DEVELOPMENT OF INFORMATION SYSTEMS

Lectures 1-12 Summary

2021/2022

Martin Radvanský

Information system

IS is the interaction between people, processes and data. Is designed to process (retrieve, transmit, store, search, manipulate, display) information. (Economic, personal, payroll system, Warehauses, CRM, online store, etc.)

Components of IS - parts which interact to do functionality (HW, SW, Network, DB, Human resource).

Architecture – plan for the structure of IS. Focuses on abstractions that can be used by multiple designs. (Enterprise architecture)
Design – concrete plan to build IS. (UI design, DB design)

Information system

By the **domain** we understand a group of related "things, concepts and terms" in the sense of a customer or user point of view. (Bank, Insurance, Hospital)

Key questions:

- **WHAT?** in the sense of data.
- **HOW?** the processes performed with the information.
- **WHERE?** places where information is worked with.
- **WHO?** Who works with information and in what role.
- **WHEN?** When and on what impulses information is worked with
- **WHY?** It is about aims and rules to achieve the requested goals.

Levels of Abstraction – Strategic decision, Business model, System model, Technology, Operations.
Roles: Planner, Owner, Designer, Bulder, Programmer, User

Information system

Architecture of IS : the basic organization of a software system, including its components, their interrelationships and relationships with the system environment, the principles of the design of such a system and its development. **Contain**: app domain view, developers view, view of data and flows, physical layout of components.

Architecture:
Static: The structure of the system is given at design time (Simple IS)
Dynamic: Supports creation, modification and termination components and links at runtime. Structure of IS is changed dynamically. (Component based IS)
Mobile: Extends the dynamic architecture to include mobile elements. Components and link aremoved at runtime accordind to the state of computation. (Mobile based IS)

Information system

A **IS component** is a software package, service or generally a module that provides a particular functionality, and therefore encapsulating functionality and data (Warehouse, CRM, User management)

How to determine architecture:
Identify system requriments, divide IS into components, assign requirements to the components, Verification that requirements has been allocated. Use standards.

Core competencies:
Communication with the user (presenting information) gathering requirements
Information processing and (temporary) storage. Main purpose of IS
Pernament store of information (data). Persistence to DB storage)

Design patterns

By the **domain** we understand a group of related "things,

Design patterns

By the **domain** we understand a group of related "things,

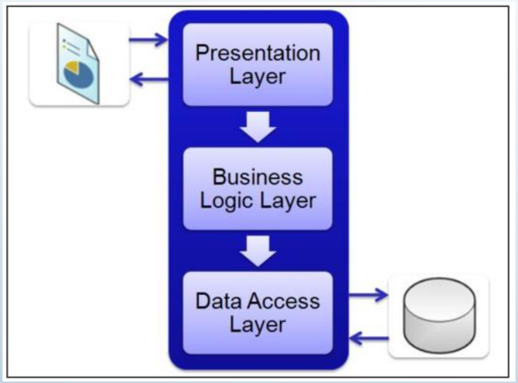
Design patterns

Design patterns represent the best practices used by experienced object-oriented software developers.

It usually contain:

- Concise name (title)
- Problem (What)
- Context (Who, When, Where, Why)
- Solution including UML diagrams (How)
- Examples including source codes

Example: singleton

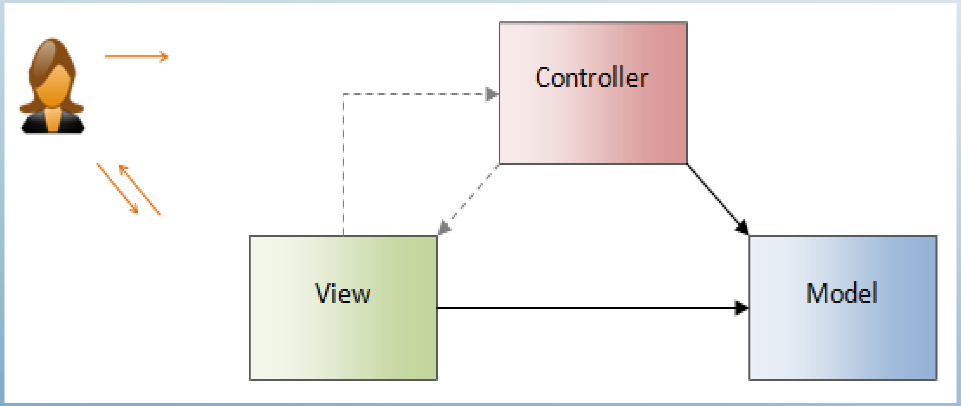


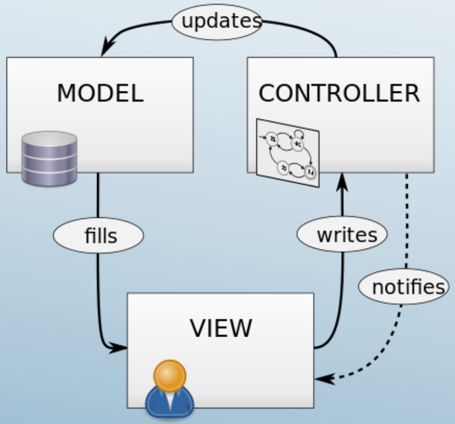
Three layer architecture

well-established software application architecture that organizes applications into three logical layers and/or physical computing tiers. The each tier runs on its own infrastructure, each tier can be developed simultaneously by a separate development team

Design patterns

- **MVC – model-view-controller** used for
  - Separation of layers at the logical level
  - Dependency minimization, isolated modification of individual layers
  - Controller - component that enables the interconnection between the views and the model so it acts as an intermediary.
  - It is triangular architecture.





- **Domain logic patterns** – handles the part of the program that encodes the real-world business rules that determine how data can be created, stored, and changed. Transaction Script, Domain Model, Table Module, Service Layer

Design patterns

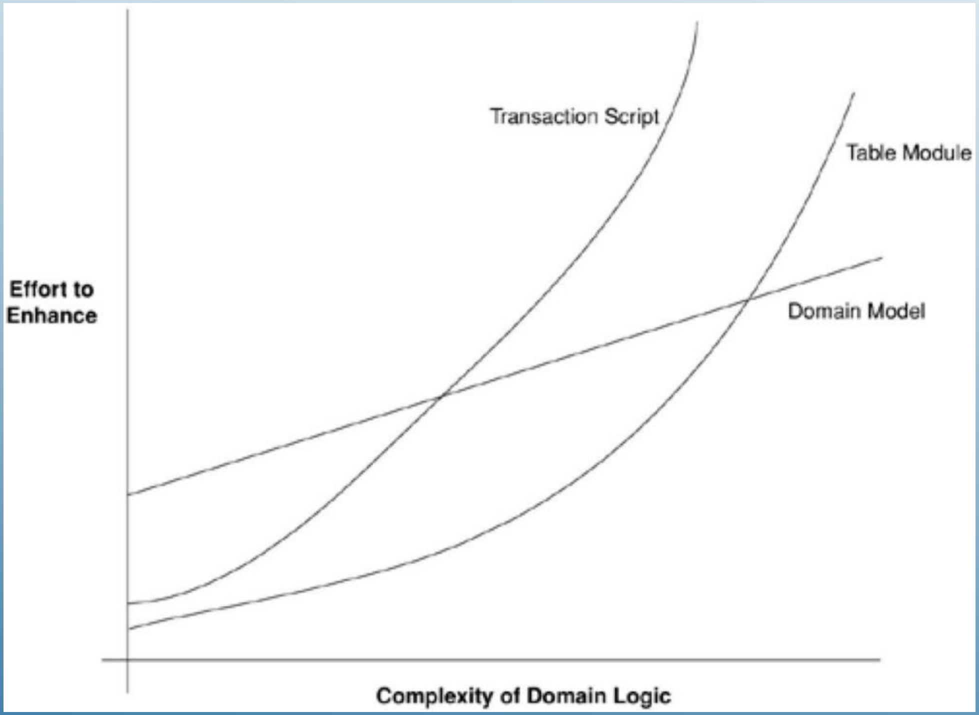
- **Transaction script -** business applications is represented as a set of transactions. It organize bussines logic into procedures which hadles a single request from presentation. Examples is just method like CalculateSalary. Usefull for simple Bussines logic.

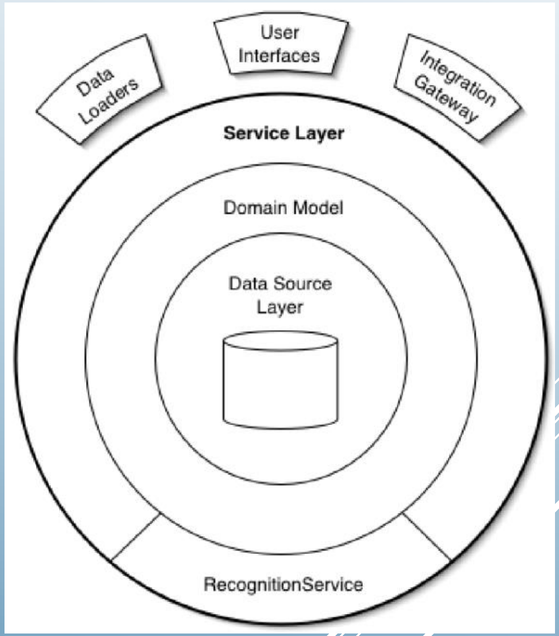
- **Domain Model** pattern **-** forms a network of interconnected objects, in which each object is a separate significant entity: it can be as big as a corporation or as small as the line from the order form. It incorporate behaviour and data. Representation is Class diagram. Usefull for easy maintenance, long term systems with complex BL. One order object is one instance of class Order.

- **Table Module** - pattern divides the logic of the definition domain (domain) into separate classes for each table in the database and one instance of the class contains various procedures that work with data. All orders are managed by single table module. Not too complex BL, A single instance that handles the business logic for all rows in a database table or view.

