CSc 131 – Computer Software Engineering

Design characteristics and metrics

Detailed Design (Left over)

Detailed Design Outline:

III. Design Characteristics and Metrics
    1. McCabe's Cyclomatic Complexity
    2. Coupling and Cohesion

Design characteristics and metrics

- Legacy Characteristics of Design Attributes
  - Programming and programming modules were considered the most important artifacts.
  - Metrics and characteristics focused on the code (and very detailed design, if any).

- More Current Good Design Attributes
  - Design diagrams/models are considered the important design artifacts now.
  - Simplicity is the main design goal now (simplify a complex system into smaller pieces, etc.)
  - How do we measure simplicity?

McCabe's Cyclomatic Complexity

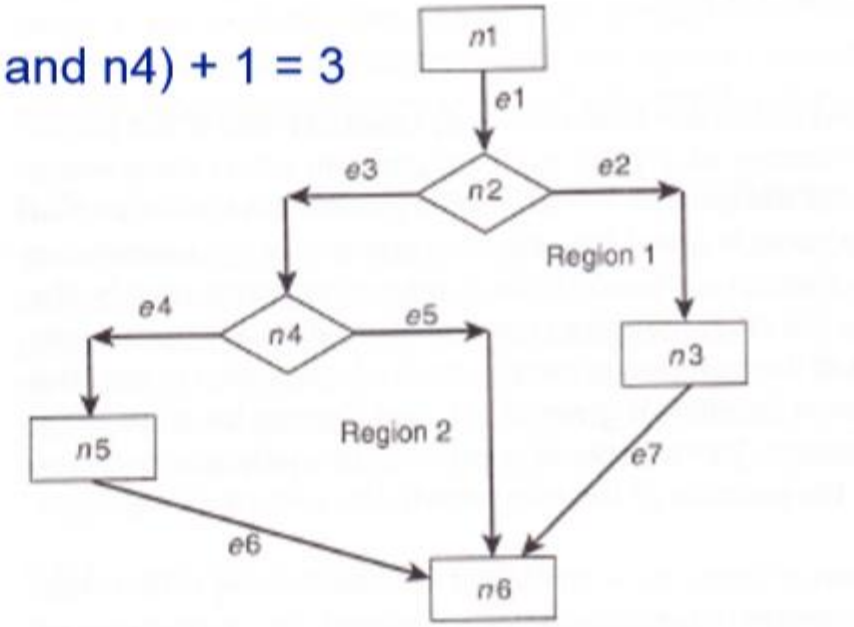
- Basic idea: program quality is directly related to the complexity of the control flow (branching)

- Computed from a control flow diagram
  - Cyclomatic complexity = E - N + 2p
  - E = number of edges of the graph
  - N = number of nodes of the graph
  - p = number of connected components (usually 1)

- Alternate computations:
  - number of binary decision + 1
  - number of closed regions +1

McCabe's Cyclomatic Complexity Example

- Using the different computations:
  - 7 edges - 6 nodes + 2*1 = 3
  - 2 regions + 1 = 3
  - 2 binary decisions (n2 and n4) + 1 = 3



McCabe's Cyclomatic Complexity

- What does the number mean?

- It's the maximum number of linearly independent paths through the flow diagram - used to determine the number of test cases needed to cover each path through the system

- The higher the number, the more risk exists (and more testing is needed)
  - 1-10 is considered low risk
  - greater than 50 is considered high risk

Characteristics of Good Design

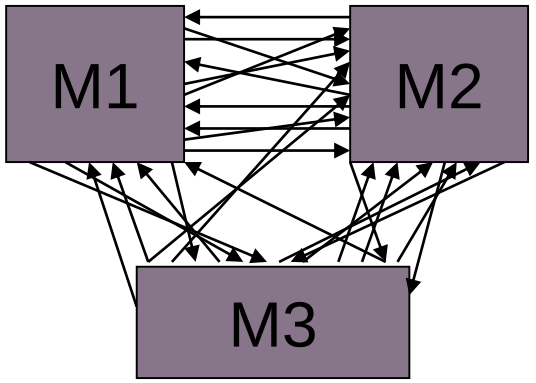
Component independence

- Coupling (goal: loose coupling)
- Cohesion (goal: strong cohesion)

Exception identification and handling

Fault prevention and fault tolerance

Design for change



Coupling

• Coupling is an attribute that specifies the number of dependencies between two software units.

- It measures the dependencies between two subsystems.

• If two subsystems are loosely coupled, they are relatively independent

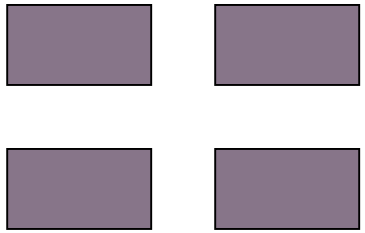
- Modifications to one of the subsystems will have little impact on the other.

• If two subsystems are strongly coupled, modifications to one subsystem is likely to have impact on the other.

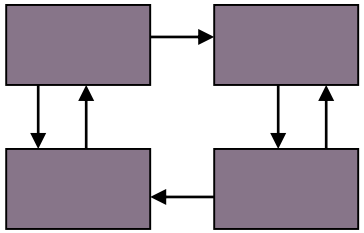
• Goal: subsystems should be as loosely coupled as is reasonable.

