Vývoj informačních systémů

Architektura, návrh
Vzory: Doménová logika

2021-22

Zachman Framework

	DATA *What*	FUNCTION *How*	NETWORK *Where*	PEOPLE *Who*	TIME *When*	MOTIVATION *Why*
Objective/Scope (contextual) *Role: Planner*	List of things important in the business	List of Business Processes	List of Business Locations	List of important Organizations	List of Events	List of Business Goal & Strategies
Enterprise Model (conceptual) *Role: Owner*	Conceptual Data/ Object Model	Business Process Model	Business Logistics System	Work Flow Model	Master Schedule	Business Plan
System Model (logical) *Role:Designer*	Logical Data Model	System Architecture Model	Distributed Systems Architecture	Human Interface Architecture	Processing Structure	Business Rule Model
Technology Model (physical) *Role:Builder*	Physical Data/Class Model	Technology Design Model	Technology Architecture	Presentation Architecture	Control Structure	Rule Design
Detailed Reprentation (out of context) *Role: Programmer*	Data Definition	Program	Network Architecture	Security Architecture	Timing Definition	Rule Speculation
Functioning Enterprise *Role: User*	Usable Data	Working Function	Usable Network	Functioning Organization	Implemented Schedule	Working Strategy

Zdroje

- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1995.
  - Návrh programů pomocí vzorů - Stavební kameny objektově orientovaných programů. 2003. [GoF]
- Martin Fowler. *Patterns of Enterprise Application Architecture*. 2003.
- David Trowbridge, Dave Mancini,Dave Quick, Gregor Hohpe, James Newkirk, David Lavigne. *Enterprise Solution Patterns Using Microsoft .NET. 2003.*
- Adam Bien. *Real World Java EE Patterns – Rethinking Best Practices*. 2012.

Gang of Four (1994, 1995)

- Obecné principy tvorby software
- 23 vzorů (C++, Smalltalk)
- Creational patterns
- Structural patterns
- Behavioral patterns

Vzory jako připravené návody

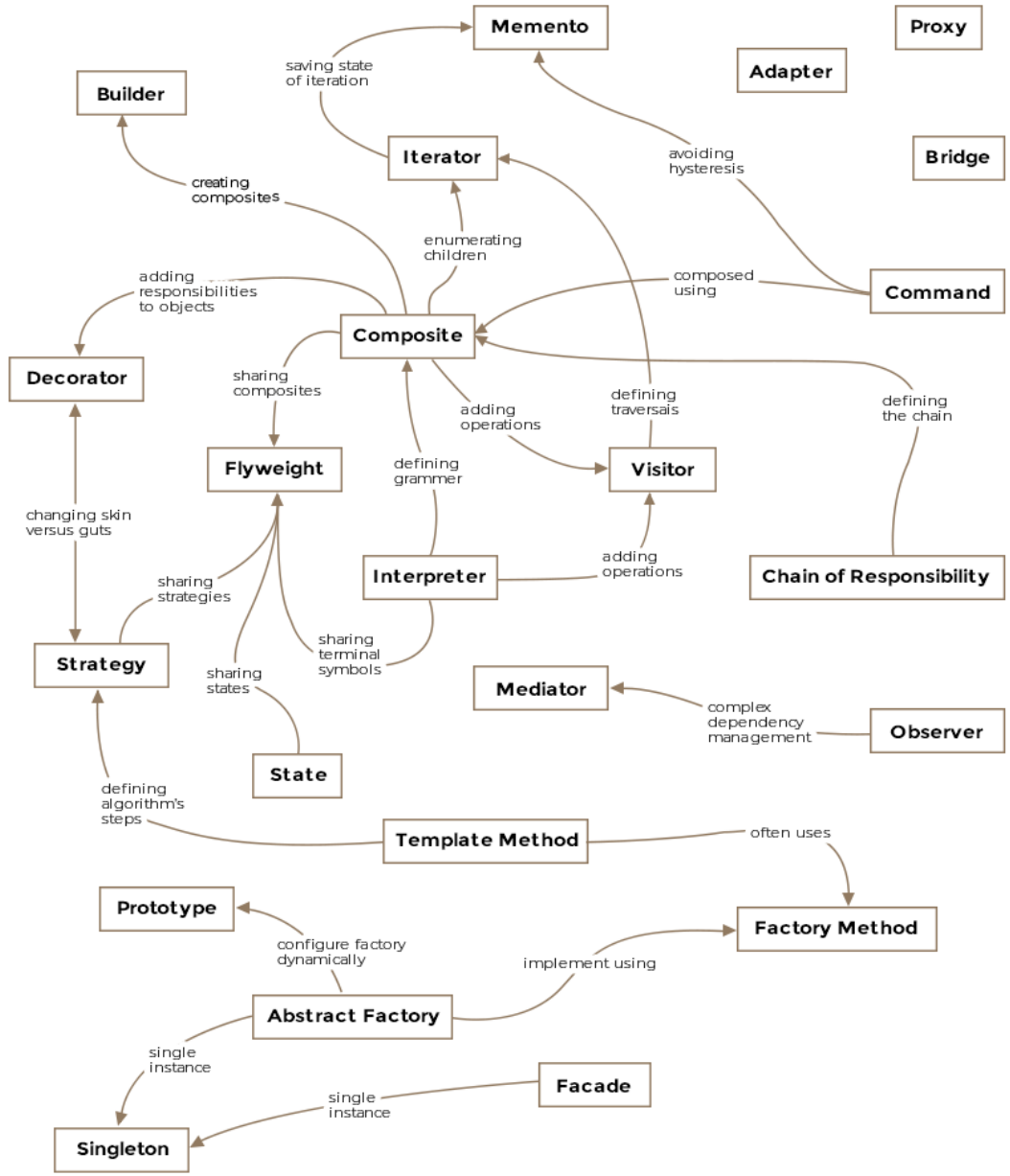
- GoF je pouze začátek.
- Vzory a jazyky vzorů.
- Vzor je to, co opakovaně funguje.
- Existují i antivzory (opakovaně nefungují ☹).

