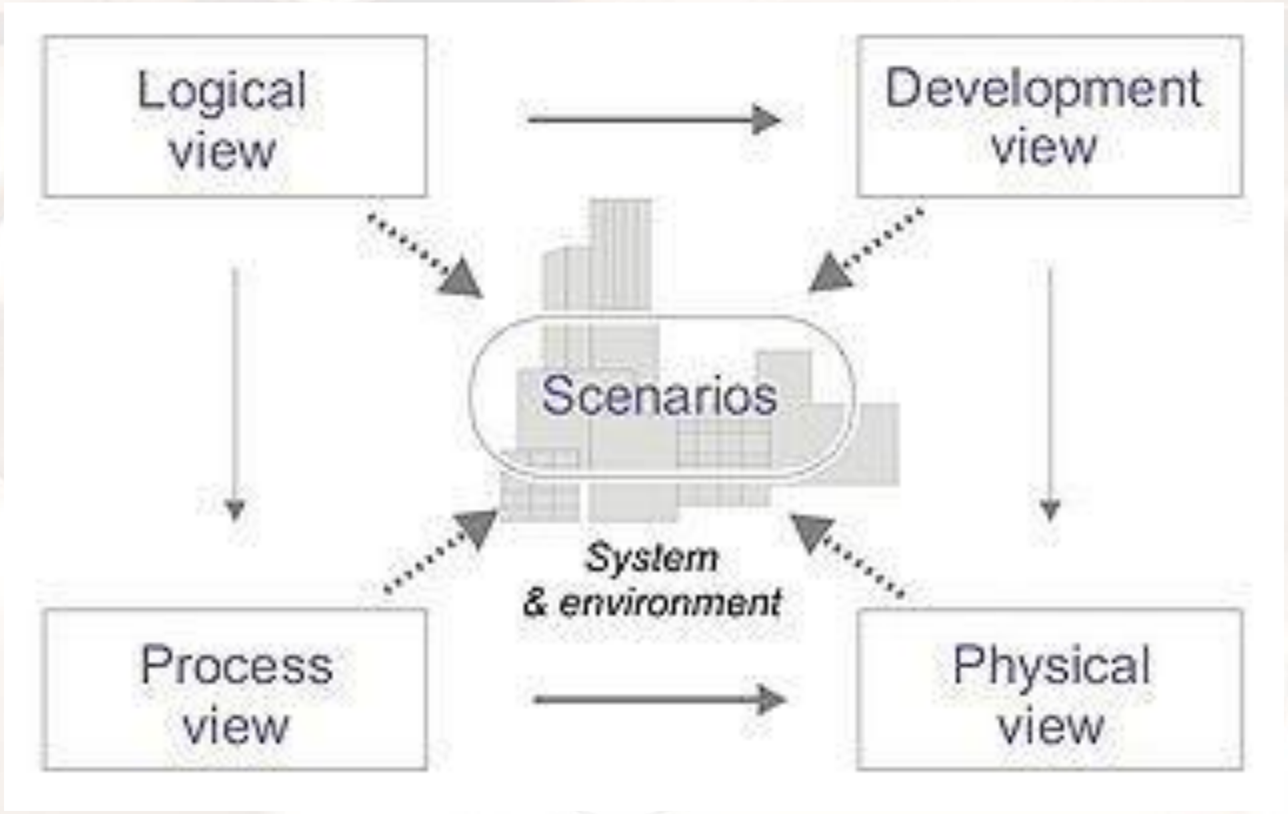
Architecture reuse

- 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'.
  - These capture the essence of an architecture and can be instantiated in different ways.
  - Discussed later in this lecture.

Software Architecture Views

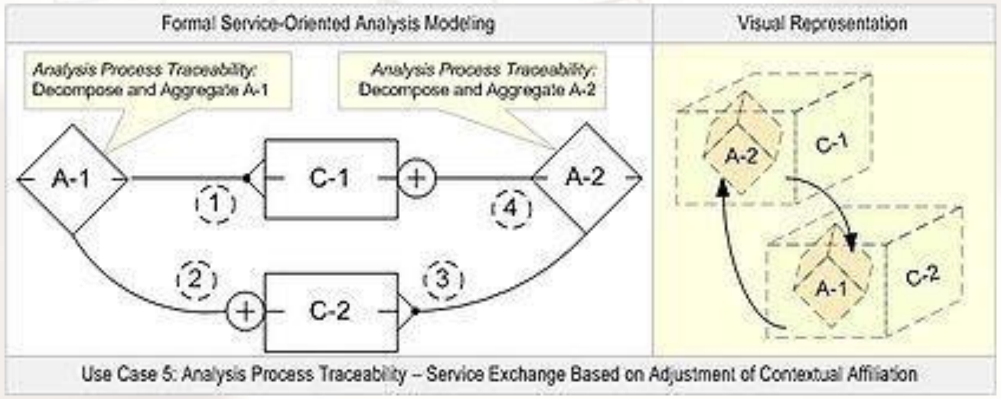
- Software architecture is commonly organized in views.
- 4+1 view model designed by Philippe Kruchten
  - Logical view (Function)
  - Process view (Concurrency/Thread view)
  - Development (Structure/Decomposition)
  - Physical (Deployment)
  - User action (Use case)
- Other views
  - Code/Module view
  - Data view

4+1


Logical view
Development view
Scenarios
System & environment
Process view
Physical view

Languages/Notations

- UML
- Service-Oriented Modeling (SOM)
  - Design and specify service-oriented business systems within a service-oriented architecture (SOA).
  - Service-oriented analysis and design (SOMA)
  - Service-oriented modeling framework (SOMF)


Formal Service-Oriented Analysis Modeling
Visual Representation
Analysis Process Traceability: Decompose and Aggregate A-1
Analysis Process Traceability: Decompose and Aggregate A-2
A-1
C-1
A-2
C-2
1
2
3
4
Use Case 5: Analysis Process Traceability – Service Exchange Based on Adjustment of Contextual Affiliation

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

Architectural Patterns (Styles)