Coupling

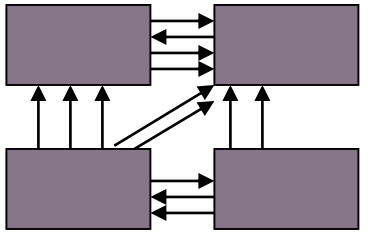
The degree of dependence such as the amount of interactions among components



No dependencies



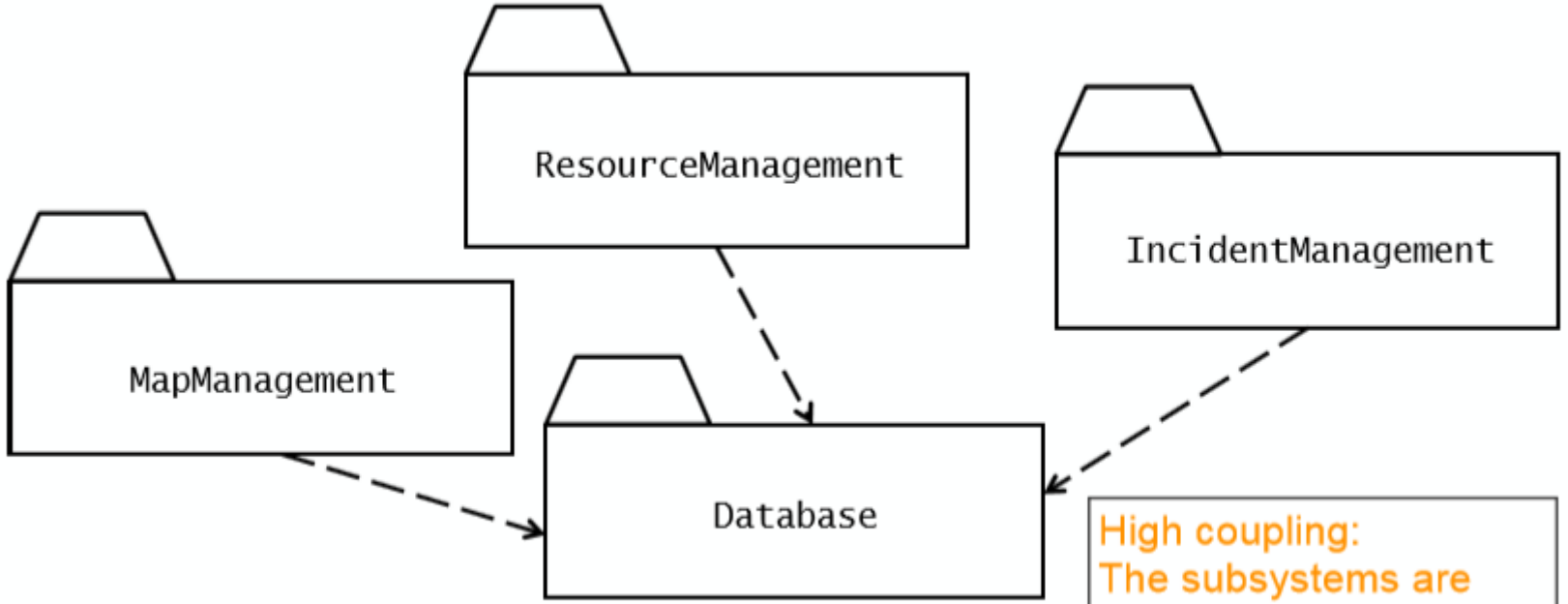
Loosely coupled
some dependencies



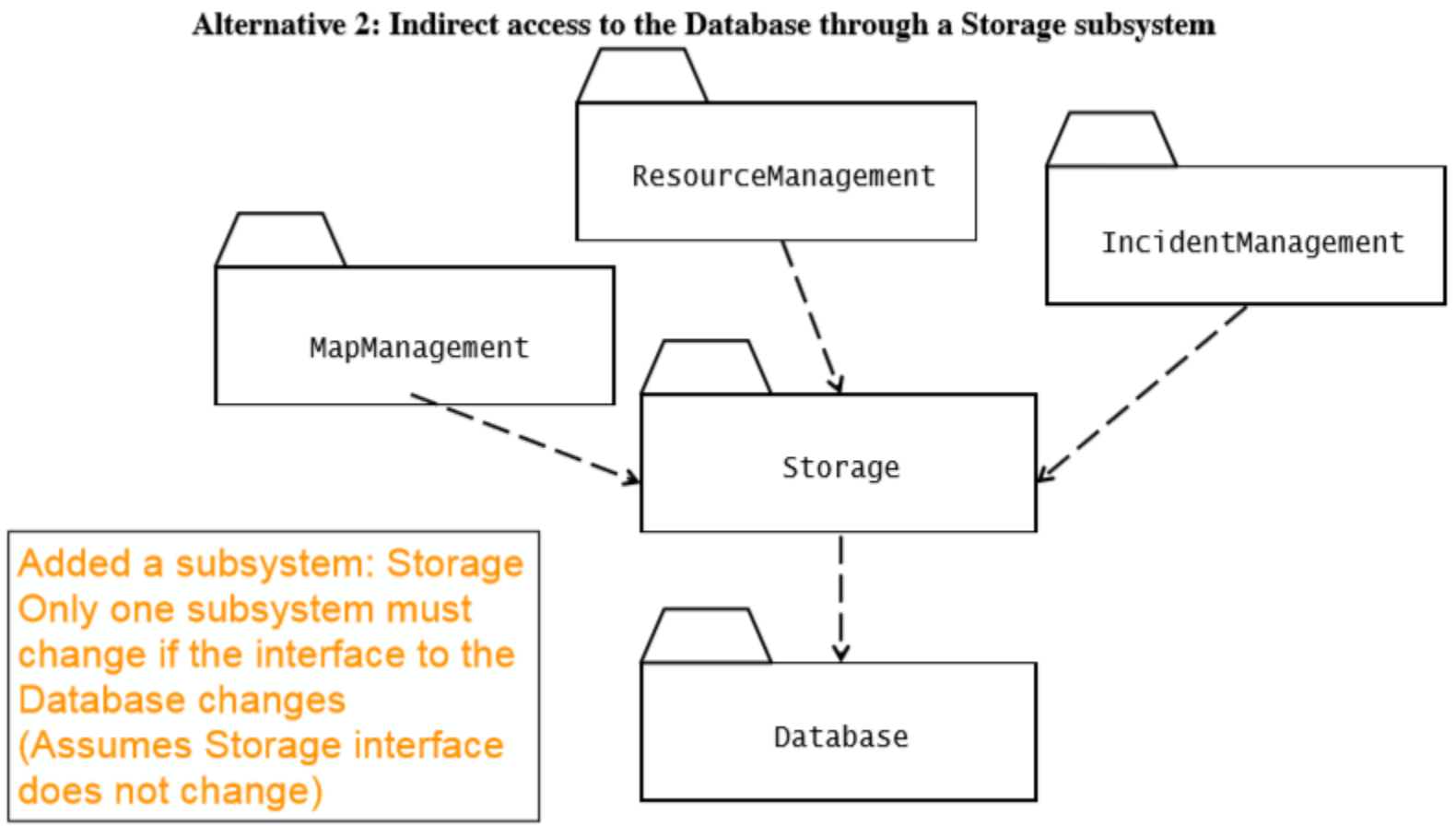
Highly coupled
many dependencies

Example: reducing the coupling of subsystems

Alternative 1: Direct access to the Database subsystem



Example: reducing the coupling of subsystems



Content Coupling

Def: One component modifies another.

Example:

Component directly modifies another's data

Component modifies another's code, e.g., jumps (goto) into the middle of a routine

Question

Language features allowing this?

Common Coupling