Data Source Architecturals patterns

Service layer - Defines an application‘s boundary with a layer of services that establishes a set of available operations and coordinates the application‘s response in each operation



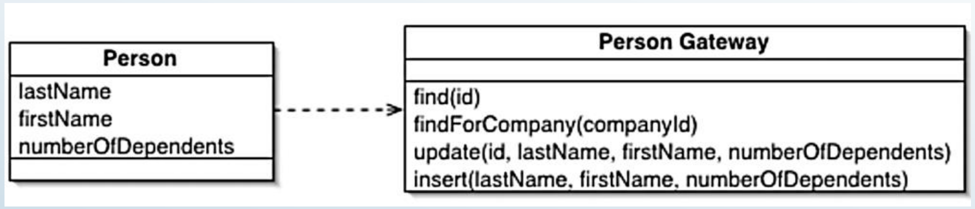


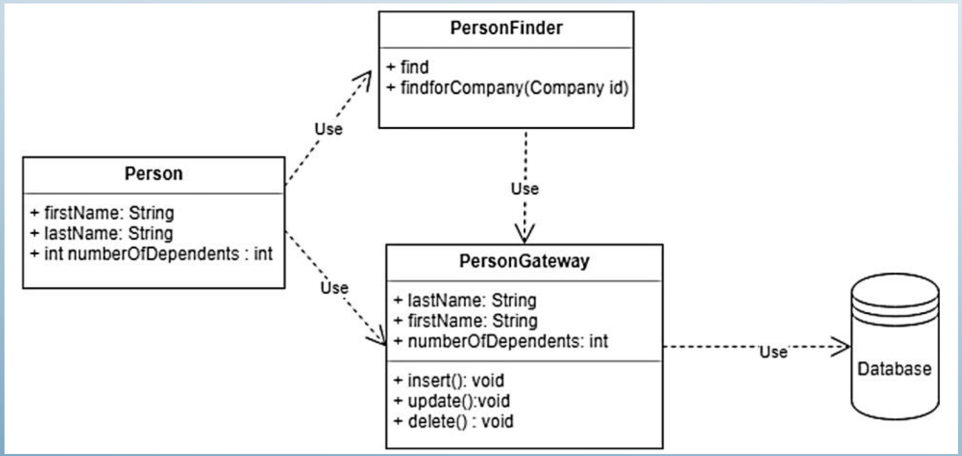
Data Source patterns

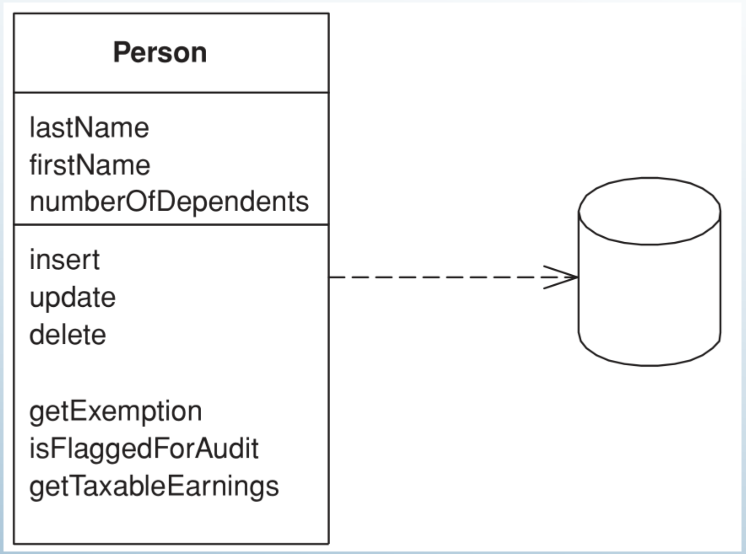
These patterns solves the basic requirement of the independence of the domain logic from the logic of data access. Does not contain domain logic methods

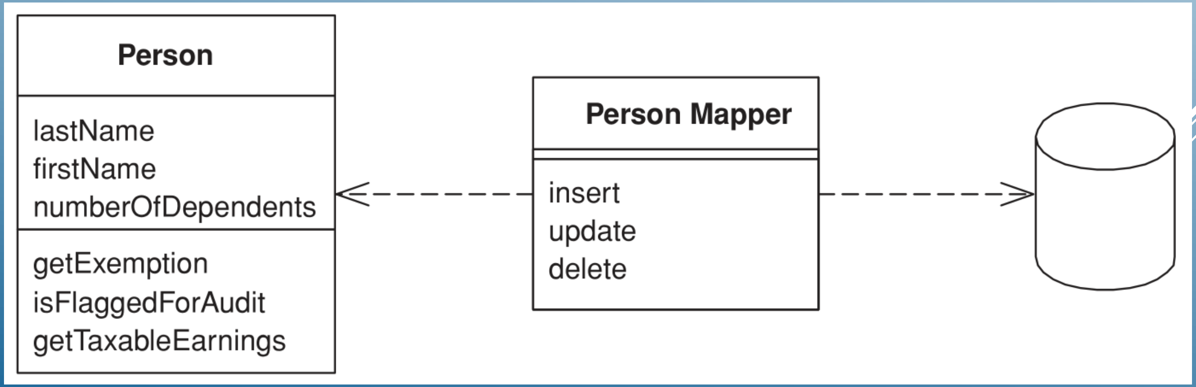
- **Table data gateway** object that acts as a Gateway (GoF) to a database table. One instance handles all the rows in the table.
- **Row data gateway** object that acts as a Gateway to a single record in a data source. There is one instance per row.
- **Active record** object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data.
- **Data mapper -** layer of Mappers that moves data between objects and a database while keeping them independent of each other and the mapper itself.

Data Source patterns

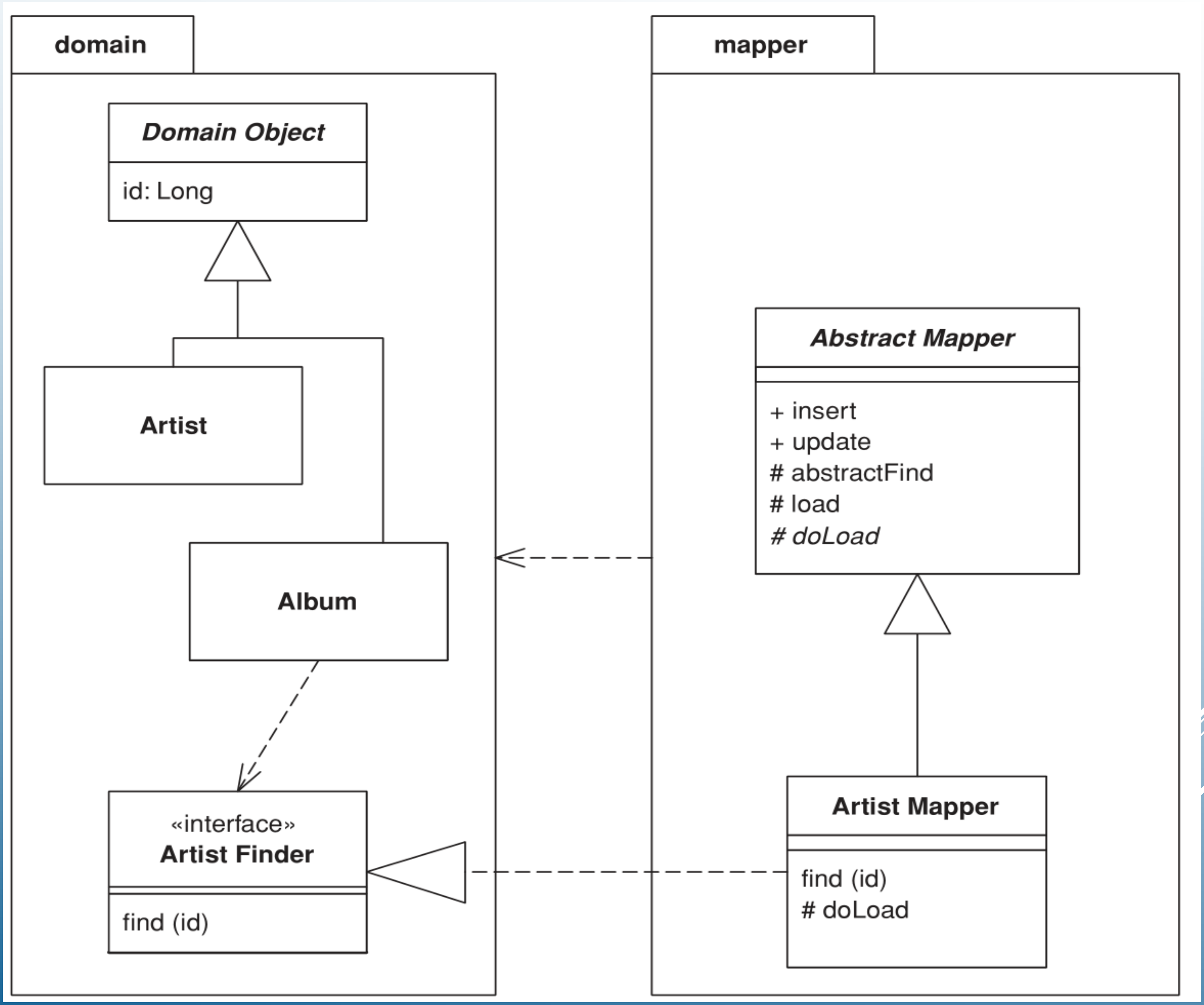

Person
lastName
firstName
numberOfDependents
Person Gateway
find(id)
findForCompany(companyId)
update(id, lastName, firstName, numberOfDependents)
insert(lastName, firstName, numberOfDependents)







Data Source patterns



Data Source patterns

- **Using Data source patterns and domain logic patterns**

- **Table data gateway**
  - **Yes –** Transaction script, Table module
  - **Yes/No –** Domain model (complexity)
- **Row data gateway**
  - **Yes –** Transaction script
  - **No –** Domain model
- **Active record**
  - **Yes –** Transaction script
  - **No –** Domain model
- **Data mapper**
  - **Yes -** Domain Model
  - **No -** Active Records, Table Module

Object-relational behavioral patterns

These patterns are designed to support and reduce problems that are inherent in data source patterns. These problems include managing of transactions and managing of in-memory objects to avoid duplication and preserve data integrity

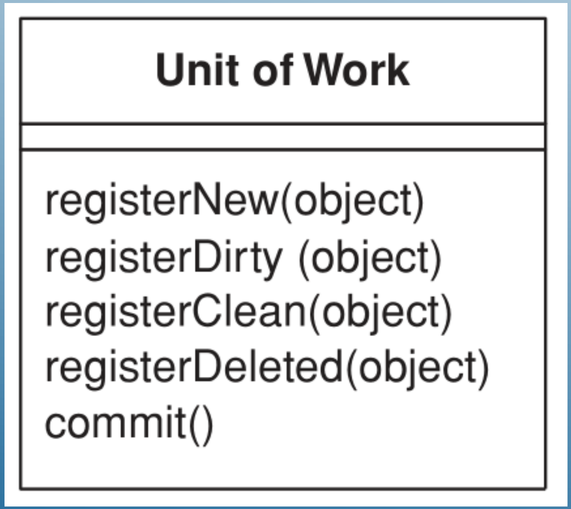
- Unit of Work
- Identity Map
- Lazy Load