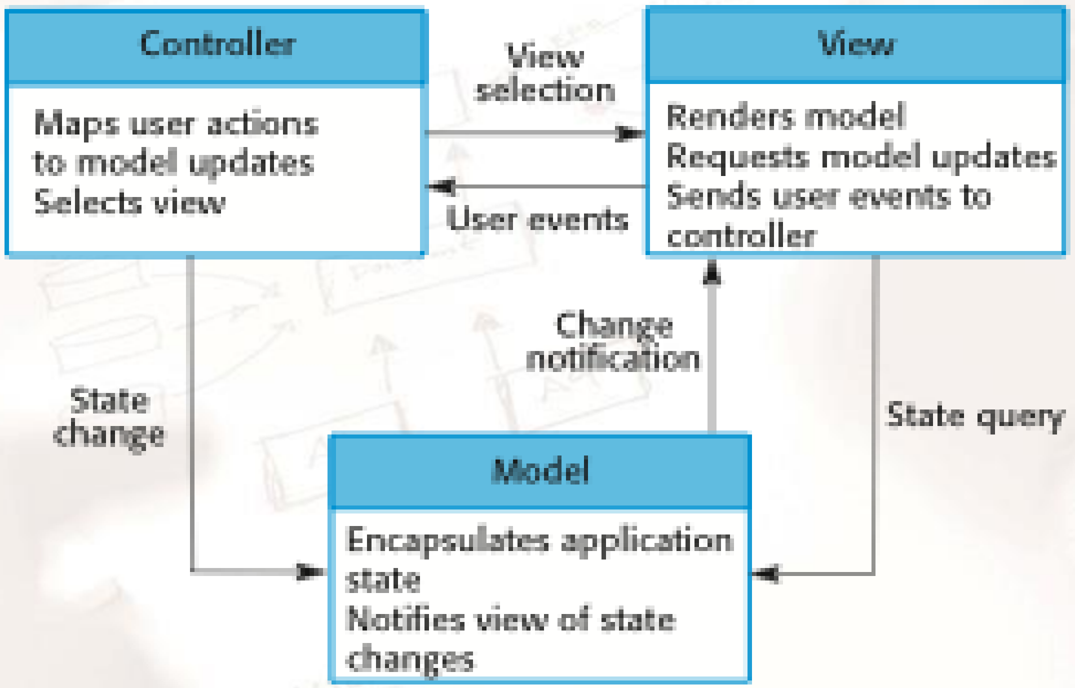
- An architectural style/pattern is a stylized description of good design practice, which has been tried and tested in different environments.
- Patterns may be represented in tabular and/or graphical forms.
- Patterns should provide information about when they are and when the are not useful.

Architectural Patterns

- Model-view-controller (MVC)
- Layered architecture
- The Repository pattern
- Client-server architecture
- Database-centric architecture
- Pipe and filter architecture
- Application architectures
- Information systems architecture
- Service-oriented architecture (SOA)

MVC

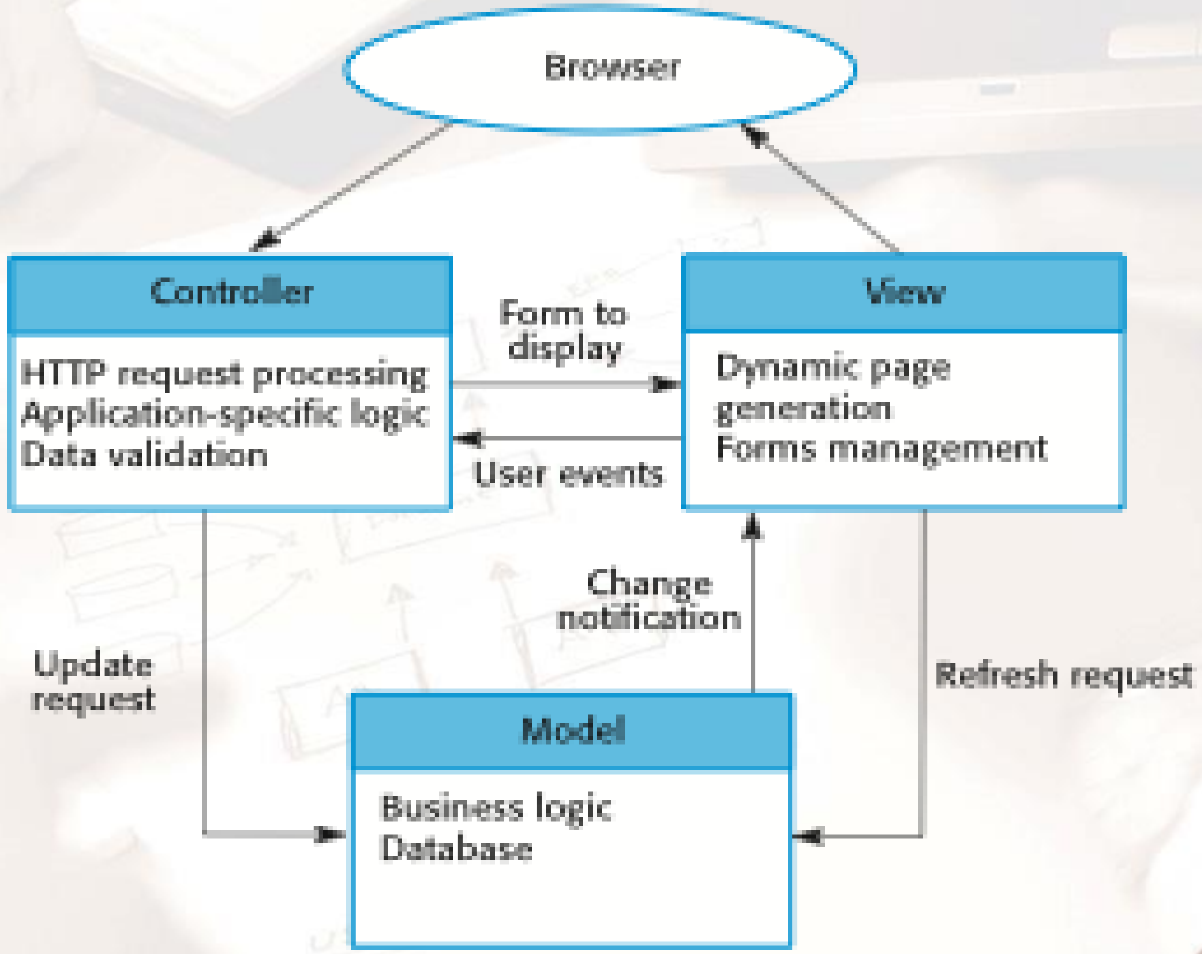
- Separate Logic from Input and presentation (GUI) to permit independent development and testing of each.



MVC

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. See Figure 6.3.
Example	Figure 6.4 shows the architecture of a web-based application system organized using the MVC pattern.
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.

Web Application Architecture using MVC



Some MVC Frameworks

- GUI
  - Cocoa
  - Microsoft Foundation Class Library (MFC)
- Web-based
  - Monorail (.NET)
  - CakePHP