Def: More than one component share data such as global data structures

Usually a poor design choice because

- Lack of clear responsibility for the data
- Reduces readability
- Difficult to determine all the components that affect a data element (reduces maintainability)
- Difficult to reuse components
- Reduces ability to control data accesses

Cohesion

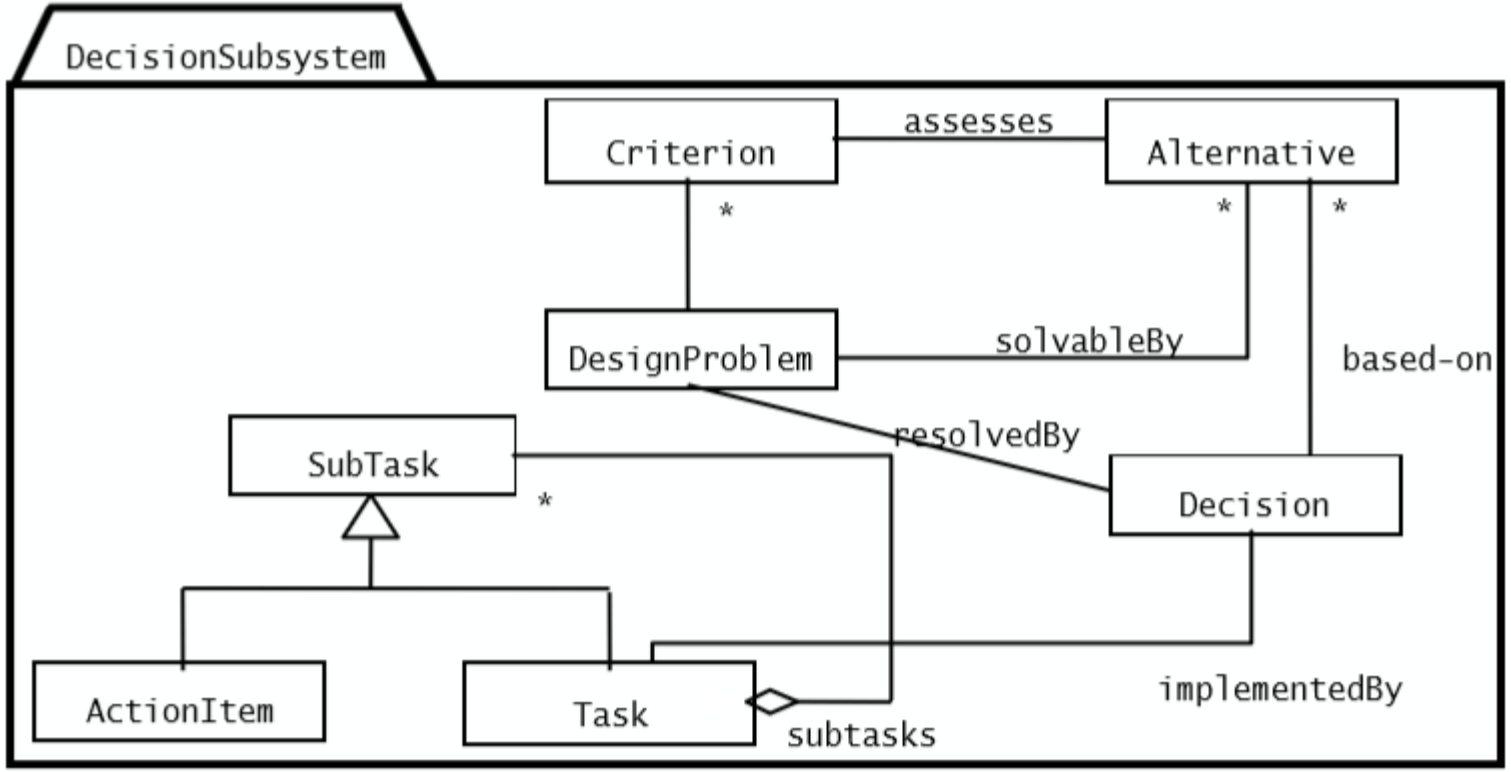
• Cohesion is the number of dependencies within a subsystem.
    - It measures the dependencies among classes within a subsystem.

• If a subsystem contains many objects that are related to each other and perform similar tasks, its cohesion is high.

• If a subsystem contains a number of unrelated objects, its cohesion is low.