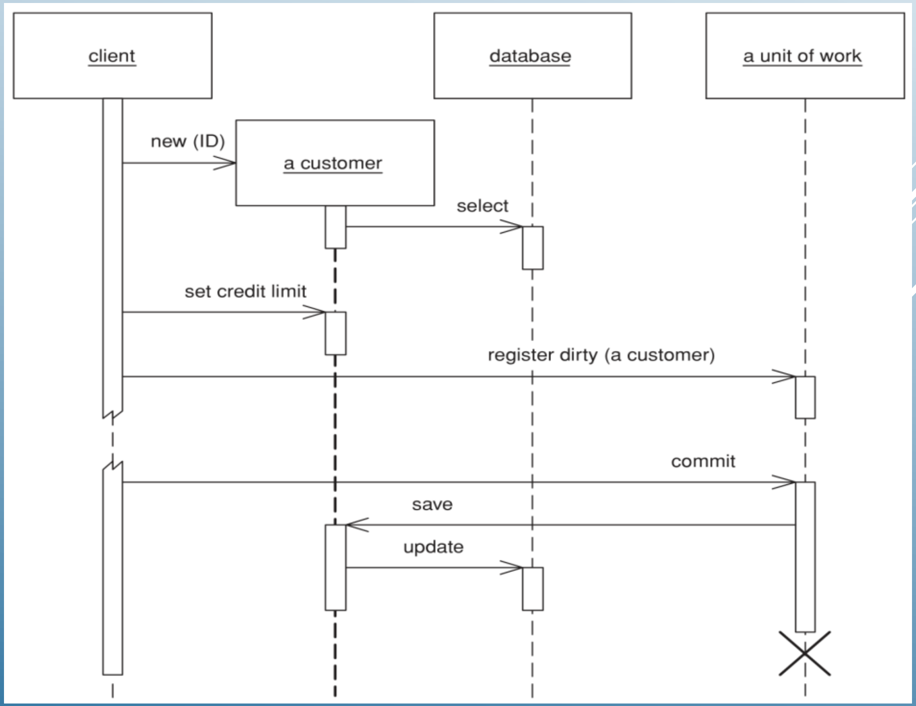
Object-relational behavioral patterns

Unit of work

Maintains a list of objects affected by a business transaction and coordinates the writing out of changes and the resolution of concurrency problems. Work keeps track of everything you do during a business transaction that can affect the database

- Active Records, -Table Data Gateway, +Row Data Gateway, +Data Mapper



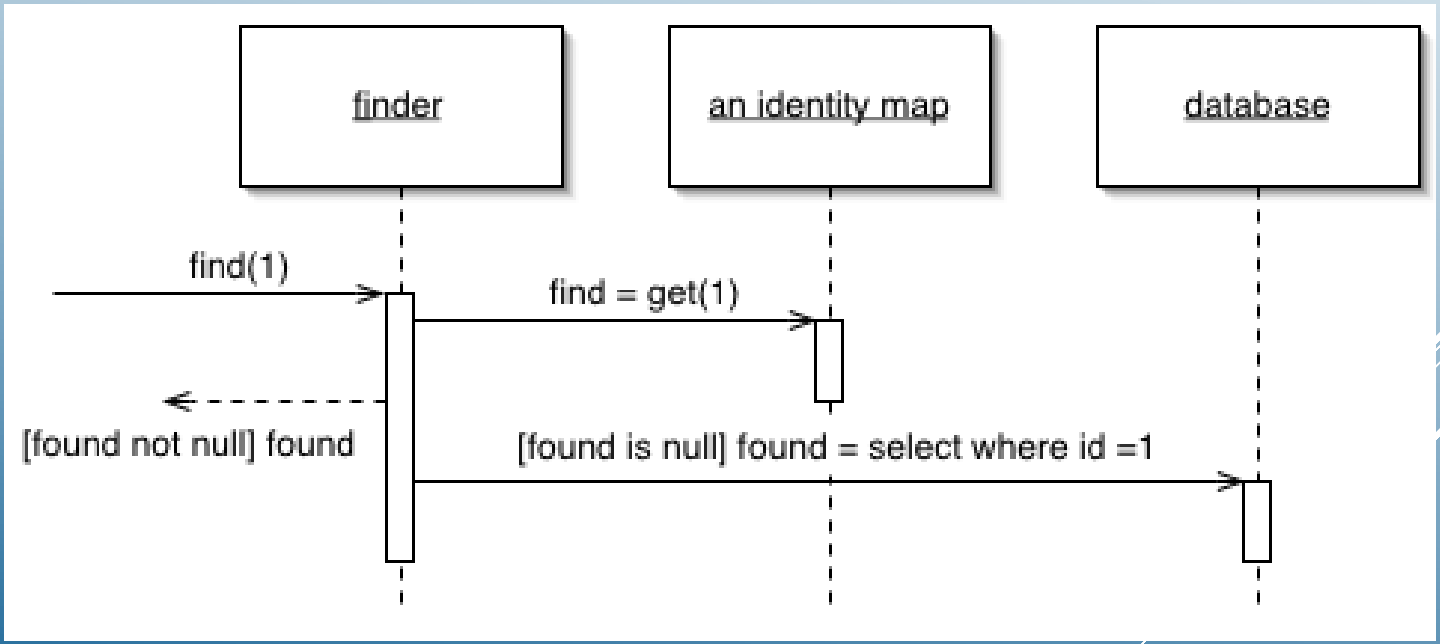


Object-relational behavioral patterns

Identity map

Ensures that each object gets loaded only once by keeping every loaded object in a map. Looks up objects using the map when referring to them. An Identity Map keeps a record of all objects that have been read from the database in a single business transaction. Whenever you want an object, you check the Identity Map first to see if you already have it.

+ Active Records, Table Data Gateway, Row Data Gateway, Data Mapper

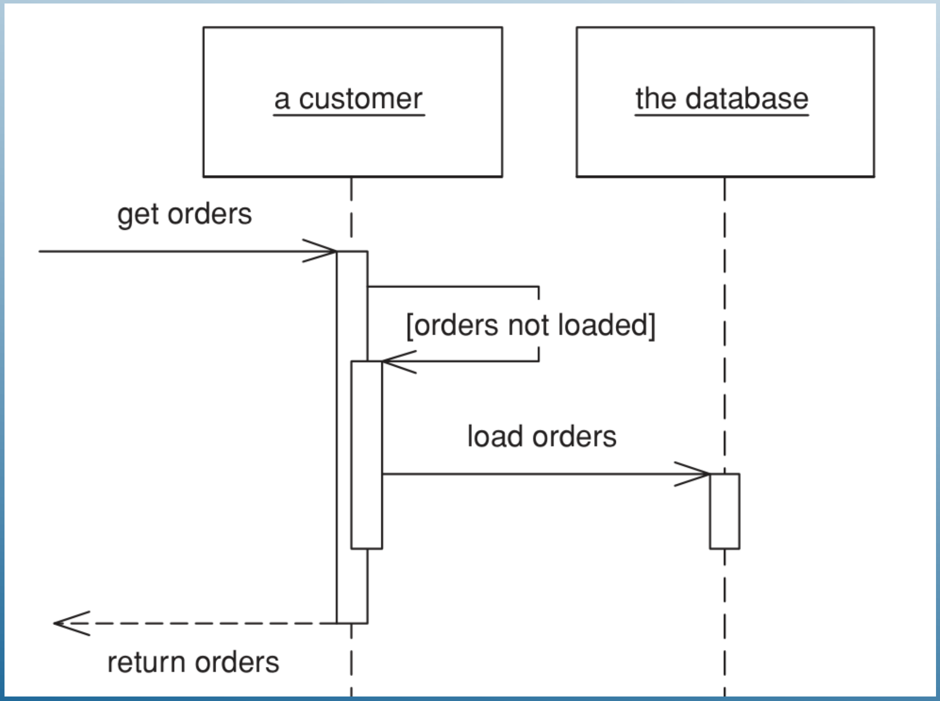


Object-relational behavioral patterns

Lazy load

An object that doesn’t contain all of the data you need but knows how to get it. For loading data from a database into memory it’s handy to design things so that as you load an object of interest you also load the objects that are related to it.

+Active Records, Table Data Gateway, Row Data Gateway, Data Mapper



Object-Relational Structural Patterns

These patterns deal with the structural differences between in memory objects and database tables/rows.

- **Identity Field** - Saves a database ID field in an object to maintain identity between an in-memory object and a database row. +Domain Model, +Row Data Gateway

- **Foreign Key Mapping** - Maps an association between objects to a foreign key reference between tables. If two objects are linked together with an association, this association can be replaced by a foreign key in the database.

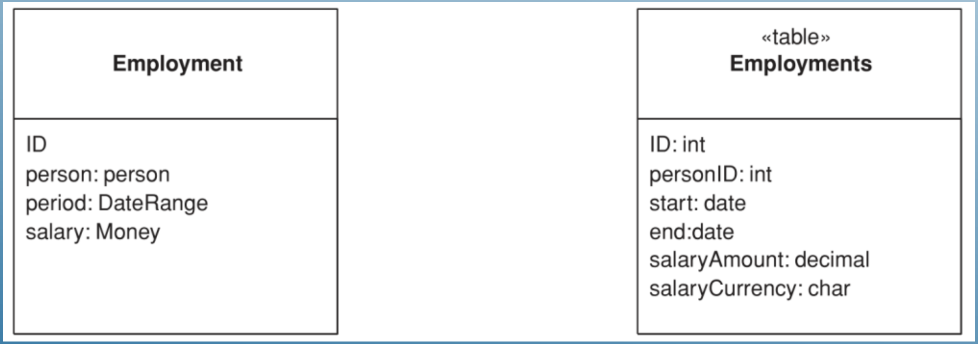
- **Association Table Mapping** - Saves an association as a table with foreign keys to the tables that are linked by the association.

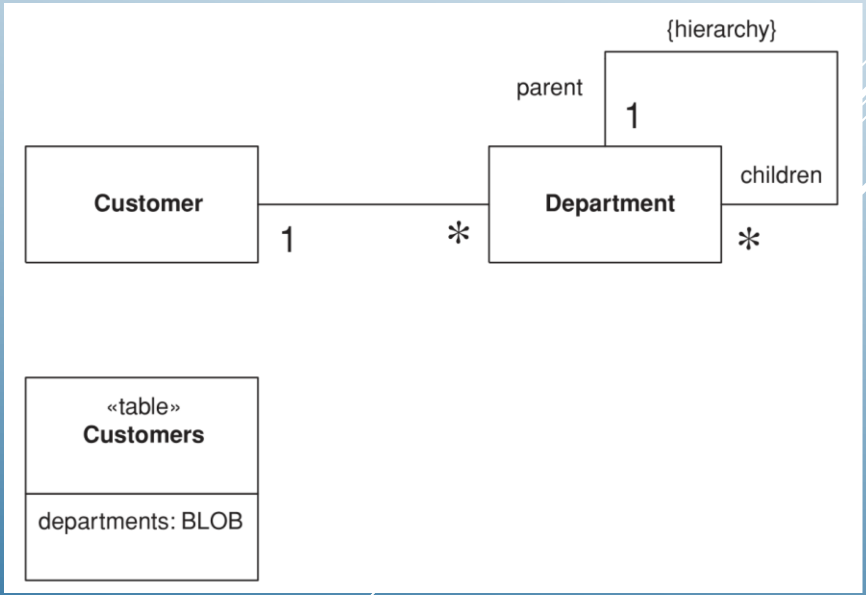
- **Dependent Mapping** - Has one class perform the database mapping for a child class. One class (the **dependent**) relies upon some other class (the **owner**) for its database persistence.