Struktura vzorů

- Výstižné jméno (název)
- Problém (Co)
- Souvislosti (Kdo, Kdy, Kde, Proč)
- Řešení včetně UML diagramů (Jak)
- Příklady včetně zdrojových kódů

GoF

Gang of Four patterns	
Creational	• Abstract factory, Builder, Factory method, Prototype, Singleton
Structural	• Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy
Behavioral	• Chain of responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template method, Visitor

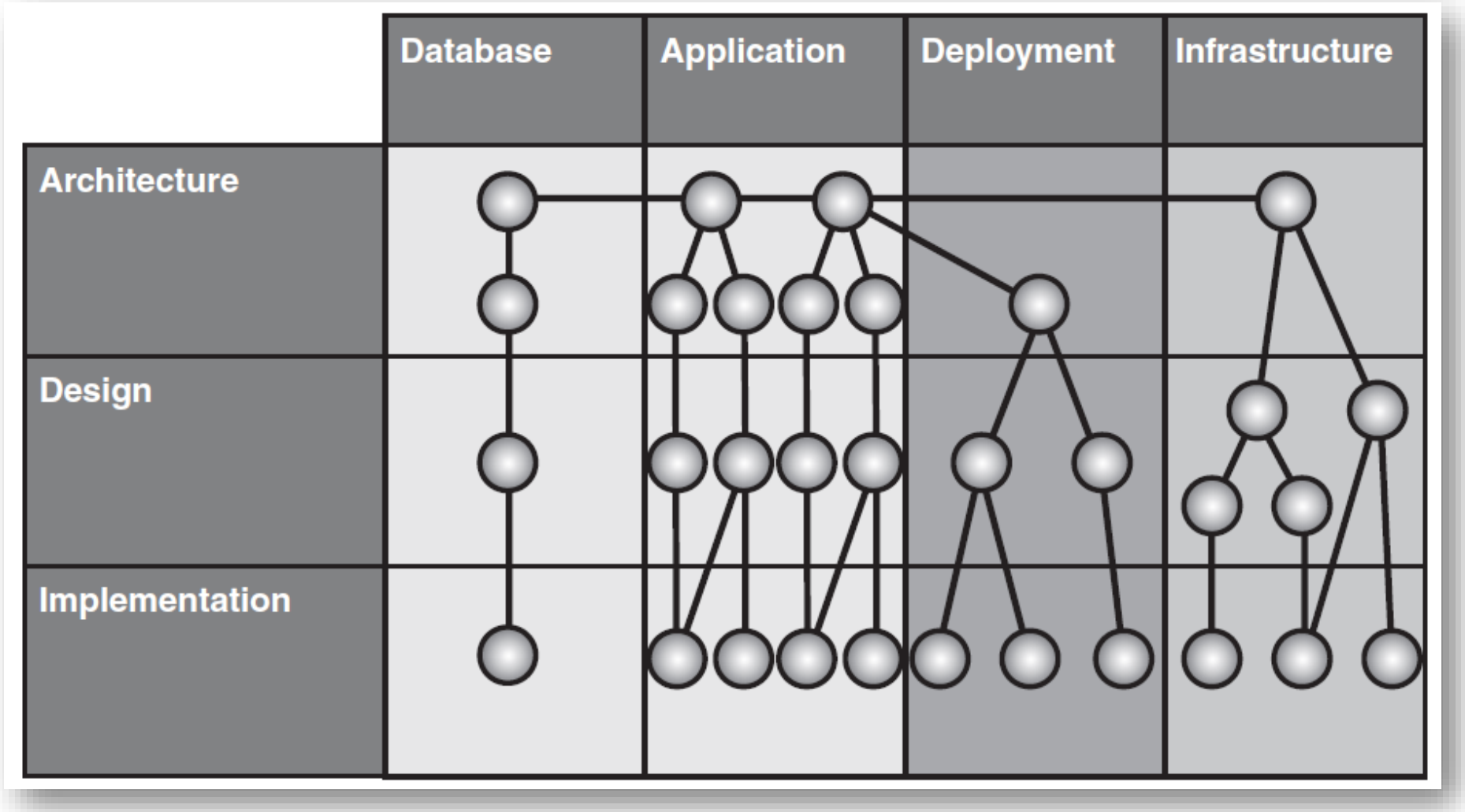


Design Pattern Relationships

Fowler (EAA, 2003)

- Různě rozsáhlá řešení formou katalogu
- Více než 50 vzorů (Java, C#).
- Domain Logic Patterns, Data Source Architectural Patterns, Object-Relational Behavioral Patterns, Object-Relational Structural Patterns, Object-Relational Metadata Mapping Patterns, Web Presentation Patterns, Distribution Patterns, Offline Concurrency Patterns, Session State Patterns, Base Patterns
- Další na http://martinfowler.com/eaaDev/

Pohled architekta, návrháře, vývojáře



Three-Layered Application

Context

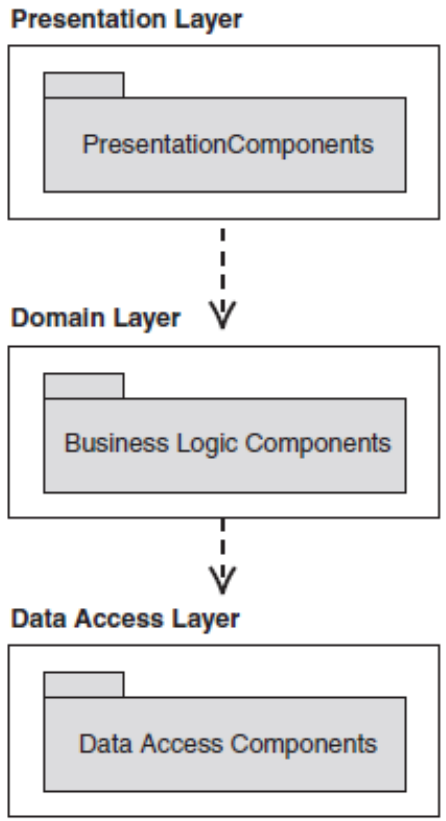
You are building a business solution using layers to organize your application

Problem