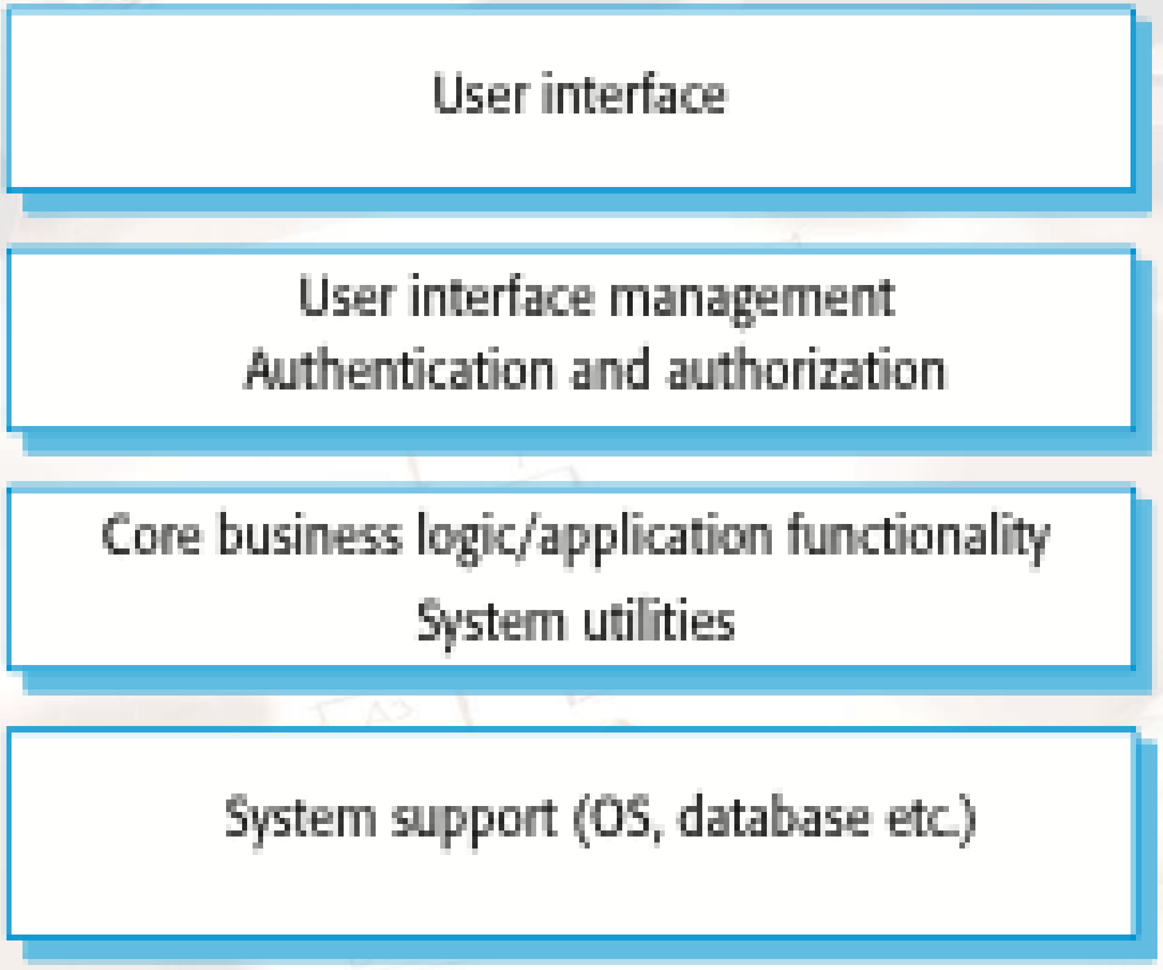
Layered architecture

- Used to model the interfacing of sub-systems.
- Organises the system into a set of layers. Each provides 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.

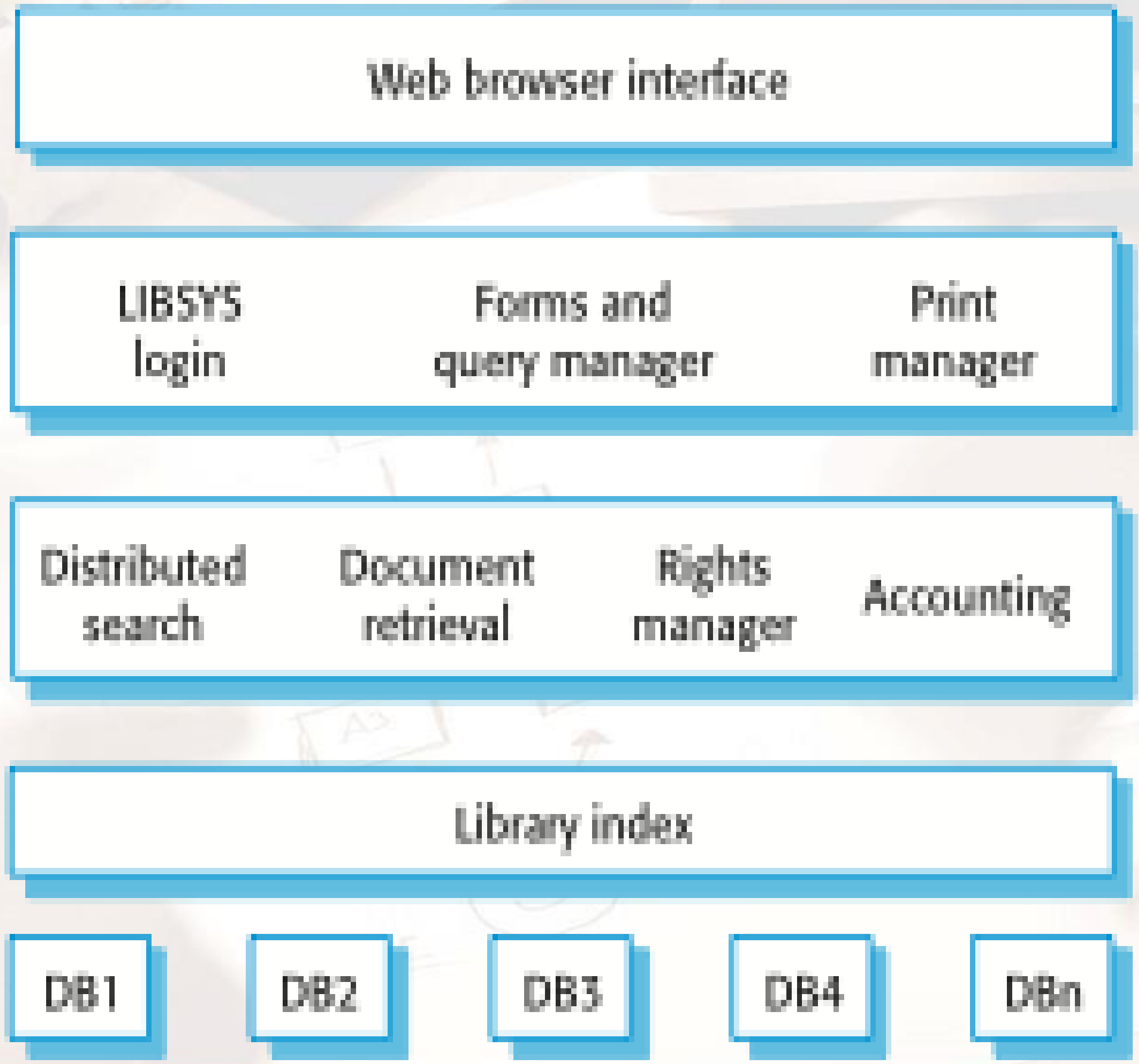
The Layered architecture pattern

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 Figure 6.6.
Example	A layered model of a system for sharing copyright documents held in different libraries, as shown in Figure 6.7.
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.