Object-Relational Structural Patterns

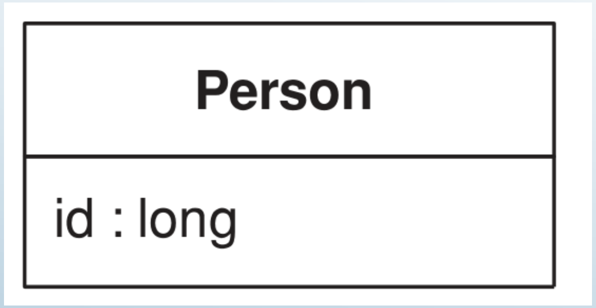
- **Embedded Value** - maps an object into several fields of another object's table. (Salary maps into Amount and Currency. Name maps into First name and Surrname Name. Datetime into Date, Time, Day of week)

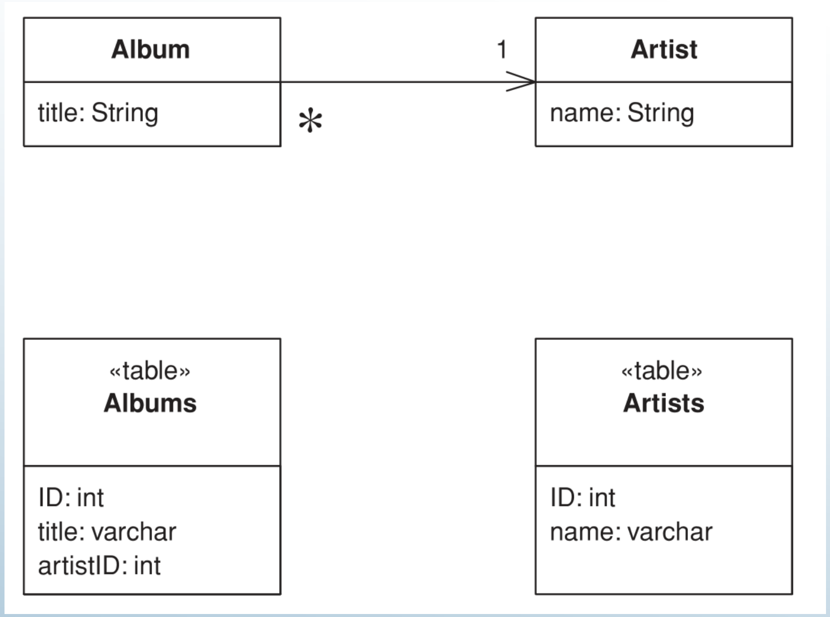
- **Serialized LOB** - Saves a graph of objects by serializing them into a single large object (LOB), which it stores in a database field. (Order-Details – store Customer details data into LOB in Order. Save order items into LOB of Order)

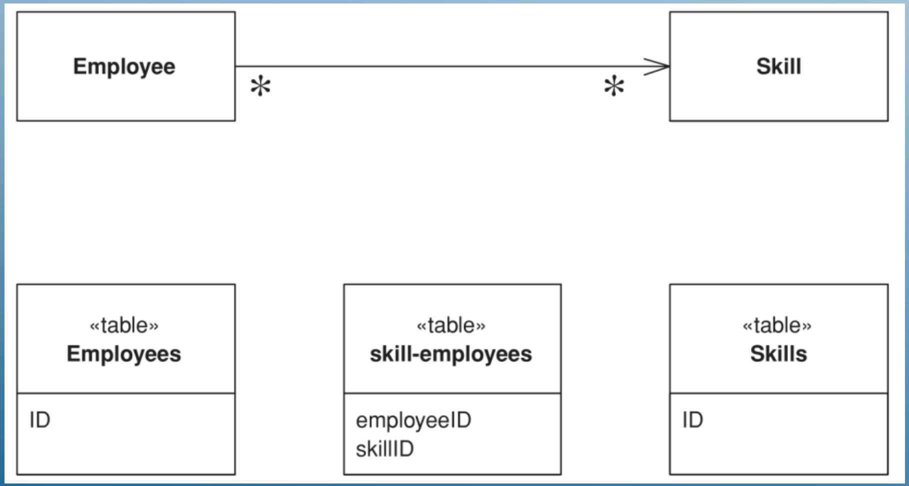


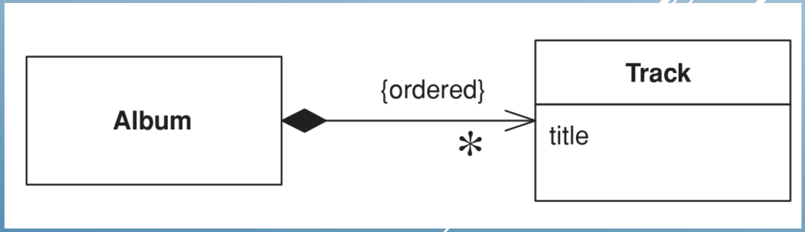


Object-Relational Structural Patterns



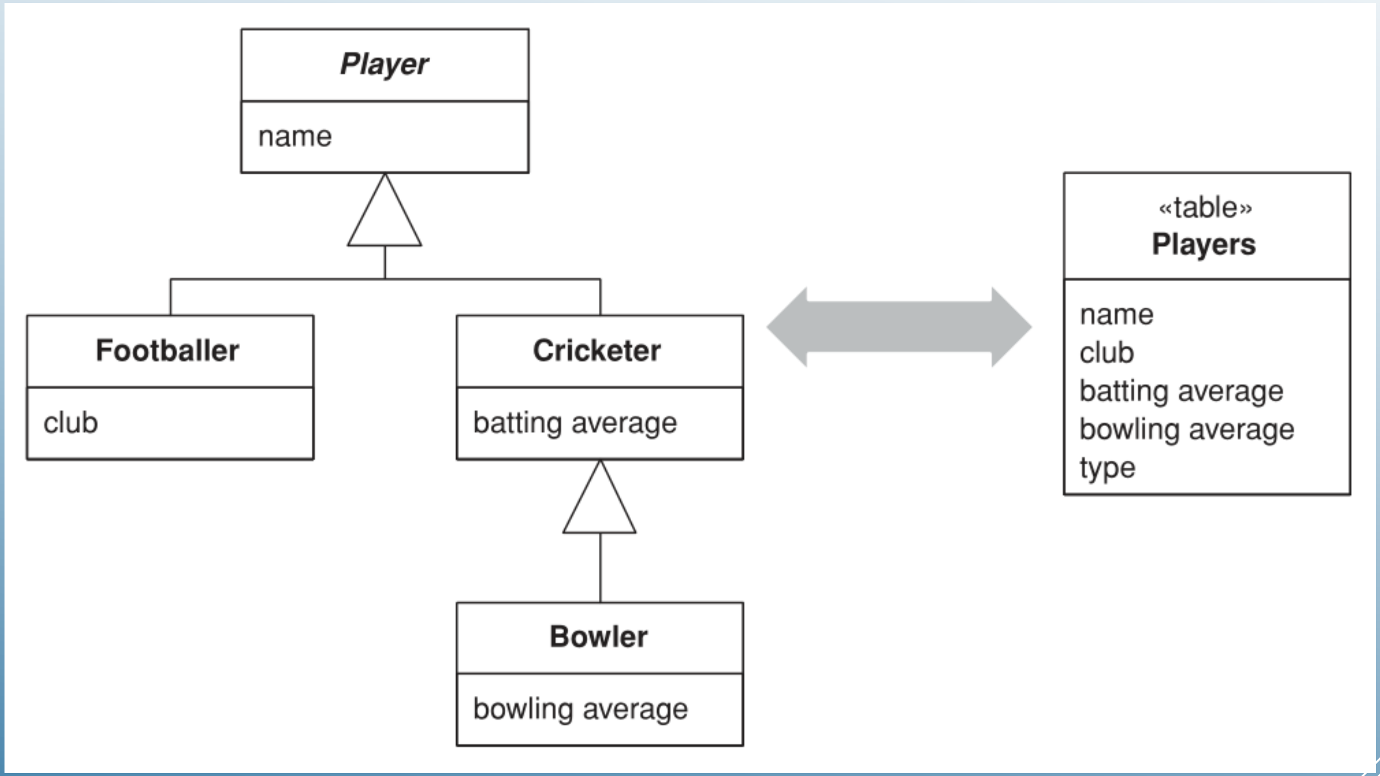






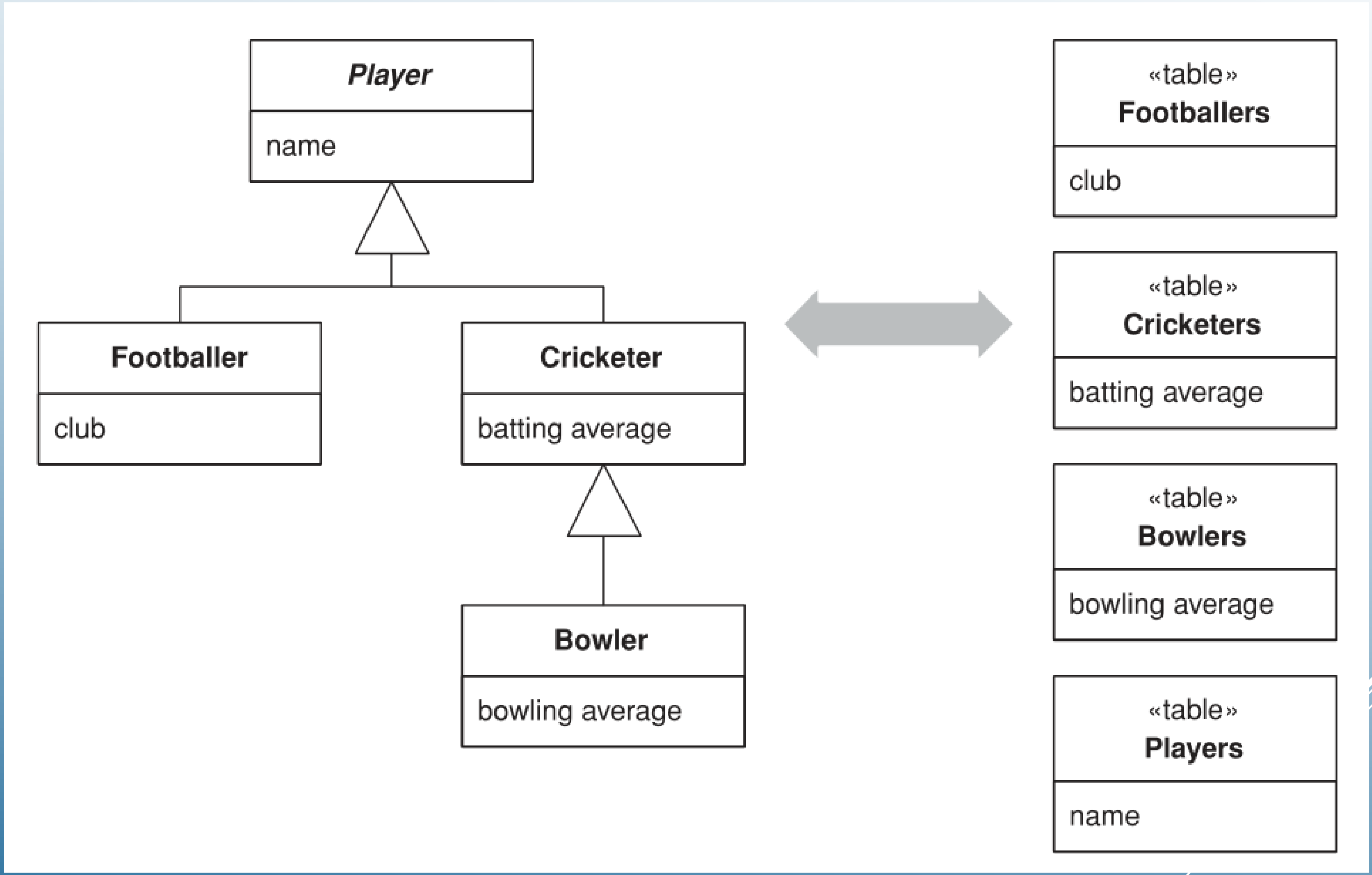
Inheritance mappings

Single Table Inheritance - Represents an inheritance hierarchy of classes as a single table that has columns for all the fields of the various classes.