• Goal: decompose system so that it leads to subsystems with high cohesion.
    - These subsystems are more likely to be reusable

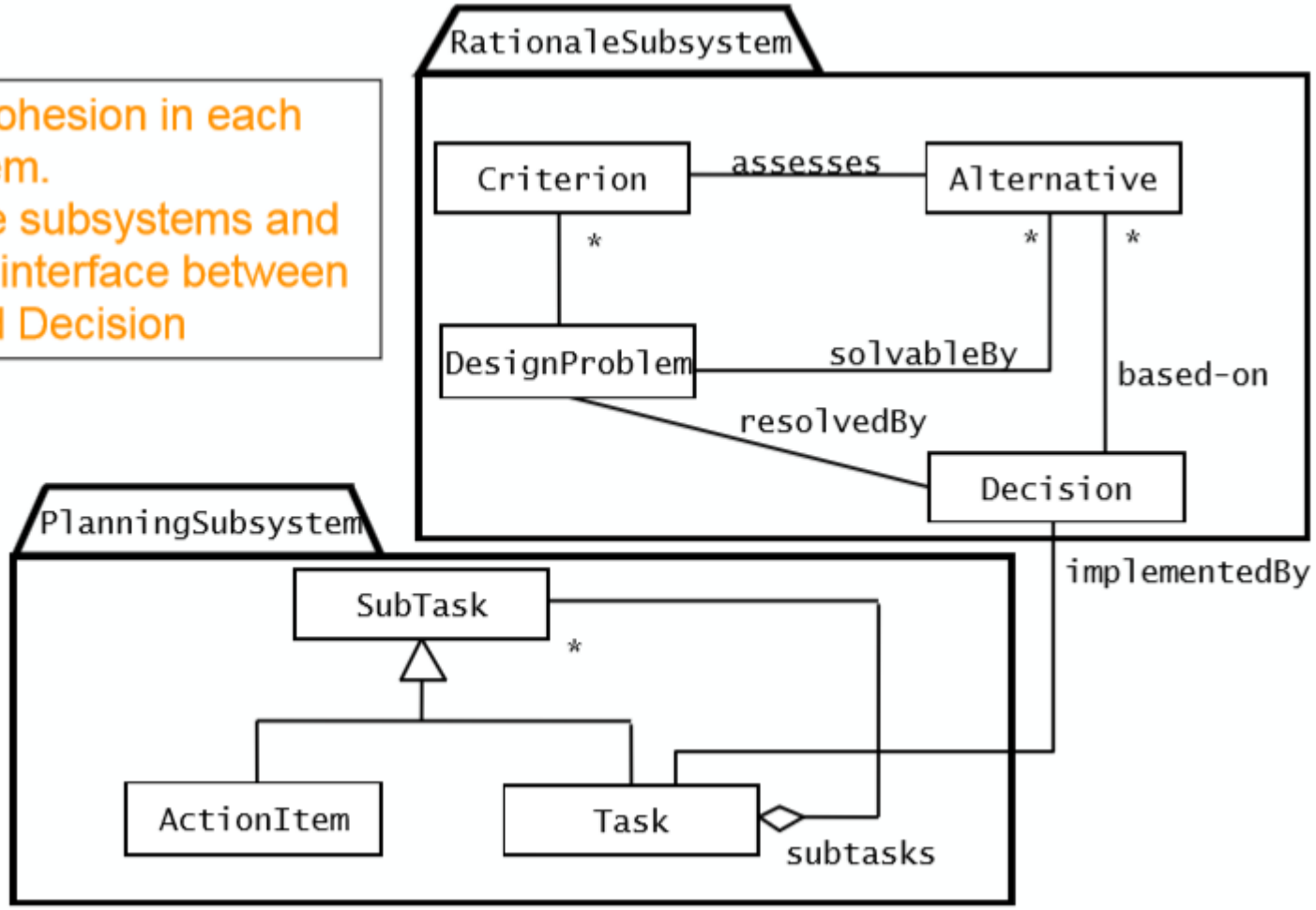
Example: Decision tracking system



Low Cohesion:
Criterion, Alternative, and DesignProblem have No relationships with SubTask, ActionItem, and Task

Example: Alternative decomposition: Decision tracking system

Higher cohesion in each subsystem.
But more subsystems and an extra interface between Task and Decision



Coincidental Cohesion

Def: Parts of the component are unrelated (unrelated functions, processes, or data)

Parts of the component are only related by their location in source code.

Elements needed to achieve some functionality are scattered throughout the system.

Accidental

Worst form

Example

1. Print next line
2. Reverse string of characters in second argument
3. Add 7 to 5th argument
4. Convert 4th argument to float

Logical Cohesion

Def: Elements of component are related logically and not functionally.

Several logically related elements are in the same component and one of the elements is selected by the client component.

Example

A component reads inputs from tape, disk, and network.

All the code for these functions are in the same component.

Operations are related, but the functions are significantly different.