A generic layered architecture


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

The architecture of the LIBSYS system



Repository architecture

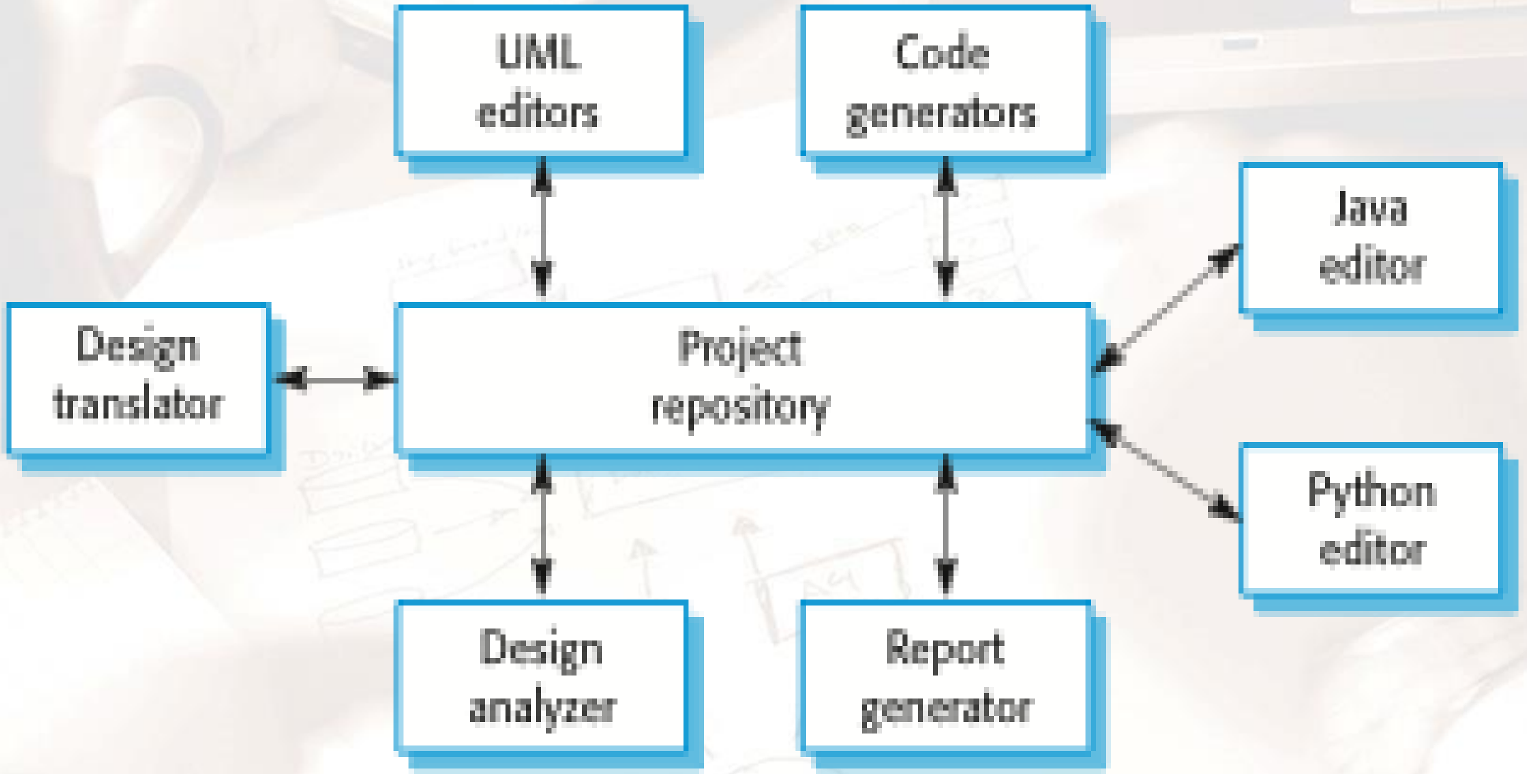
- Sub-systems must exchange data. This may be done in two ways:
  - Shared data is held in a central database or repository and may be accessed by all sub-systems;
  - Each sub-system maintains its own database and passes data explicitly to other sub-systems.
- When large amounts of data are to be shared,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

COMPUTER engineering CHIANG MAI UNIVERSITY

The 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	Figure 6.9 is an example of an IDE 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	You should use this pattern when you have a system in which large volumes of information are generated that has to be stored for a long time. You 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.

A repository architecture for an IDE



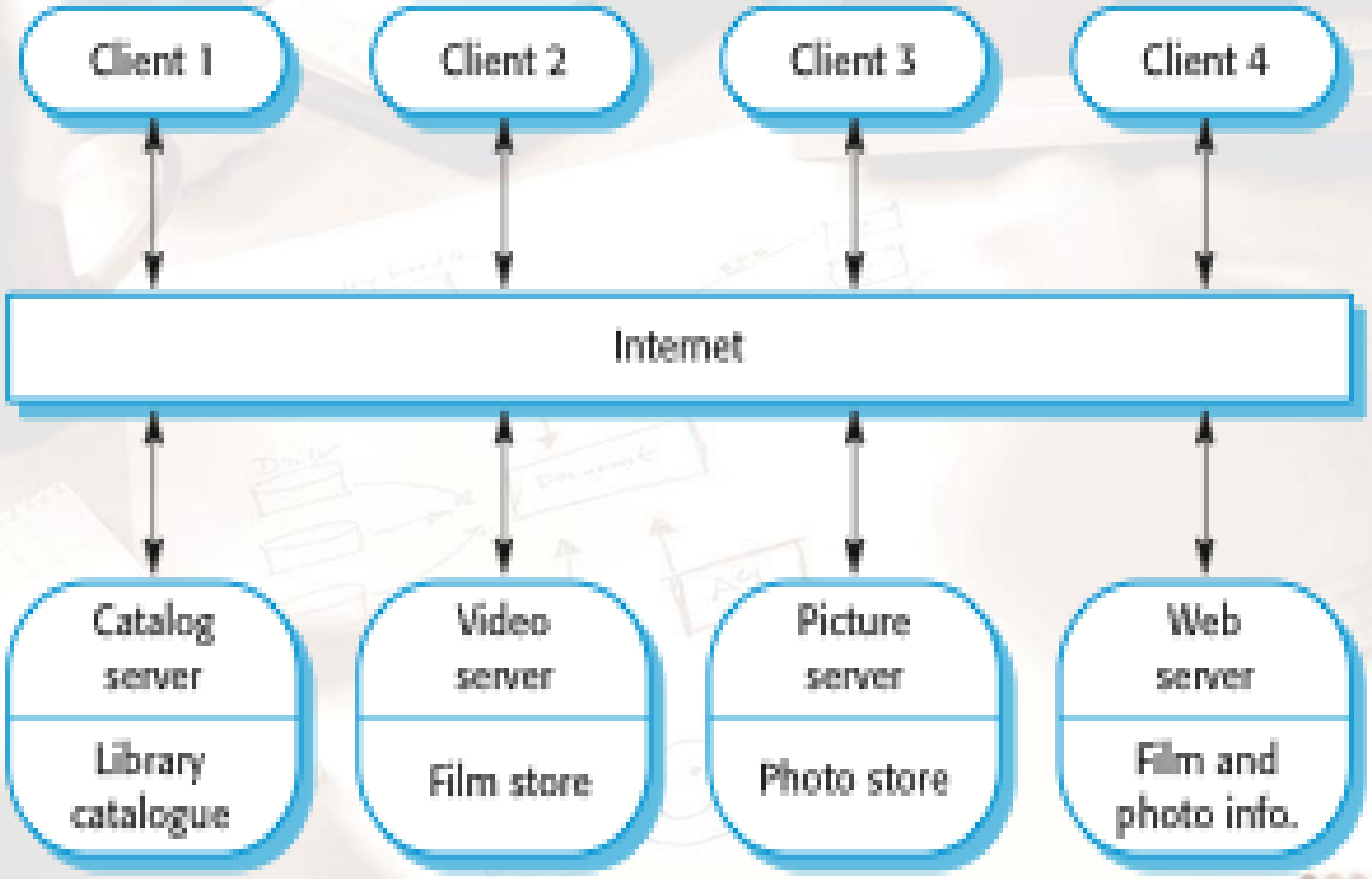
Client-server architecture

- 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.