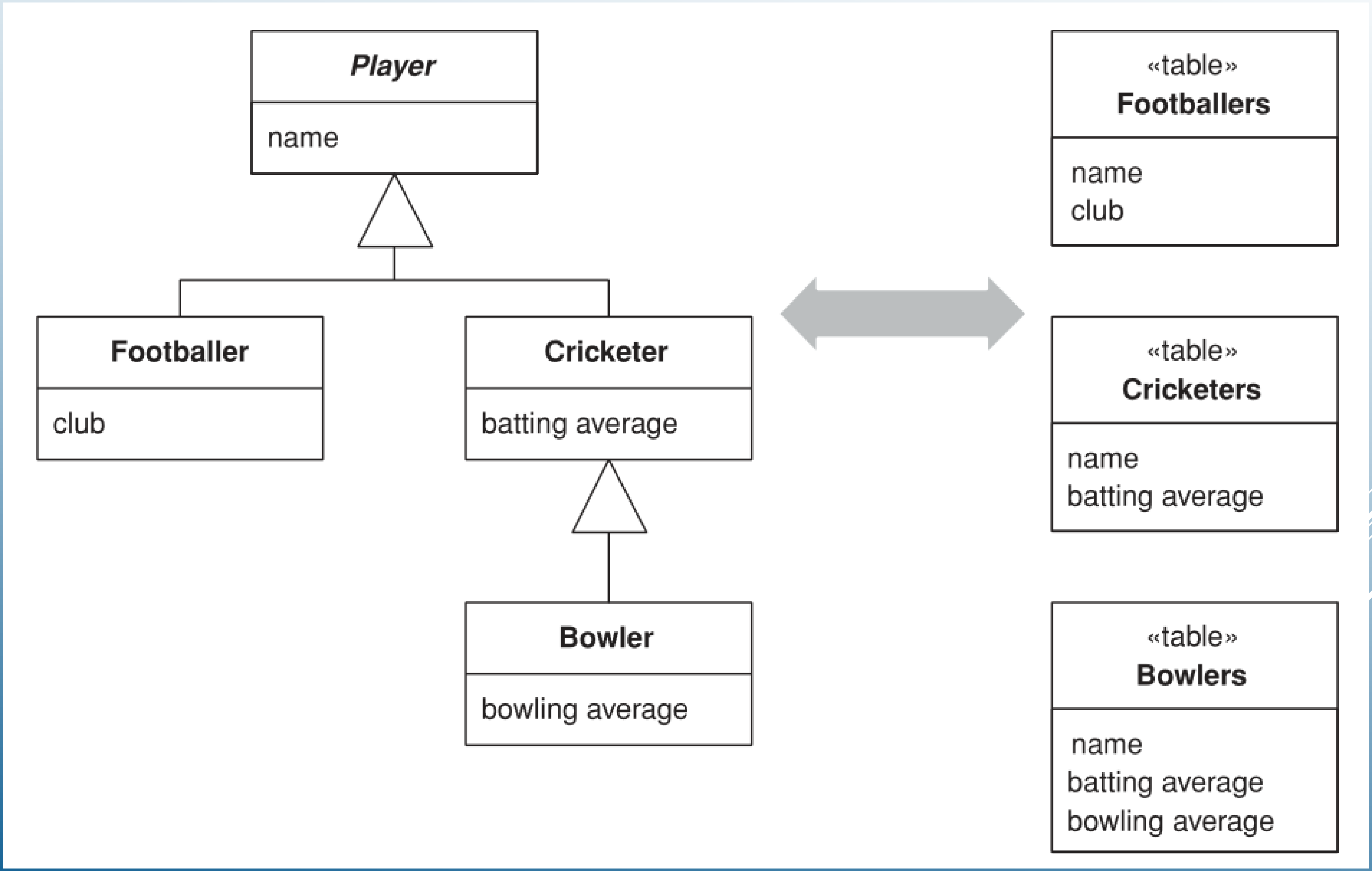
Inheritance mappings

Class Table Inheritance - Represents an inheritance hierarchy of classes with one table for each class.



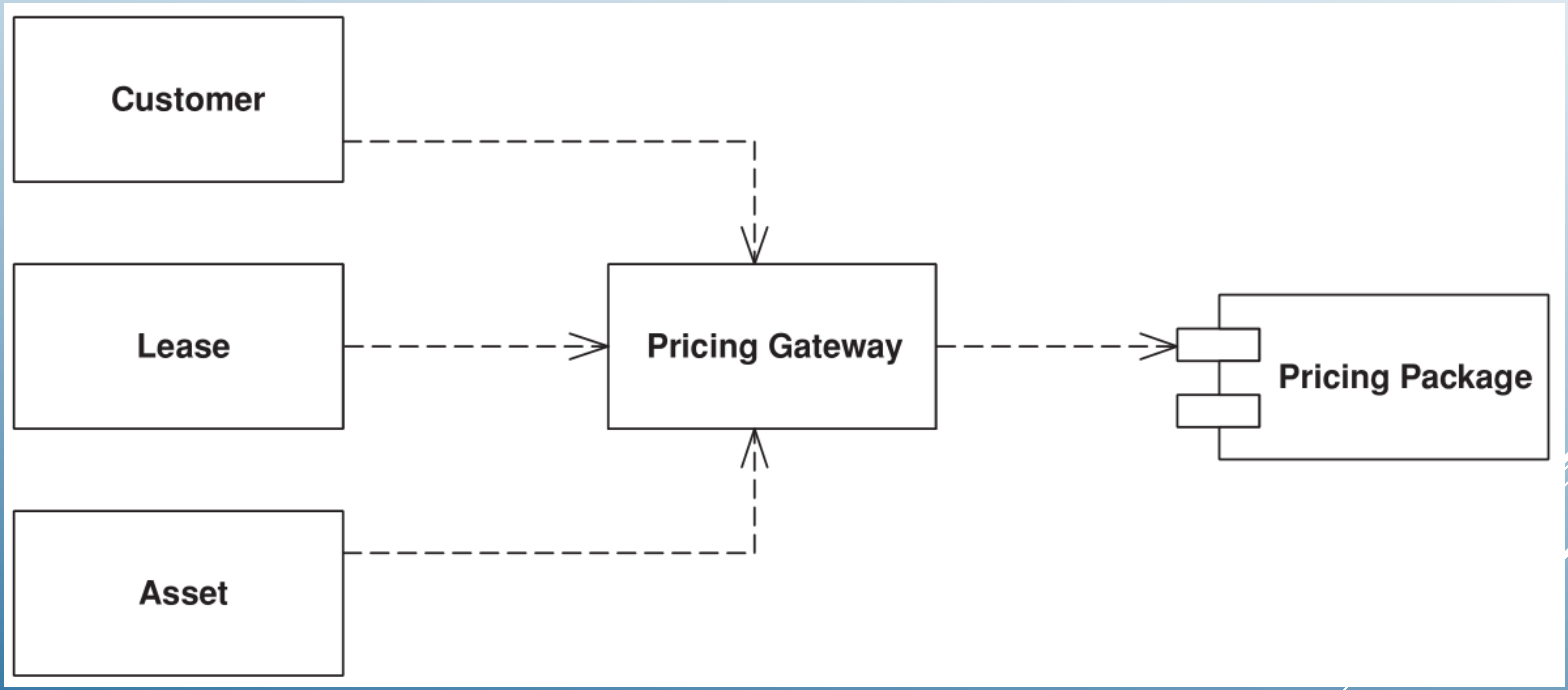
Inheritance mappings

Concrete Table Inheritance - Represents an inheritance hierarchy of classes with one table per concrete class in the hierarchy.



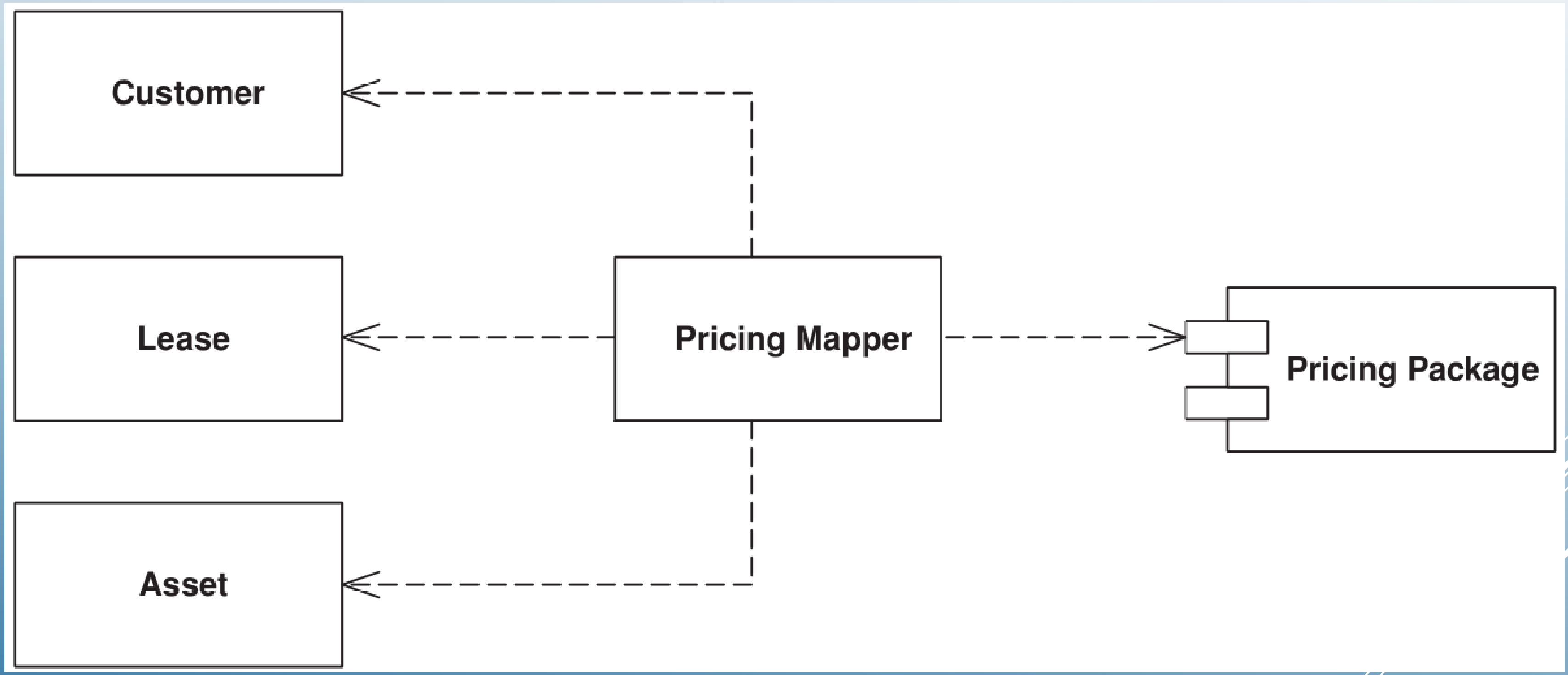
General Patterns

Gateway - An object that encapsulates access to an external system or resource. (Table data gateway). Wrap all the special API code into a class whose interface looks like a regular object.