CSc 131 – Computer Software Engineering

Architectural patterns

Architectural Models

- Simple box and line diagrams
- Each box is a component of the system (a subsystem)
- Boxes within boxes are subcomponents of a subsystem
- Arrows indicate data and/or messages are passed between components

Architectural Patterns

Architectural patterns

- Fundamental structural organization for software systems.
- High-level subdivision of the system.
- Highest level of pattern

Some architectural patterns

- Layers
- Pipes and filters
- Broker
- Model-View-Controller (MVC)
- Master-slave

Categories of architectural patterns

From mud to structure

- Patterns: Layers, “Pipes and filters”

Distributed systems

- Patterns: Broker, Master-Slave

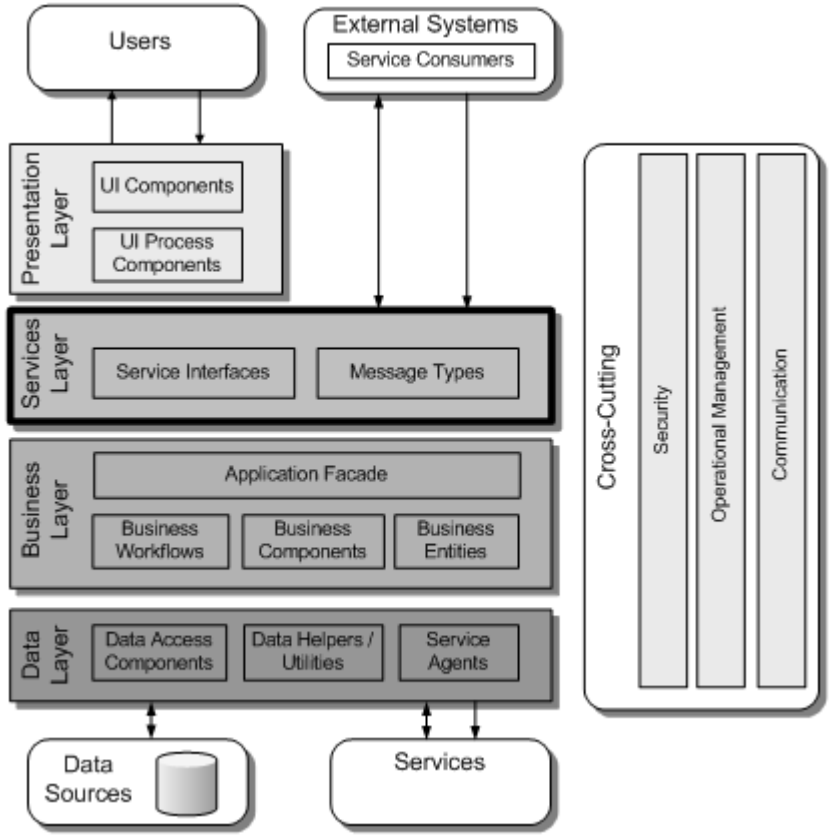
Interactive systems

- Patterns: Model-View-Controller (MVC)

Layered Architecture Pattern

- System functionality is organized into separate layers.
- Each layer relies only on facilities and services of layer immediately beneath it.
- Advantages:
  - Changes to one layer do not affect layers above
  - If interface changes, affects only layer above.
  - Easily replace one layer by another equivalent one (with same interface).
  - Portability: need to change only bottom layer to port to different machine(s).

Layered Architecture Pattern (example)



When an application will act as the provider of services to other applications, as well as implementing features to support clients directly, a common approach is to use a **services layer** that exposes the **functionality** of the application.

Service interfaces. Services expose a service interface to which all inbound messages are sent.

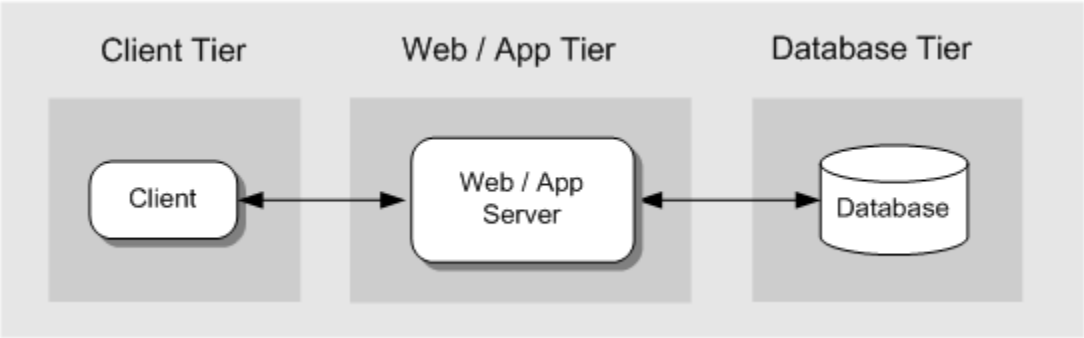
Message types. When exchanging data across the service layer, data structures are wrapped by message structures that support different types of operations.

Source: https://msdn.microsoft.com/en-us/library/ee658109.aspx

Tiers

Three-Tier (example)

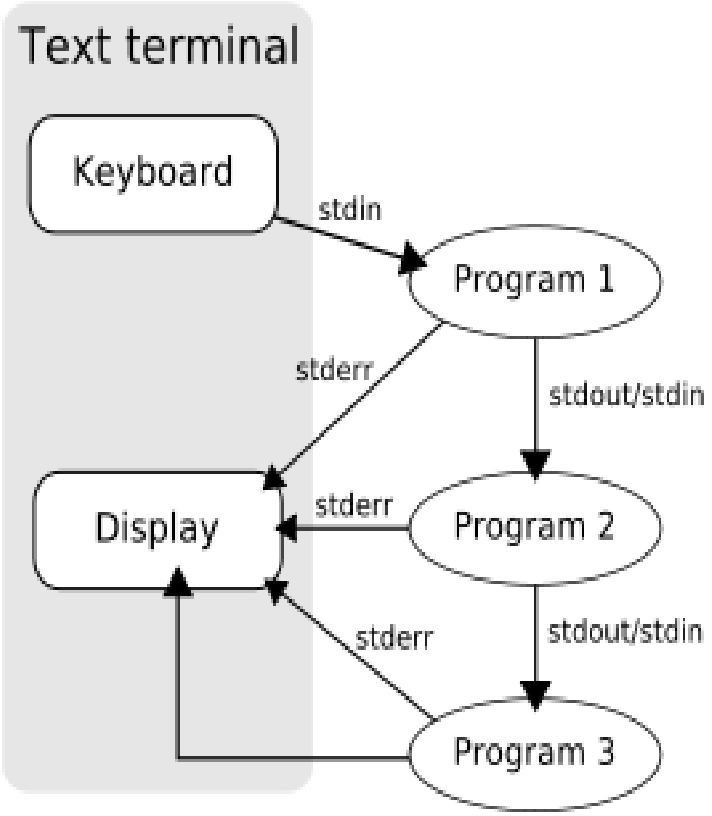
In a three-tier design, the client interacts with application software deployed on a separate server, and the application server interacts with a database that is also located on a separate server. This is a very common pattern for most Web applications and Web services. Figure 7 illustrates the three-tier deployment pattern.