The 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	Figure 6.11 is an example of a film and video/DVD library organized as a client–server system.
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.

A client–server architecture for a film library



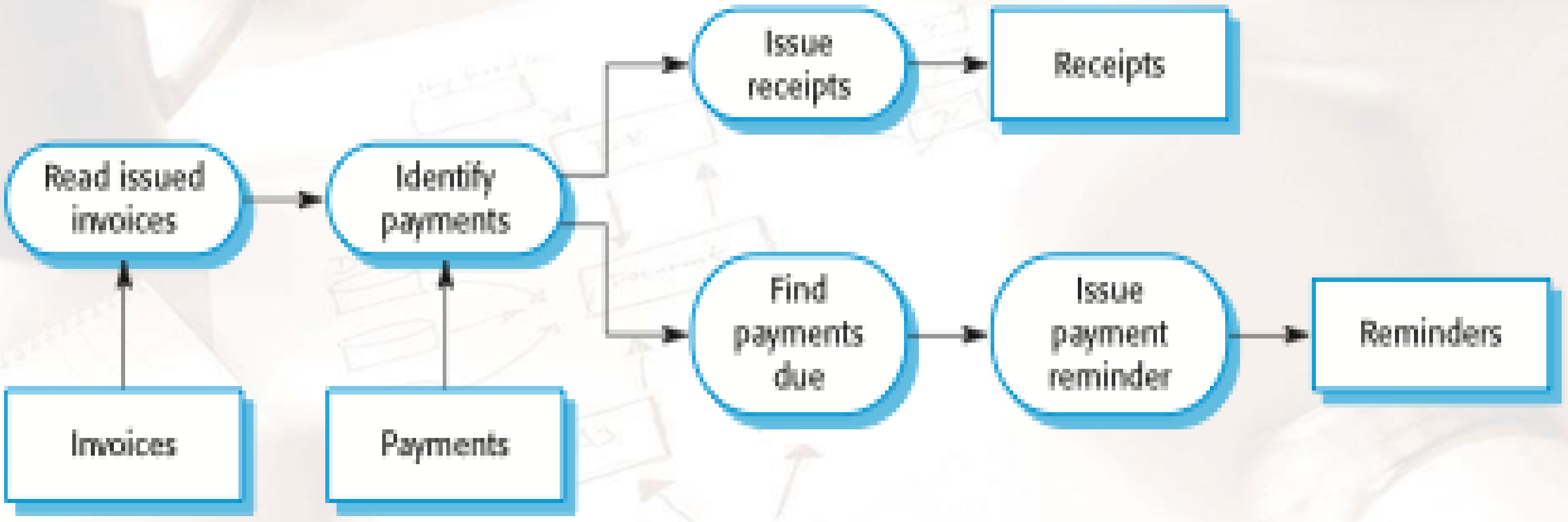
Pipe and filter architecture

- 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.

The 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	Figure 6.13 is an example of a pipe and filter system used for processing invoices.
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 agreAed 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



Application architectures

- Application systems are designed to meet an organisational need.
- As businesses have much in common, their application systems also tend to have a common architecture that reflects the application requirements.
- A generic application architecture is an architecture for a type of software system that may be configured and adapted to create a system that meets specific requirements.

Use of application architectures

- As a starting point for architectural design.
- As a design checklist.
- As a way of organising the work of the development team.
- As a means of assessing components for reuse.
- As a vocabulary for talking about application types.

Examples of application types

- Data processing applications
  - Data driven applications that process data in batches without explicit user intervention during the processing.
- Transaction processing applications
  - Data-centred applications that process user requests and update information in a system database.
- Event processing systems
  - Applications where system actions depend on interpreting events from the system's environment.
- Language processing systems
  - Applications where the users' intentions are specified in a formal language that is processed and interpreted by the system.

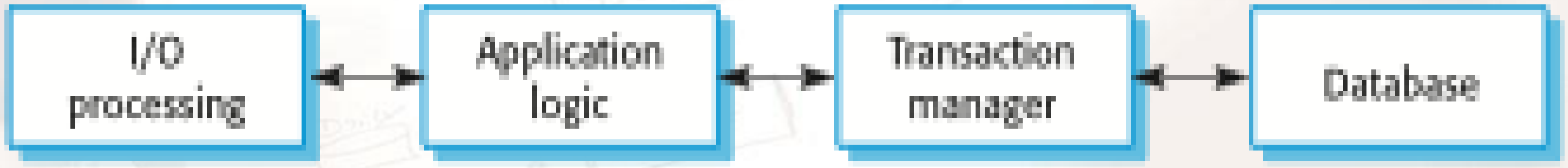
Application type examples

- Focus here is on transaction processing and language processing systems.
- Transaction processing systems
  - E-commerce systems;
  - Reservation systems.
- Language processing systems
  - Compilers;
  - Command interpreters.