General patterns

Mapper - An object that sets up a communication between two independent objects. (Data mapper)

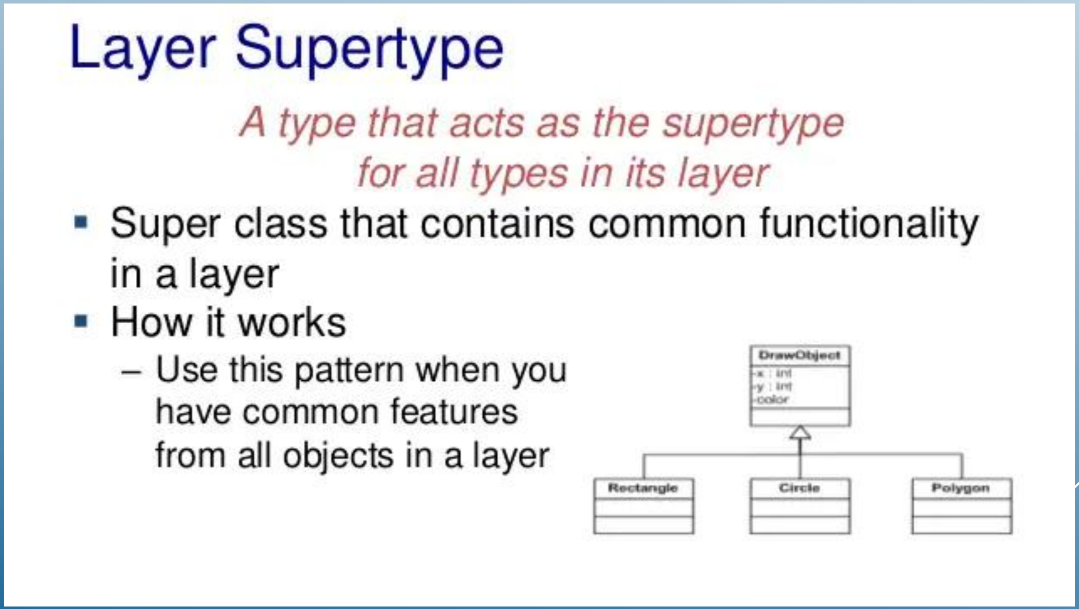


Layer supertype

- A type that acts as the supertype for all types in its layer.
- Use Layer Supertype when you have common features from all objects in a layer.

Examples:

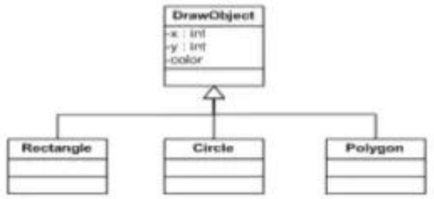
- Abstract class for Identity Field pattern
- Abstract class for common behavior of objects of the Data pattern Mapper
- Domain Object superclass for all the domain objects in a Domain Model - common features



Layer Supertype

A type that acts as the supertype for all types in its layer

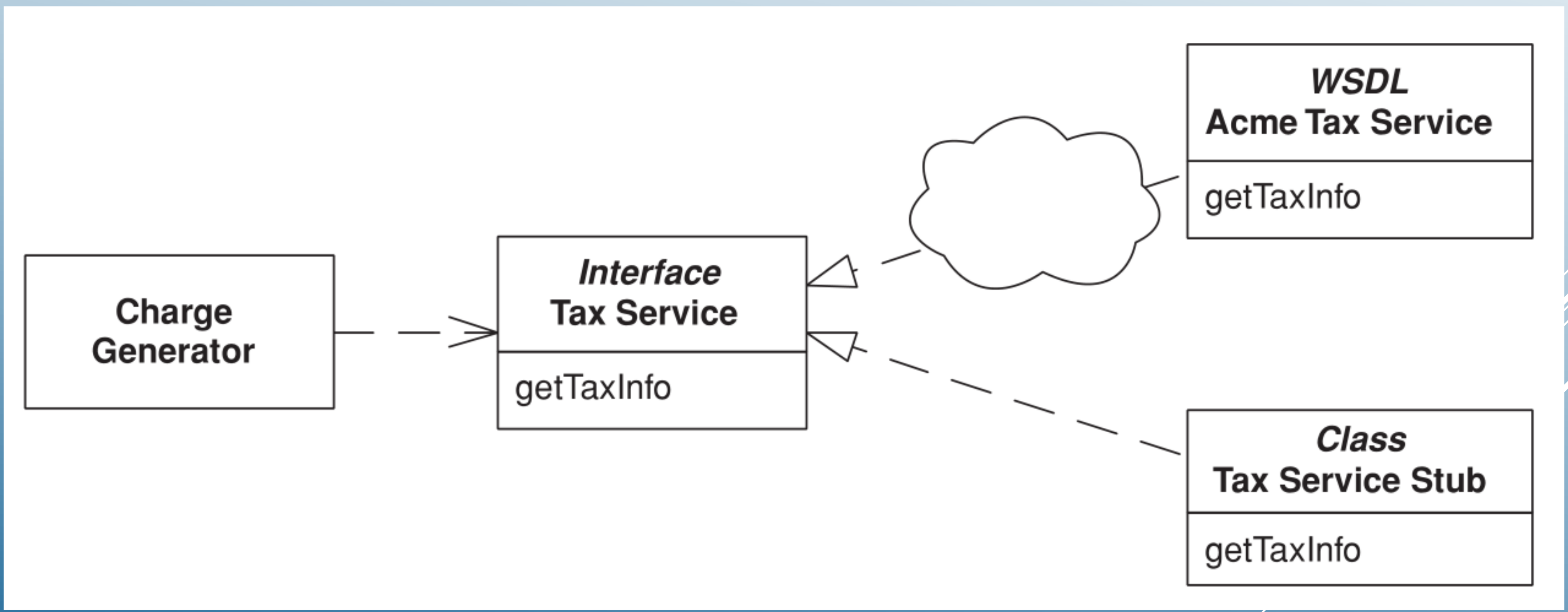
- Super class that contains common functionality in a layer
- How it works
  - Use this pattern when you have common features from all objects in a layer



Service Stub

Removes dependence upon problematic services during testing.

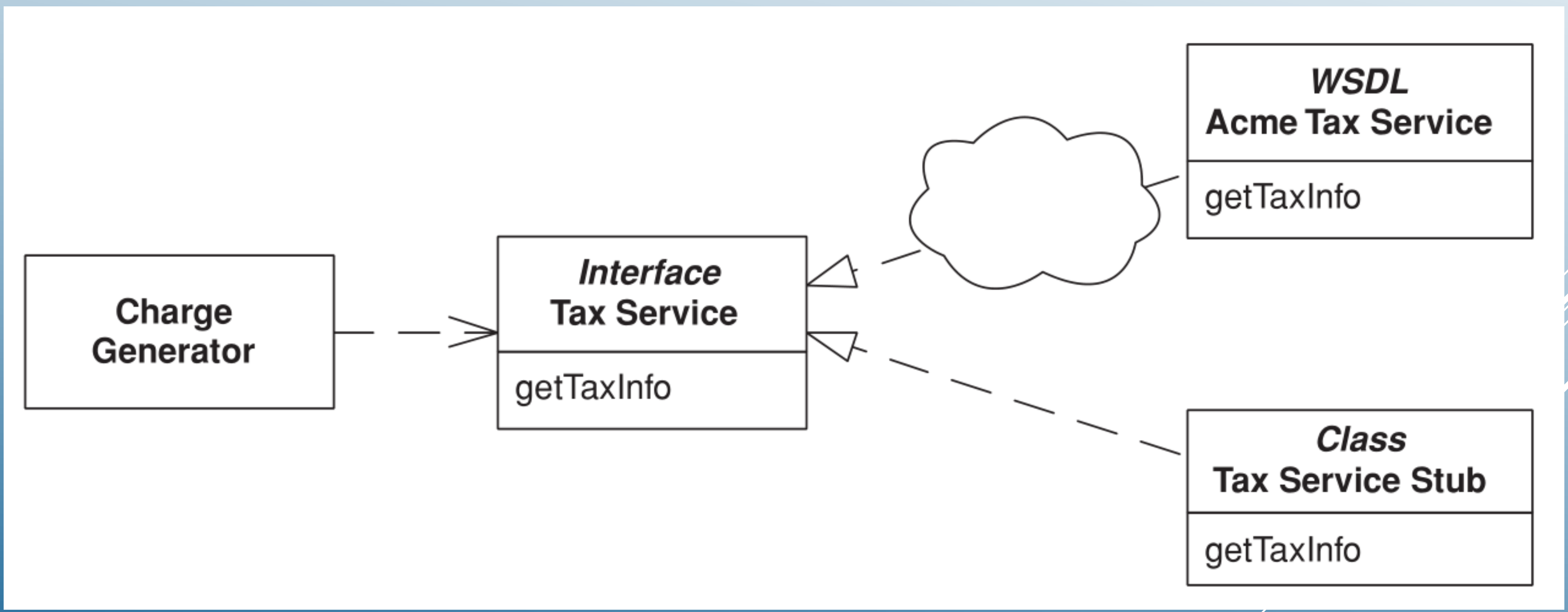
- Define access to the service with a Gateway
- The Gateway should not be a class but rather a Separated Interface so you can have one implementation that calls the real service and at least one that's only a Service Stub