How do you organize your application to reuse business logic, provide deployment flexibility and conserve valuable resource connections?

Solution

Create three layers: presentation, business logic (domain), and data access. Place all components responsible for the view in the presentation layer. Encapsulate all business logic in domain layer components that implement well-known component interfaces. Locate all database-related code, including database client access and utility components, in the data access layer. Require the data access layer to be responsible for connection pooling when accessing resources. Make sure you eliminate the dependencies between data access components and business layer components. Either eliminate dependencies between the business layer and the presentation layer or manage the dependencies here using the *Observer* pattern.



Three-Layered Services Application

Context

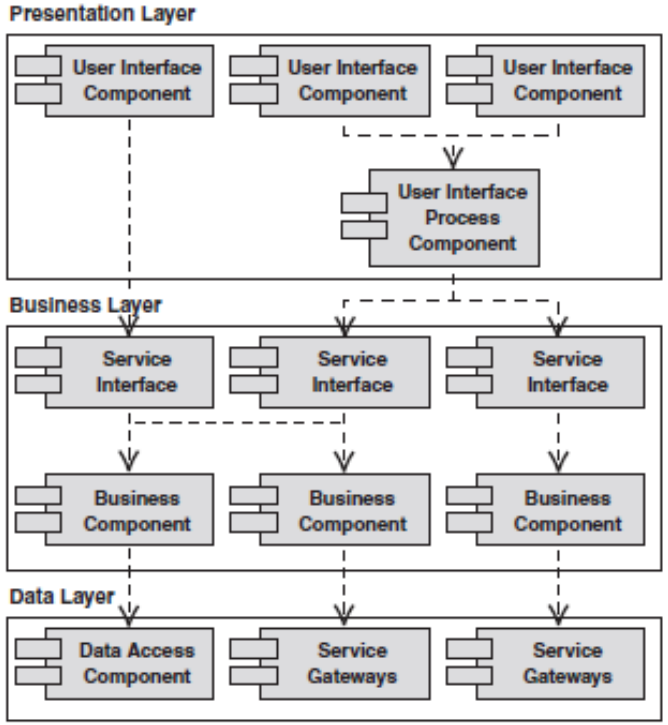
You are building a business solution that uses presentation, business, and data access layers to organize your application. You want to expose some of the core functionality of your application as services that other applications can consume and enable your application to consume other services.