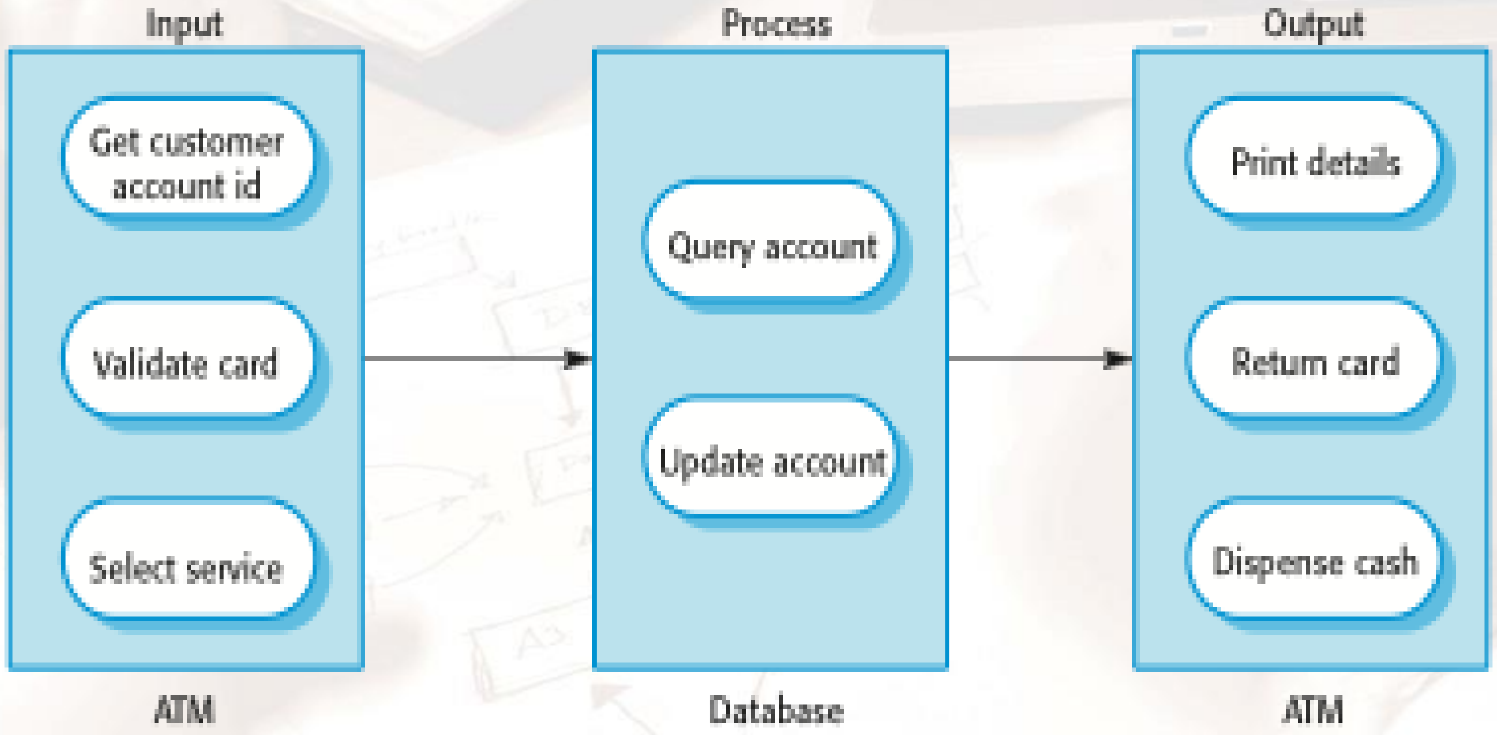
Transaction processing systems

- Process user requests for information from a database or requests to update the database.
- From a user perspective a transaction is:
  - Any coherent sequence of operations that satisfies a goal;
  - For example - find the times of flights from London to Paris.
- Users make asynchronous requests for service which are then processed by a transaction manager.

The structure of transaction processing applications



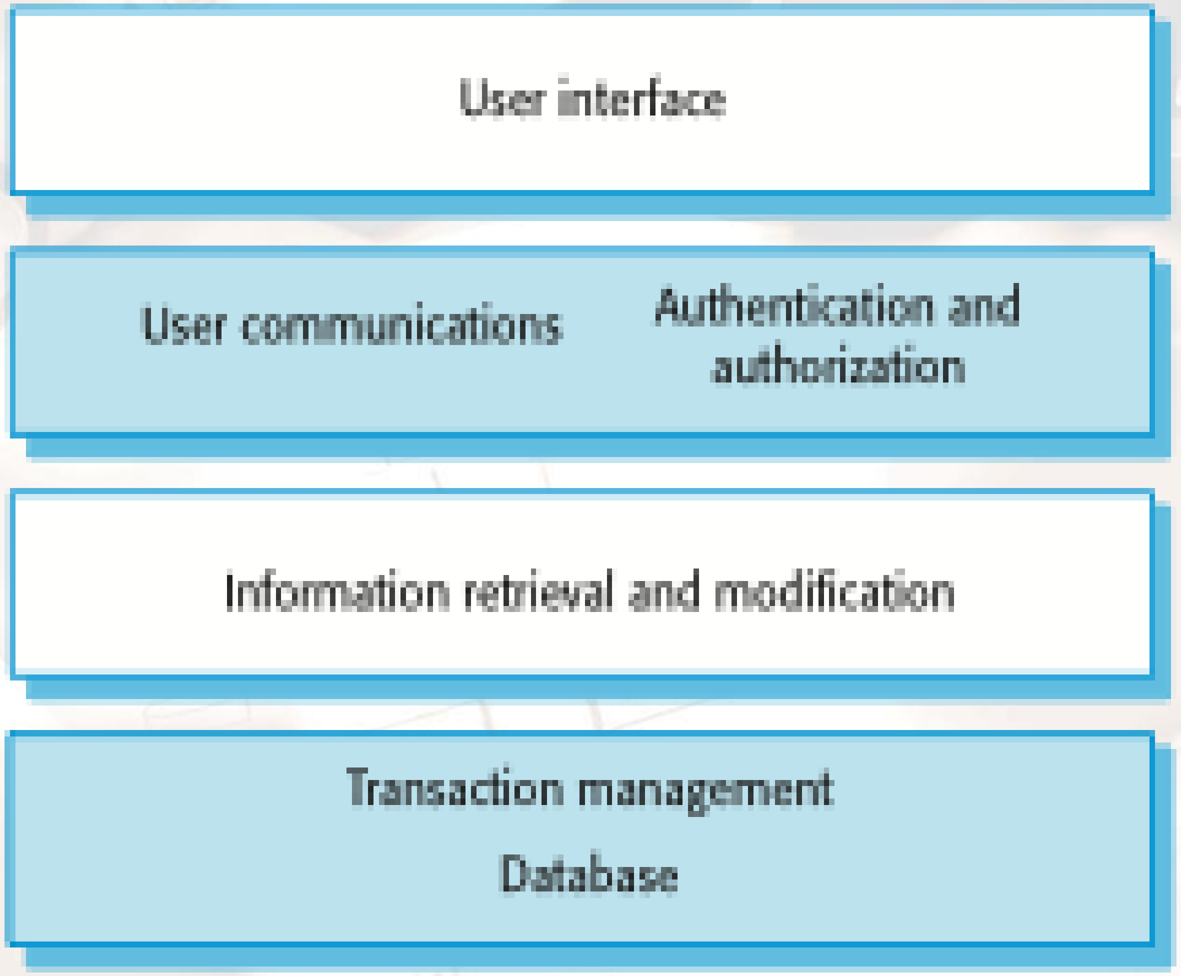
The software architecture of an ATM system