Service Stub

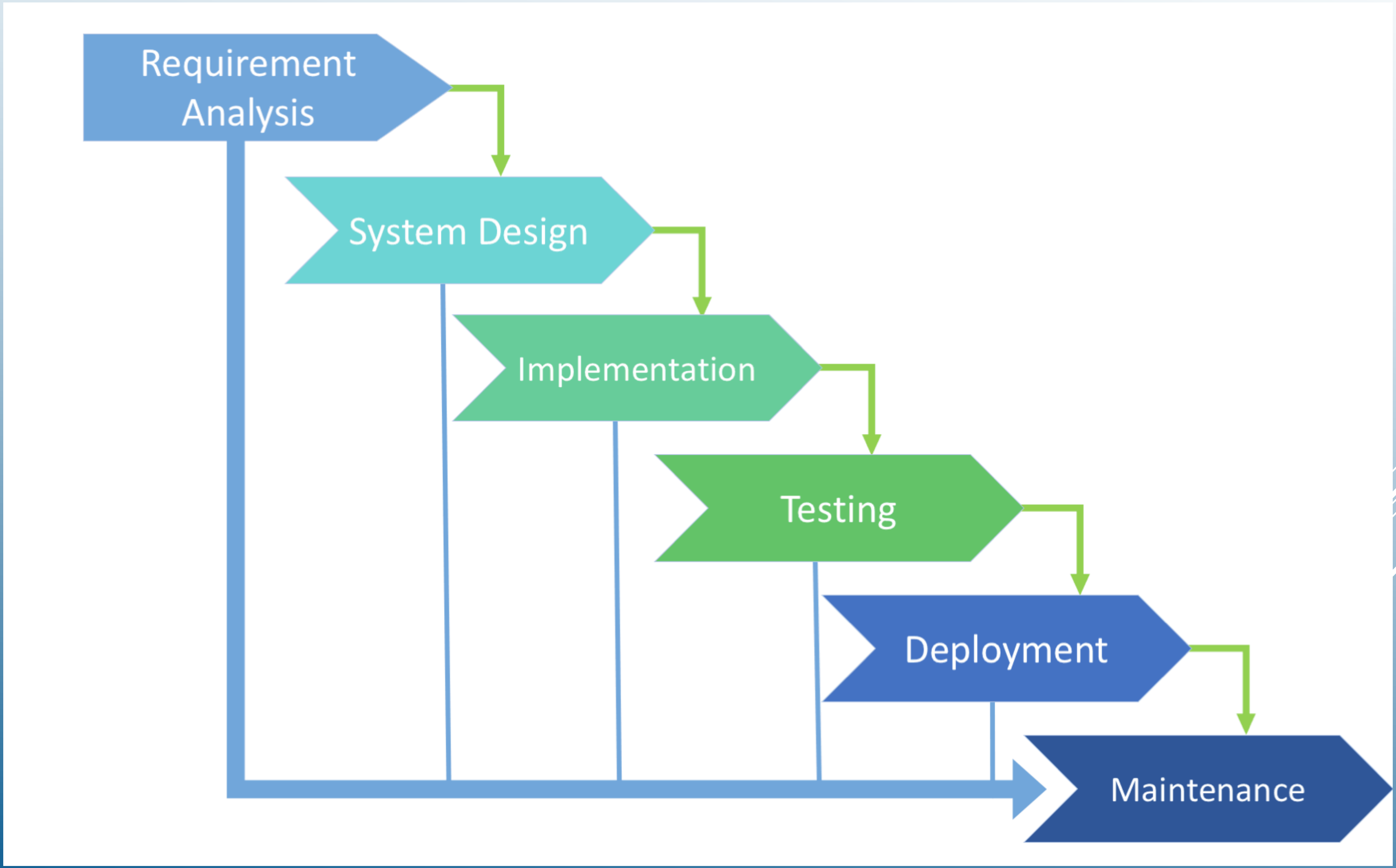
Removes dependence upon problematic services during testing.

- Define access to the service with a Gateway
- The Gateway should not be a class but rather a Separated Interface so you can have one implementation that calls the real service and at least one that's only a Service Stub



Software Development

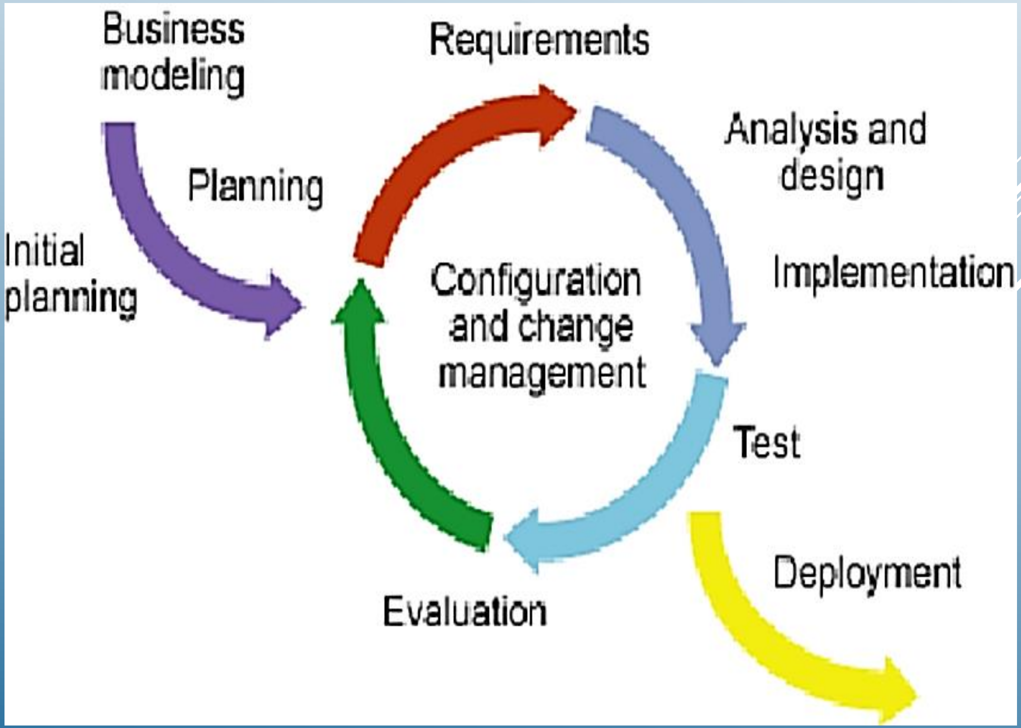
Waterfall The first and most traditional of the plan-driven process models and addresses all of the standard life cycle phases.



Software Development

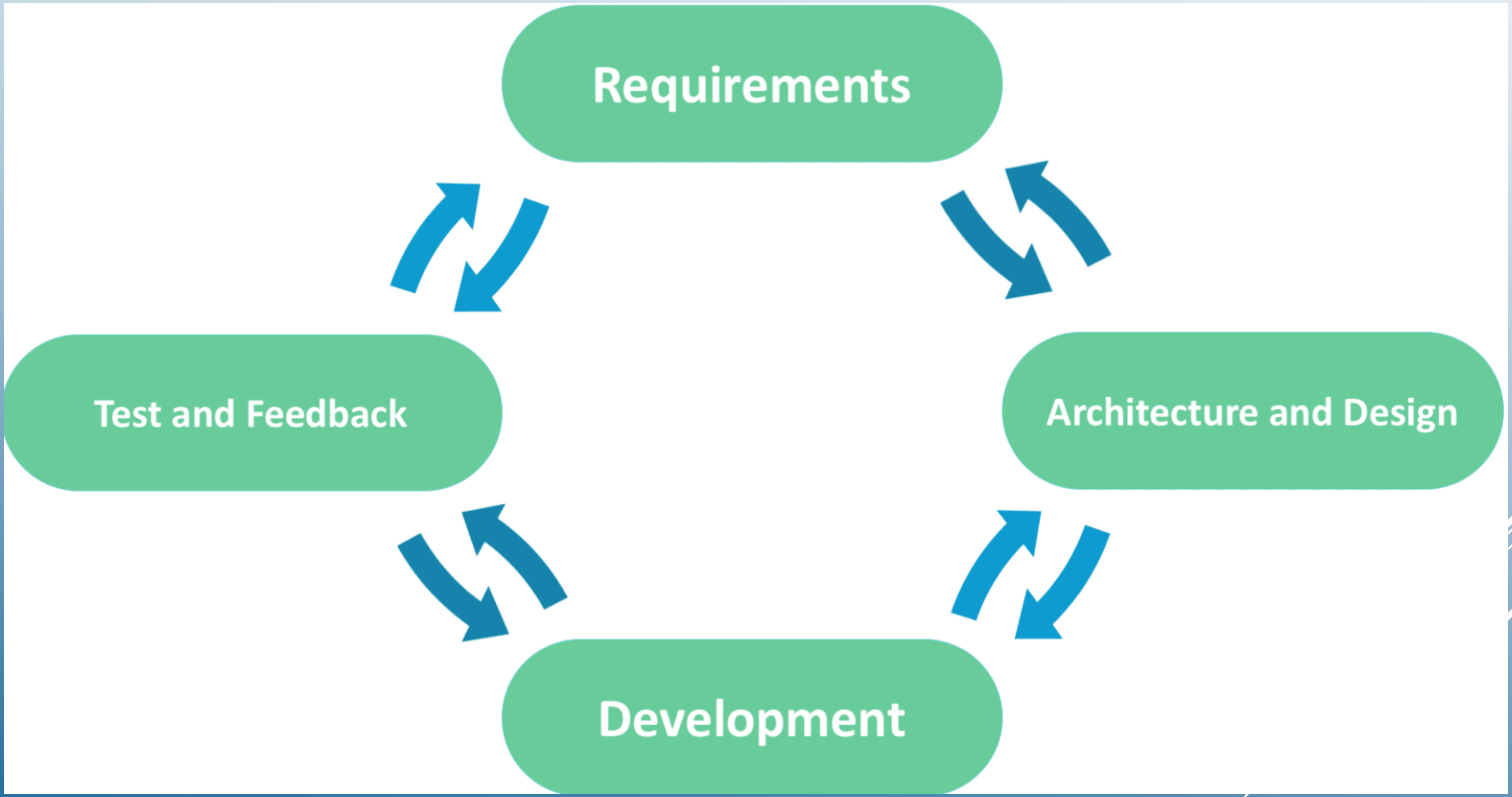
Iterative and incremental development

- Iterative design is a methodology based on an iterative process of analysis, design, implementation (prototyping), testing and redefinition of the product.
- The incremental model is based on the principle of incrementally building a production increments based on iterative design.
- The development combines the features of the waterfall model with those of the iterative prototyping.



Agile development

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- **Customer collaboration over contract negotiation**.
- Responding to change over following a plan.

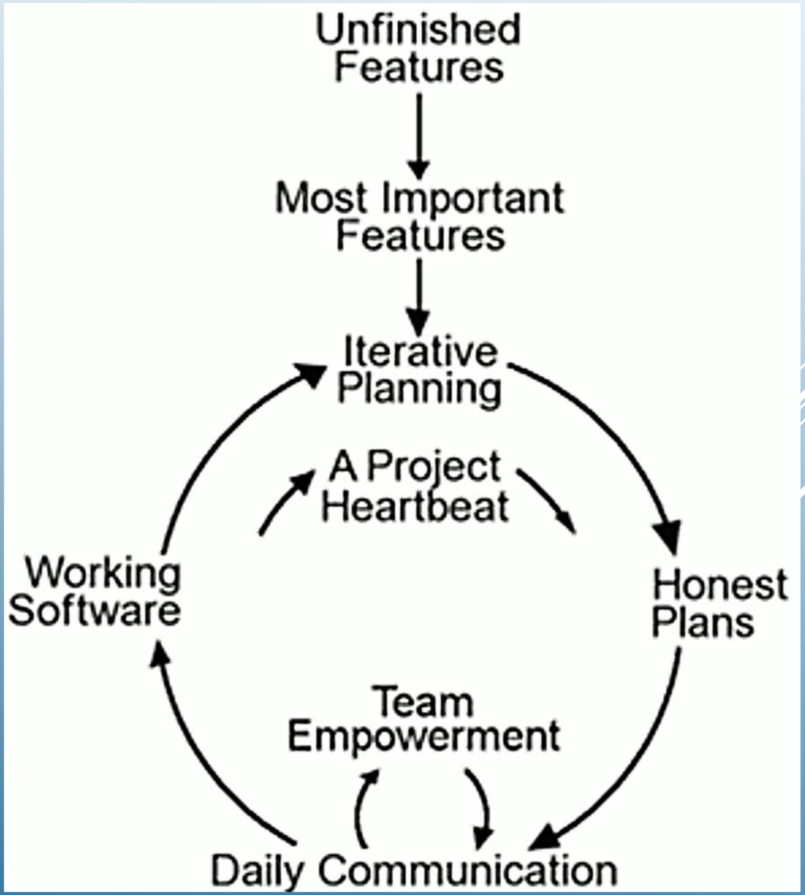


Software Development

Extreme Programming Explained (XP) is one of several popular Agile Proces.
XP is successful because it stresses customer satisfaction. Instead of delivering everything you could possibly want on some date far in the future this process delivers the software you need as you need it.