The Pipe-and-Filter Architectural Pattern

Streams of data, in a relatively simple format, passed through series of processes

- Data constantly fed into a pipeline; Each component transforms the data in some way.
- The processes work ***concurrently***. Constant data in and coming out.
- Very flexible architecture.
  - —*Almost all the components could be* ***removed***.
  - —*Components may* ***added***, ***changed***, ***deleted***, ***reordered***…
- Very flexible particularly (for example) as in converting **data** or filtering out (removing) **characters** or '**features**', etc.
- Sometimes (oftentimes) data might undergo a series of **transformations**…
- Can also split pipelines or join pipelines together.



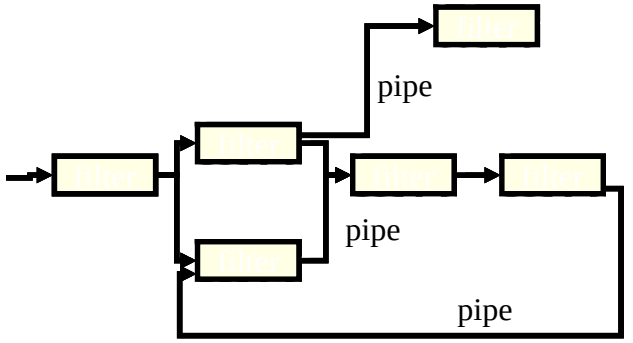
In Unix: using sed and awk

Data Flow Design

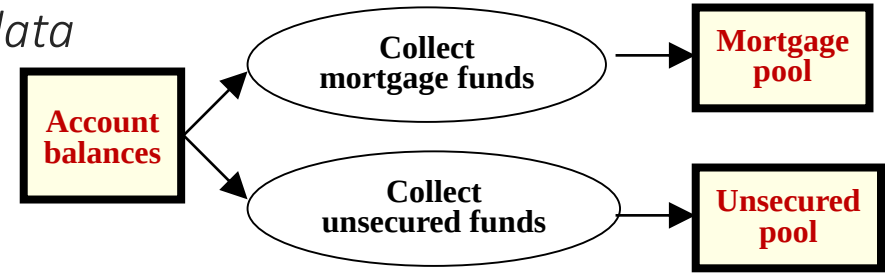
Data flowing among processes

Two categories:

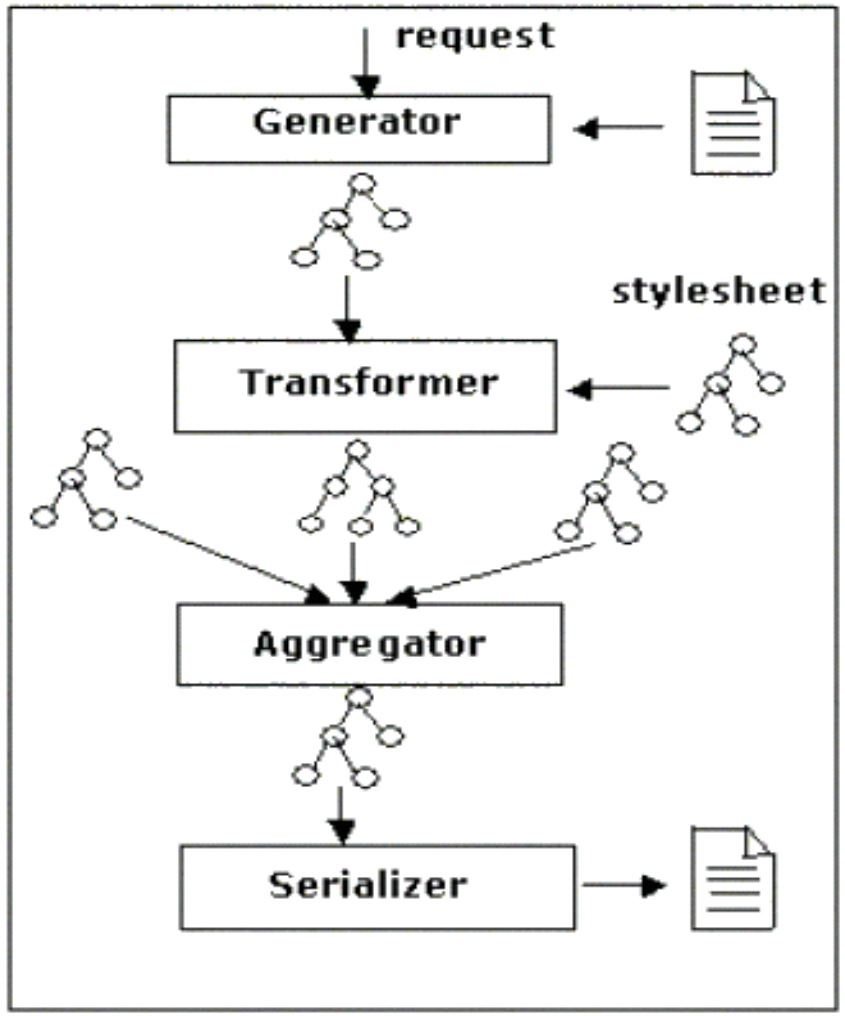
- Pipes and filters
  - *Filters: processes*
  - *Pipes: input streams*