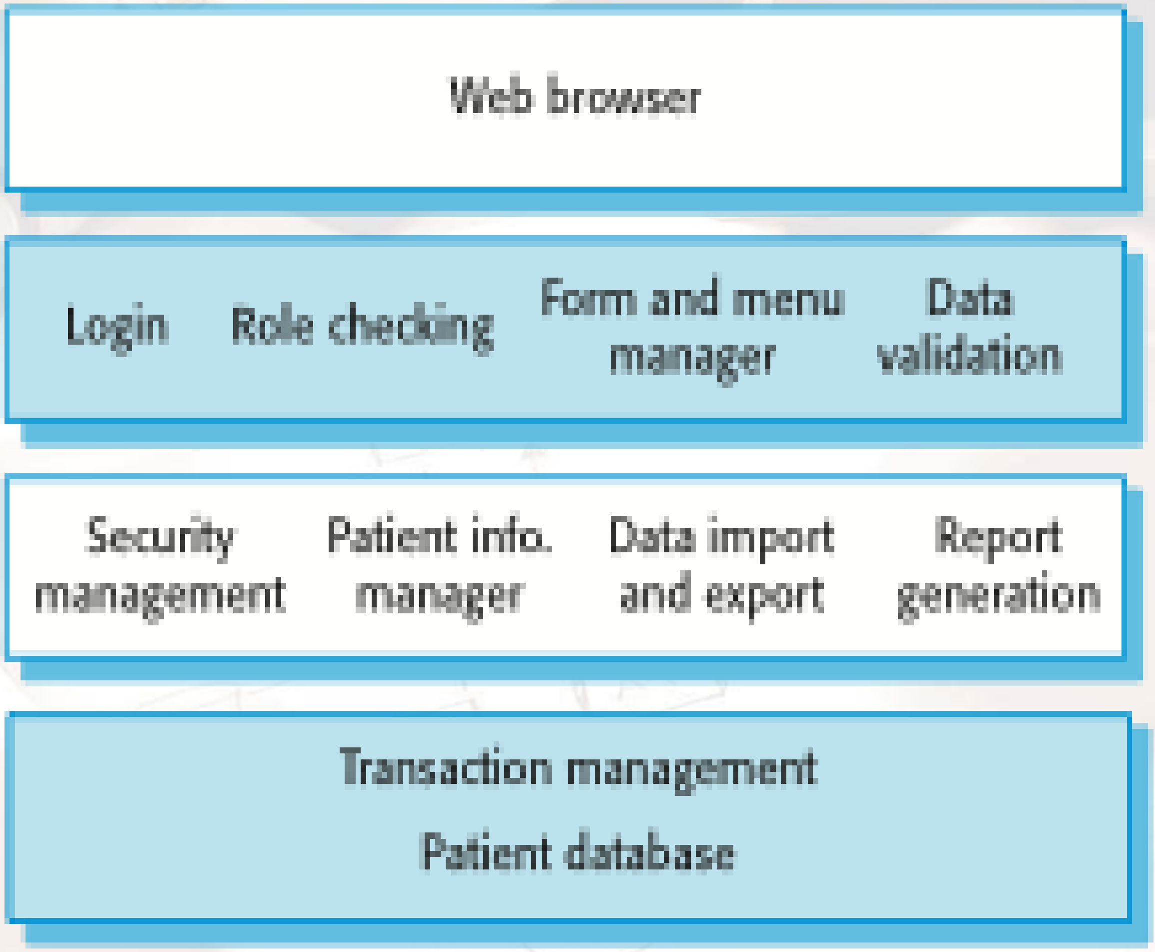
Information systems architecture

- Information systems have a generic architecture that can be organised as a layered architecture.
- These are transaction-based systems as interaction with these systems generally involves database transactions.
- Layers include:
  - The user interface
  - User communications
  - Information retrieval
  - System database

Layered information system architecture


User interface
User communications
Authentication and authorization
Information retrieval and modification
Transaction management
Database

The architecture of the MHC-PMS



Web-based information systems

- Information and resource management systems are now usually web-based systems where the user interfaces are implemented using a web browser.
- For example, e-commerce systems are Internet-based resource management systems that accept electronic orders for goods or services and then arrange delivery of these goods or services to the customer.
- In an e-commerce system, the application-specific layer includes additional functionality supporting a ‘shopping cart’ in which users can place a number of items in separate transactions, then pay for them all together in a single transaction.

Server implementation

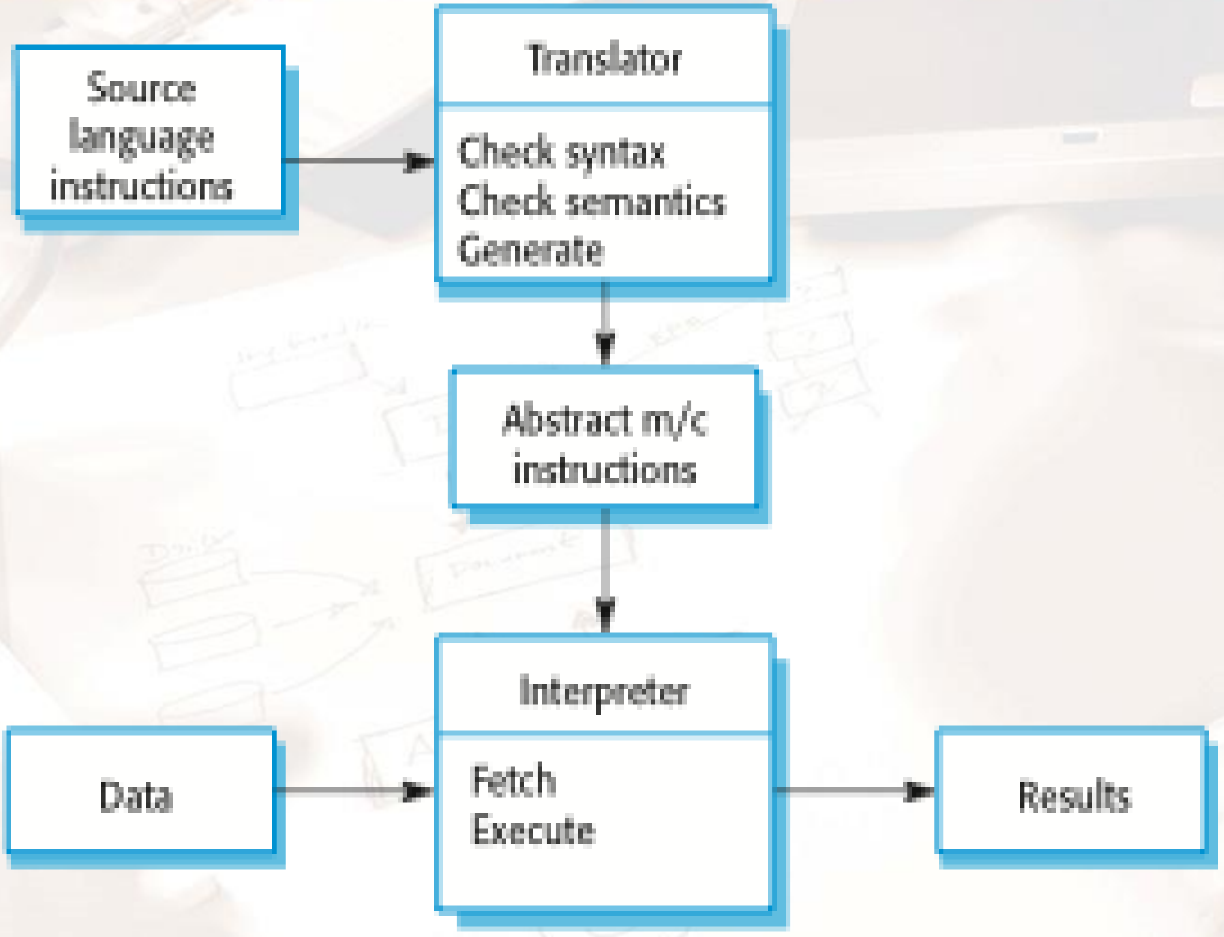
- These systems are often implemented as multi-tier client server/architectures (discussed in Chapter 18)
  - The web server is responsible for all user communications, with the user interface implemented using a web browser;
  - The application server is responsible for implementing application-specific logic as well as information storage and retrieval requests;
  - The database server moves information to and from the database and handles transaction management.

Language processing systems

- Accept a natural or artificial language as input and generate some other representation of that language.
- May include an interpreter to act on the instructions in the language that is being processed.
- Used in situations where the easiest way to solve a problem is to describe an algorithm or describe the system data
  - Meta-case tools process tool descriptions, method rules, etc and generate tools.