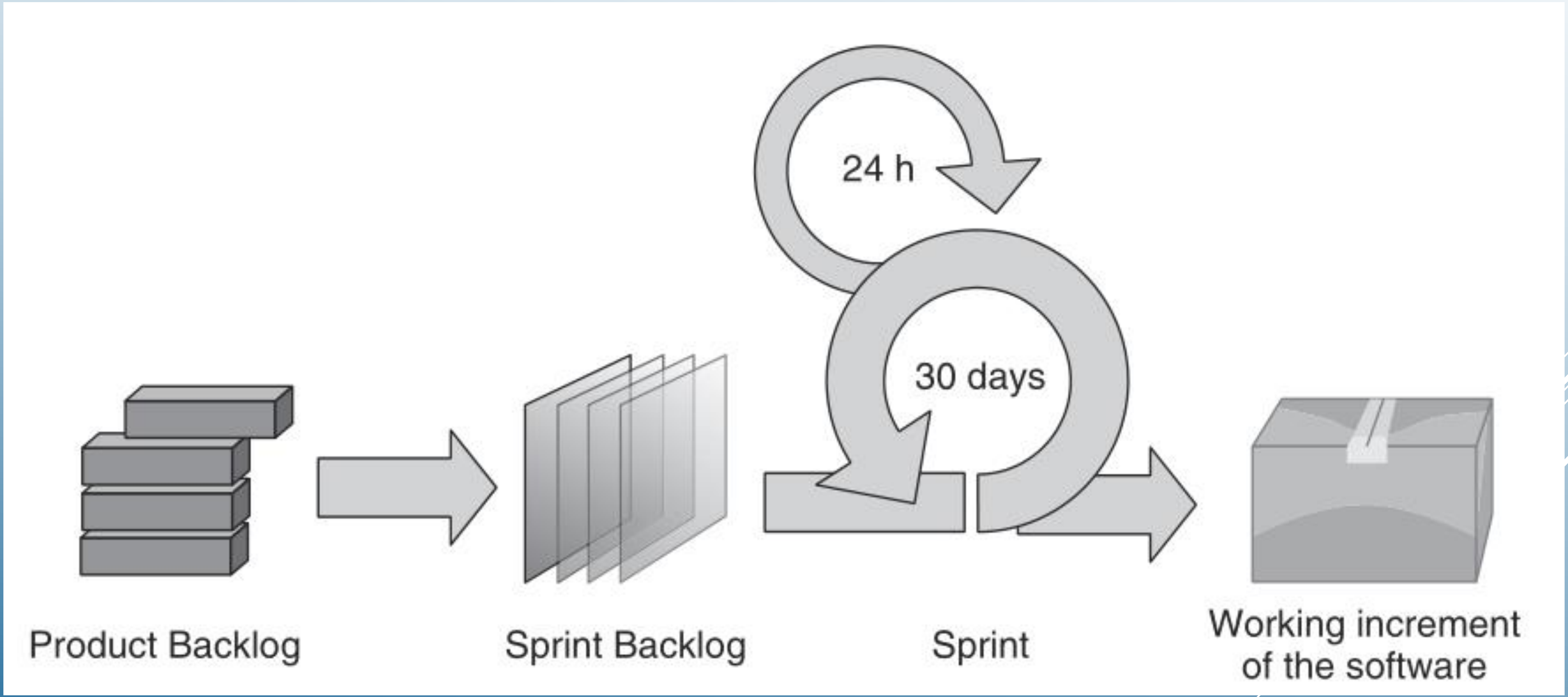
5 values of XP:

- Communiction
- Simplicity
- Feedback
- Courage
- Respect



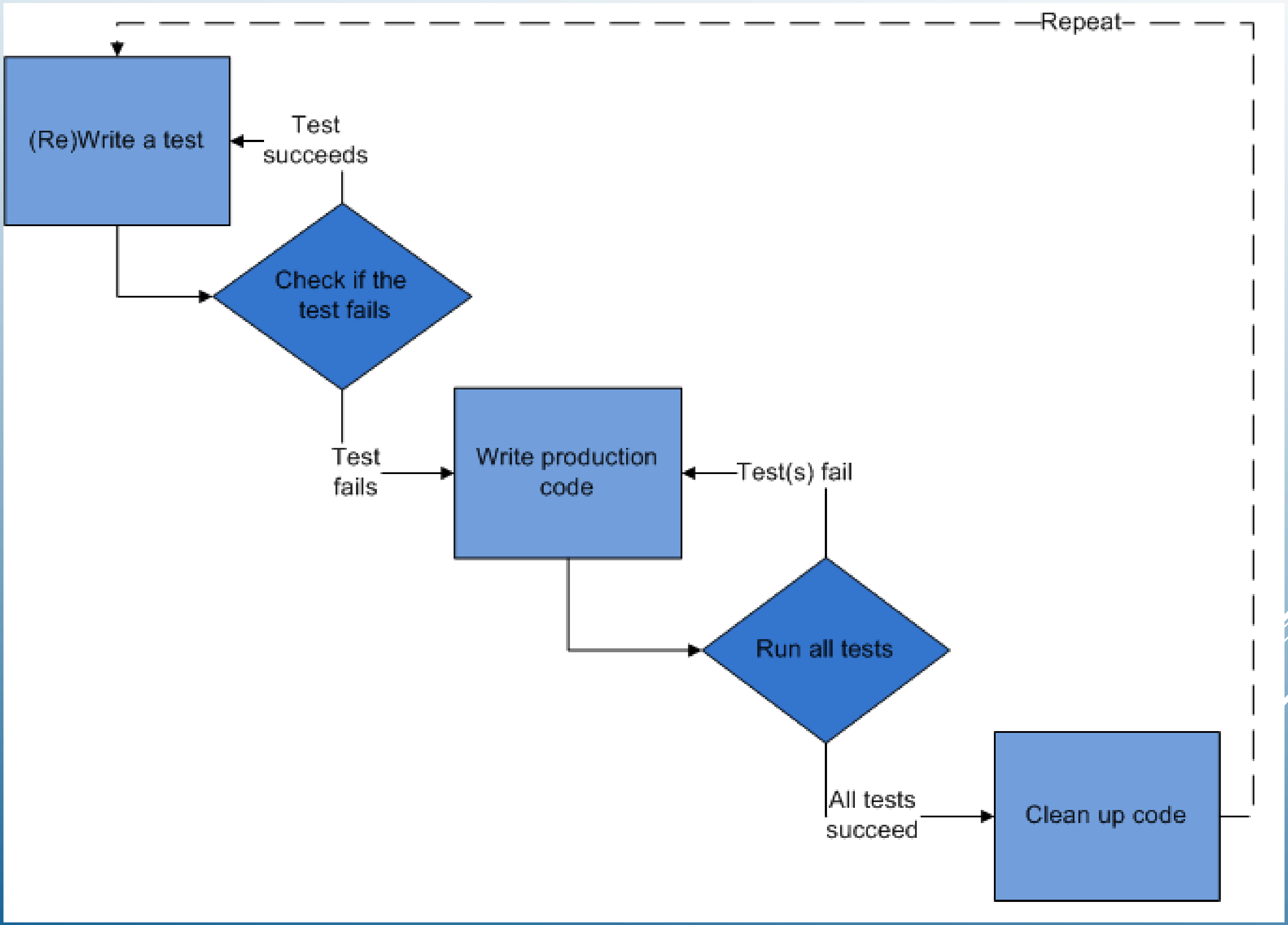
Software Development

The **Scrum** model is a framework for planning and conducting software projects based on the principles of Agile development



Software Development

Test Driven Development - refers to a style of programming in which three activities are tightly interwoven: coding, testing (in the form of writing unit tests) and design (in the form of refactoring).



Review of the last lecture

Domain Specific Languages

In general a DSL construct a shared language between the domain people and programmers.

Why DSL?

- Improving productivity for developers.
- Improving communication with domain experts.
- DSL is a laguage facade over a library or framework

Domain Specific Languages - examples

- **SQL**
- **HTML**
- **A programming language** "library"
  - E.g. **DateTime** and support for date and time tasks.
- **UML diagram** of a State machine.
- **Regular expression**
- **Configuration file, XAML**

Categories of Domain Specific Languages

- **External DSL** - Language separate from the main language of the application it works with. Usually, has a custom syntax, but using another language's syntax is also common (e.g. XML config files, SQL, regular expressions).
- **Internal DSL** - A particular way of using a general-purpose language. A script in an internal DSL is valid code in its general-purpose language, but only uses a subset of the language's features in a particular style to handle one small aspect of the overall system. (Ruby, Lisp)
- **Language workbench** - A specialized IDE for defining and building DSLs. Is used not just to determine the structure of a DSL but also as a custom editing environment for people to write DSL scripts.

Benefits of using DSLs

- **Productivity** - You can replace a lot of GPL code with a few lines of DSL code.
- **Quality** - fewer bugs, better architectural conformance, increased maintainability.
- **Validation and Verification** - programs are more semantically rich than GPL programs. Analyses are much easier to implement,
- **Data Longevity** - models are independent of specific implementation techniques.
- **Productive Tooling** – external DSLs can come with tools, i.e. IDEs that are aware of the language.

Problems of DSL

- **Language Cacophony**
  - Using many languages will be much more complicated than using a single one.
- **Cost of Building**
  - The cost of a DSL is the cost over the cost of building the model.
- **Ghetto Language**
  - Company that's built a lot of its systems on an in-house language which is not used anywhere else.
- **Blinkered Abstraction (Tunnel vision)**
  - DSL It allows you to express the behavior of a domain much more easily than if you think in terms of lower-level constructs.

Final test information

The final test will take place in January 2022

The exact date of the final test will be published in the school information system within the next 14 days.

The final test is a written exam with 6 open questions.

The test questions cover the areas of IS development presented in the lectures and questions from the lectures.

The maximum score for the final test is 55 points. A score of at least 50% is required to pass the final test.