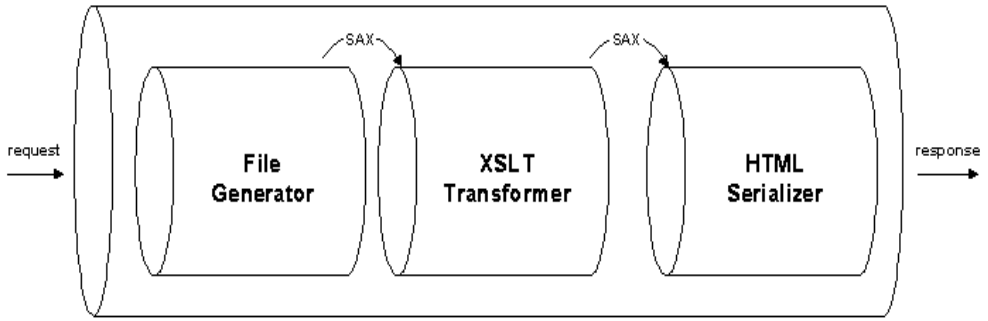


- Batch sequential
  - *Pipe and filter where input streams are batches of data*



Example: Apache Cocoon's pipes & xslt filters





What is MVC?

Architectural design pattern which works to separate data and UI for a more cohesive and modularized system

What is MVC?

Model represents the data model

- “Manages behavior and data of the application domain”

View represents the screen(s) shown to the user

- “Manages the graphical and/or textual output to the portion of the bitmapped display that is allocated to its application”

Controller represents interactions from the user that changes the data and the view

- “Interprets the mouse and keyboard inputs from the user, commanding the model and/or the view to change as appropriate” (Burbeck)

How does it work?

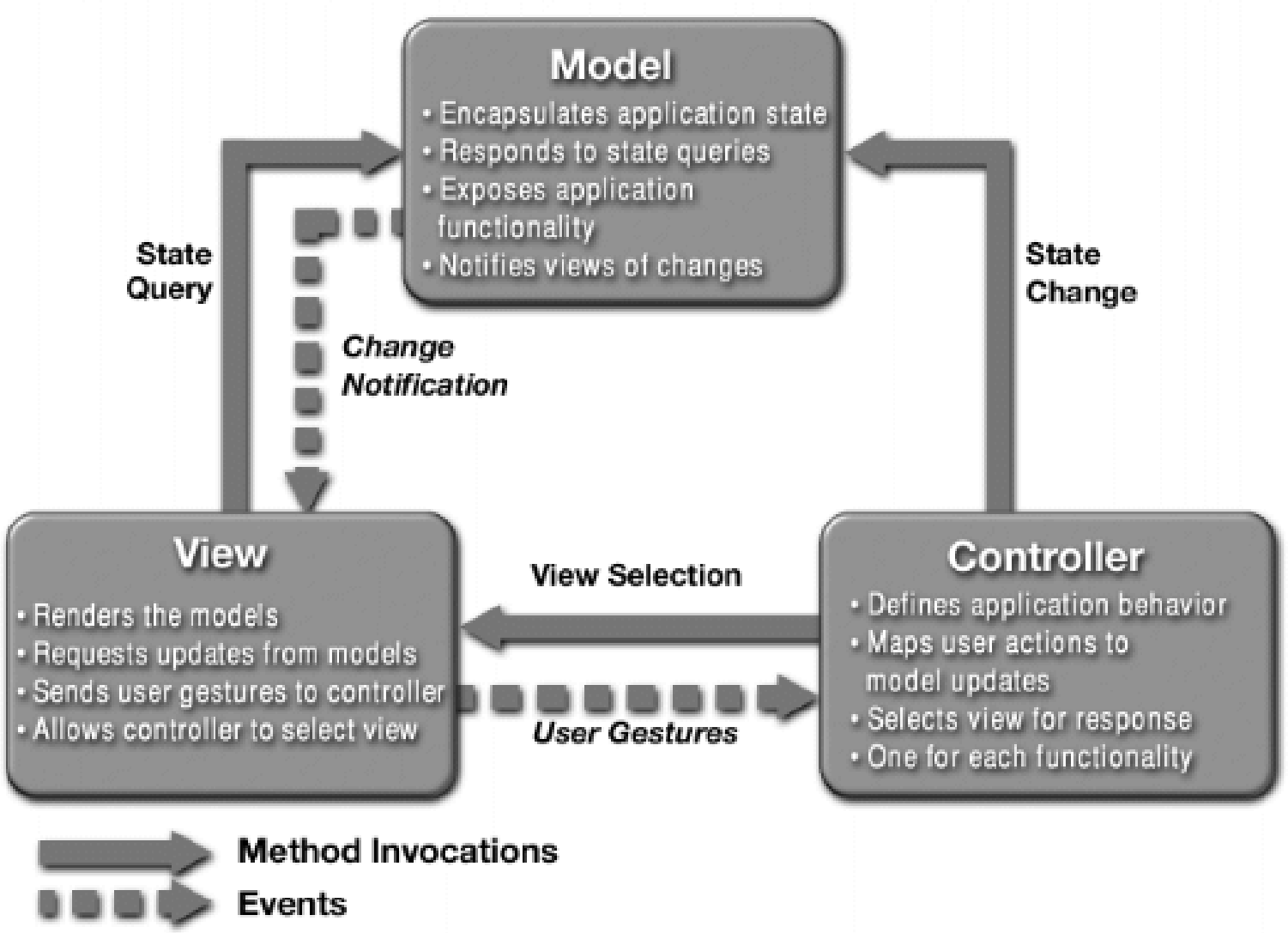
User inputs a command

Controller handles input and updates model or changes the view

View, which relies on model to show data to user, updates if necessary

Rinse and Repeat

What is MVC?


Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
Change Notification
State Change
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

Service-oriented architecture

SOA

Web services

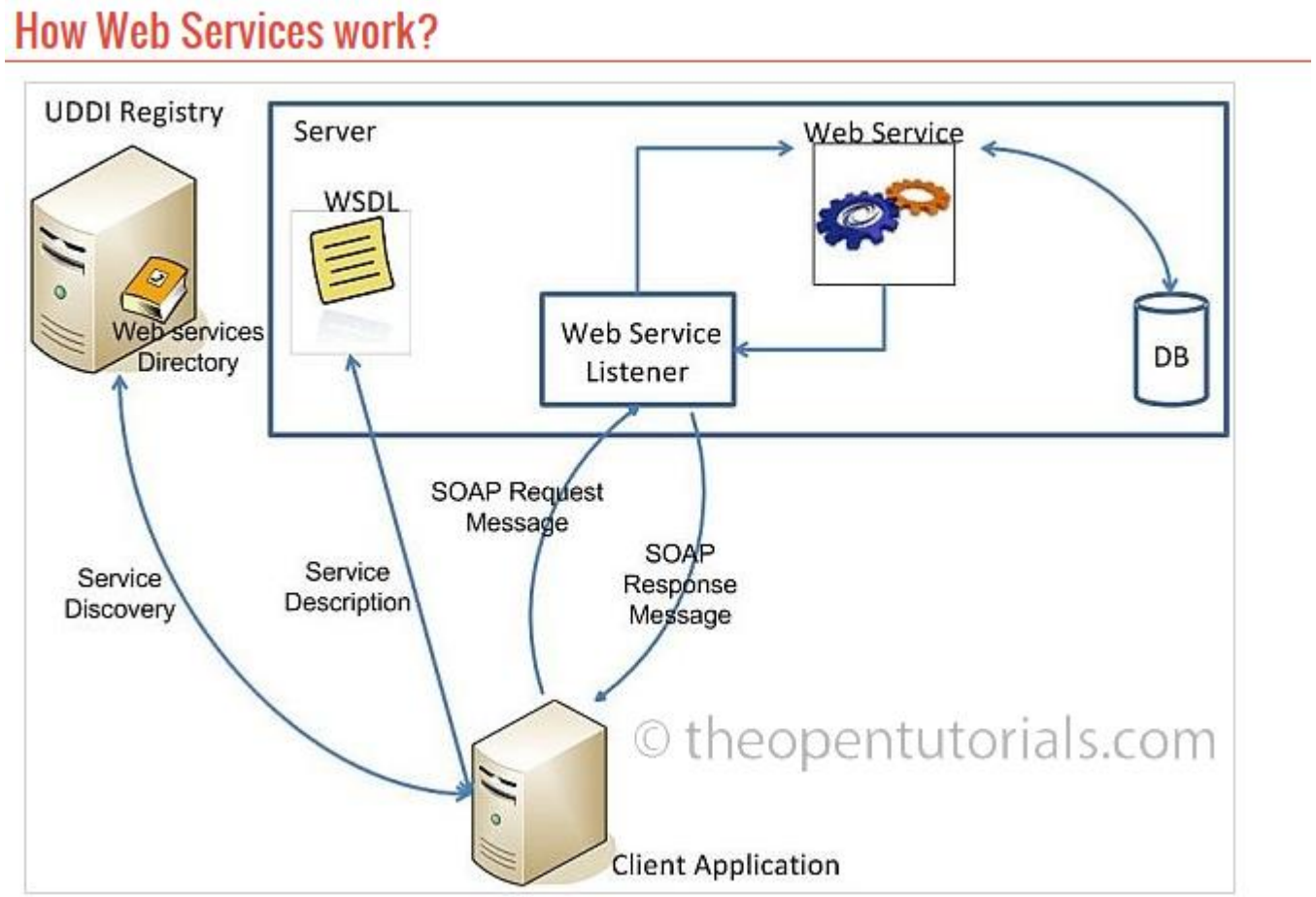
- HTTP for transportation
- SOAP structured data exchange
- UDDI for discovery
- WSDL for description
- XML used everywhere

Service registry

Service provider

Service requestor

How web service works ?



Source: http://theopentutorials.com/tutorials/web-services/web-services-platform/

How web service works ?

UDDI: UDDI (Universal Description, Discovery and Integration) is a platform-independent, XML based registry service where companies can register and search for Web services.

WSDL: WSDL (Web Services Description Language) is an XML-based language for locating and describing Web services. WSDL definition describes how to access a web service and what operations it will perform along with the message format and protocol details for the web service. WSDL is a W3C standard.

SOAP: SOAP (Simple Object Access Protocol) is an XML-based communication protocol for exchanging structured information between applications over HTTP, SMTP or any other protocol. In other words, SOAP is a protocol for accessing a Web Service.

Source: http://theopentutorials.com/tutorials/web-services/web-services-platform/

Web Services

Characteristic of Web Services

- Provide interoperability between various software application
- Standard interface on server
- Enable to run on disparate platforms

Benefit of Web Services

- Allow to share business logic, data and processes as services
- Reduce the software purchasing cost
- Form the technical foundation for software as service business model

Source: http://people.ischool.berkeley.edu/~hal/Courses/StratTech09/Tech/Tech05/D-slides.ppt

Software as a service (Saas)

Traditional Software

Build Your Own

On-Demand Utility

Plug In, Subscribe

Pay-per-Use

Comparison of business model

Traditional packaged software

- Designed for customers to install, manage and maintain.
- Architect solutions to be run by an individual company in a dedicated instantiation of the software

Software as a serivce

- Designed from the outset up for delivery as Internet-based services
- Designed to run thousands of different customers on a single code

Example of a WSDL

http://www.webservicex.net/geoipservice.asmx?WSDL