COMPUTER engineering CHIANG MAI UNIVERSITY

The architecture of a language processing system



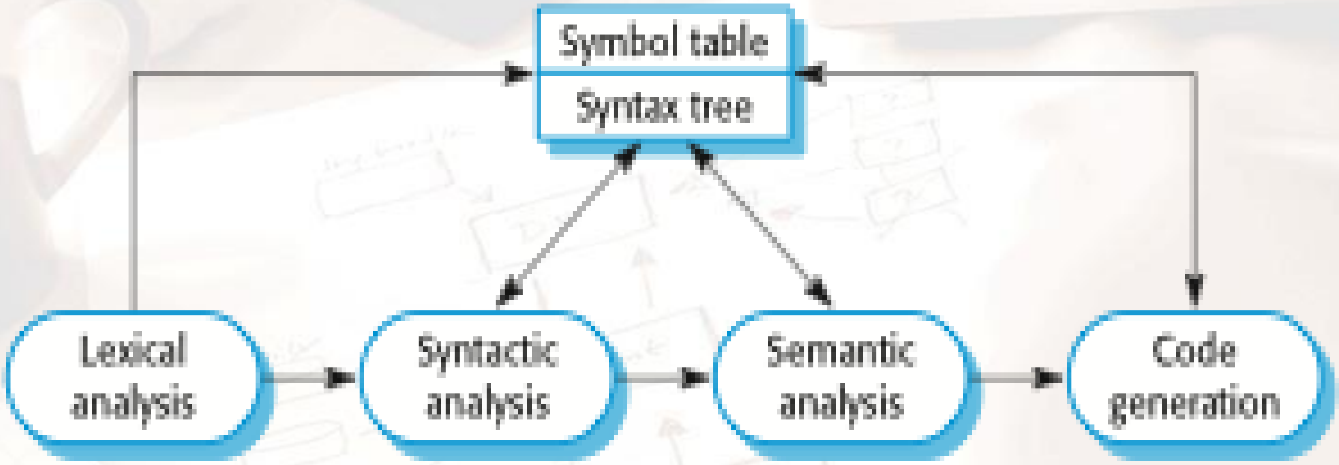
Compiler components

- A lexical analyzer, which takes input language tokens and converts them to an internal form.
- A symbol table, which holds information about the names of entities (variables, class names, object names, etc.) used in the text that is being translated.
- A syntax analyzer, which checks the syntax of the language being translated.
- A syntax tree, which is an internal structure representing the program being compiled.

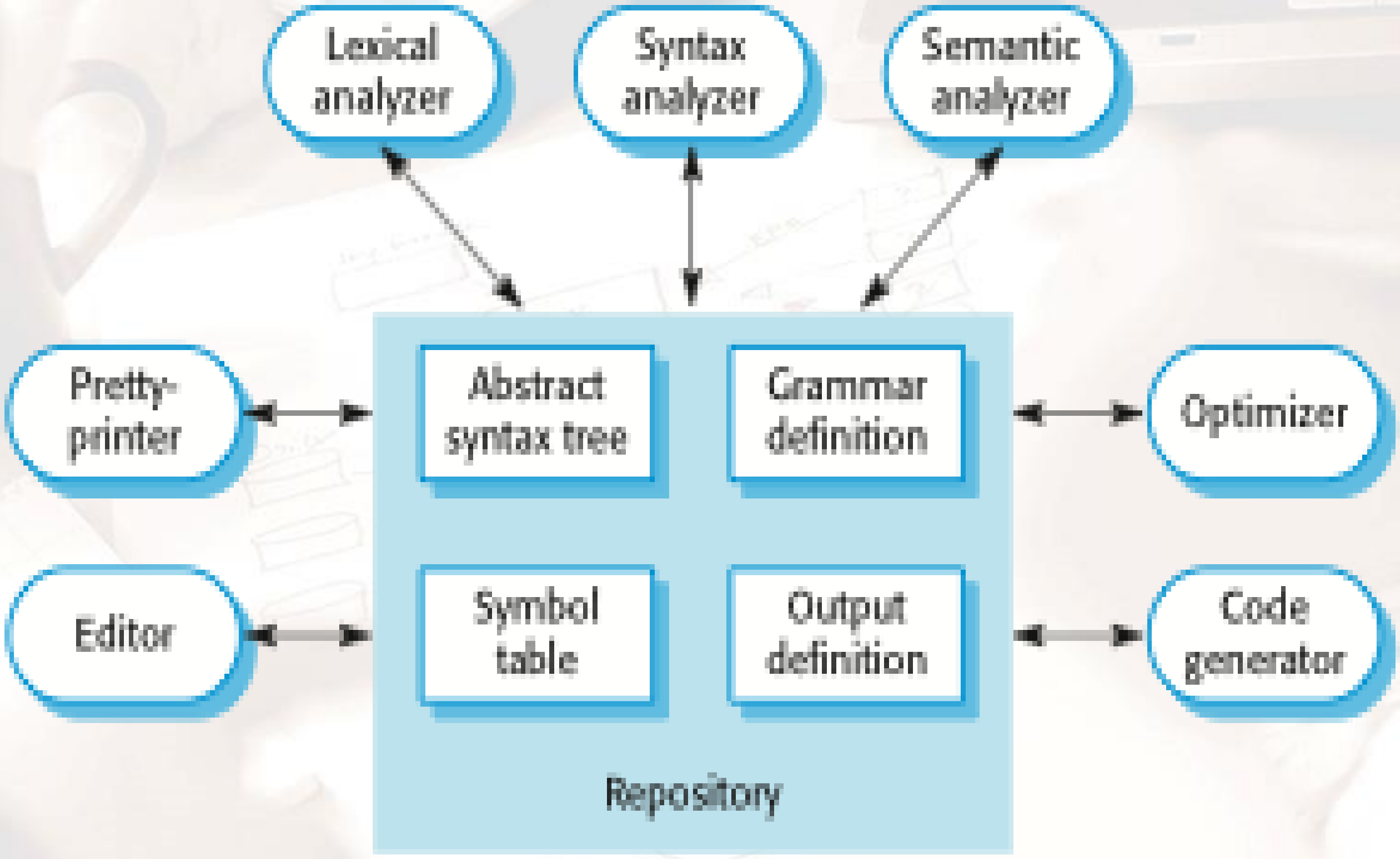
Compiler components

- A semantic analyzer that uses information from the syntax tree and the symbol table to check the semantic correctness of the input language text.
- A code generator that 'walks' the syntax tree and generates abstract machine code.

A pipe and filter compiler architecture



A repository architecture for a language processing system



Summary

- A software architecture should represent the overview of system’s logic, process, development and deployment.
- Reusing architectural patterns is encouraged.

End of Lecture Questions

1. What is an architectural design?
2. What is 4+1 views model?
3. Name a few architectural patterns.
4. Why bother desiging software architecture?
5. Do software architectural patterns speed up software development? How?

References

1. What is?
   - NNGroup.com, “When and How to Create Customer Journey Maps”
2. Process
   - Heekyung Moon, Sung H. Han, Jaemin Chun, and Sang W. Hong , “A Design Process for a Customer Journey Map: A Case Study on Mobile Services”
3. Examples
   - Reforge.com, “Explore real examples of User Journey Maps”