Problem

How do you organize your application to provide and consume granular data and logical elements from highly variable sources?

Solution

Decompose your application logic into a collaborating set of services that provide parts of the overall system functionality. Next, in the domain layer, identify a Service Interface for each service that is independent of the underlying implementation. Finally, extend the data access layer to use Service Gateways to communicate with other service providers. If application navigation logic is sufficiently complex, consider user interface process components as part of the presentation layer to encapsulate and reuse this logic.

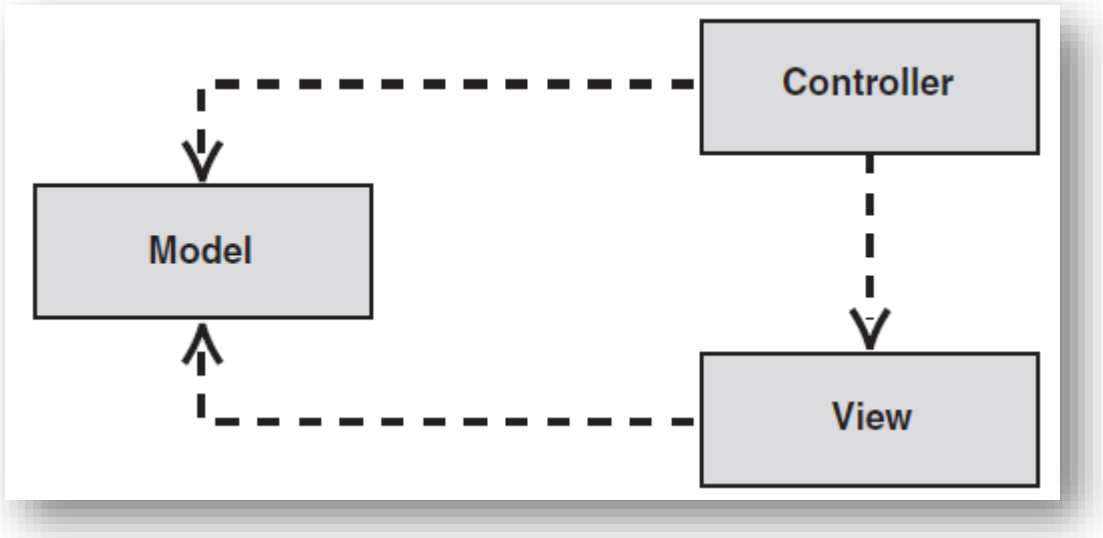


Třívrstvá architektura

- Three-Layer Architecture
- Three-Tier Architecture
- První se stává druhou, pokud je fyzicky oddělena od toho, kdo ji používá…
- Prezentace, doménová logika, přístup k datům, servisní vrstvy.
- Klient – server, p-vrstvá architektura.

MVC – Model View Controller

- Klíčový koncept návrhu (Trygve Reenskaug, Smalltalk-76)
- Oddělení vrstev na logické úrovni
- Minimalizace závislostí, izolovaná modifikace jednotlivých vrstev



MVC - nedorozumění

- Obvykle vůbec neřeší přístup k datům (ve smyslu přístupu k databázi).
- Existují variace pro různé platformy a situace.
- Je nutno chápat jako velmi obecný (a správný) návrhový/architektonický koncept.
- Neplést si s třívrstvou architekturou (která je lineární).

Vzory blízké MVC

- *Observer* a *Data Binding* (řešení založená na událostech)
- *Presentation Model* a jeho varianta *Model View ViewModel* (Microsoft)
- Model View Presenter (*Supervising Controller* a *Passive View*)
- *Page Controller* a *Front Controller* (středně složitá a složitá řešení pro web)

Vzory pro „enterprise“ architekturu

- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. 1995.
  - (Návrh programů pomocí vzorů - Stavební kameny objektově orientovaných programů. 2003.

- **Martin Fowler. Patterns of Enterprise Application Architecture. 2003.**

- David Trowbridge, Dave Mancini,Dave Quick, Gregor Hohpe, James Newkirk, David Lavigne. Enterprise Solution Patterns Using Microsoft .NET. 2003.

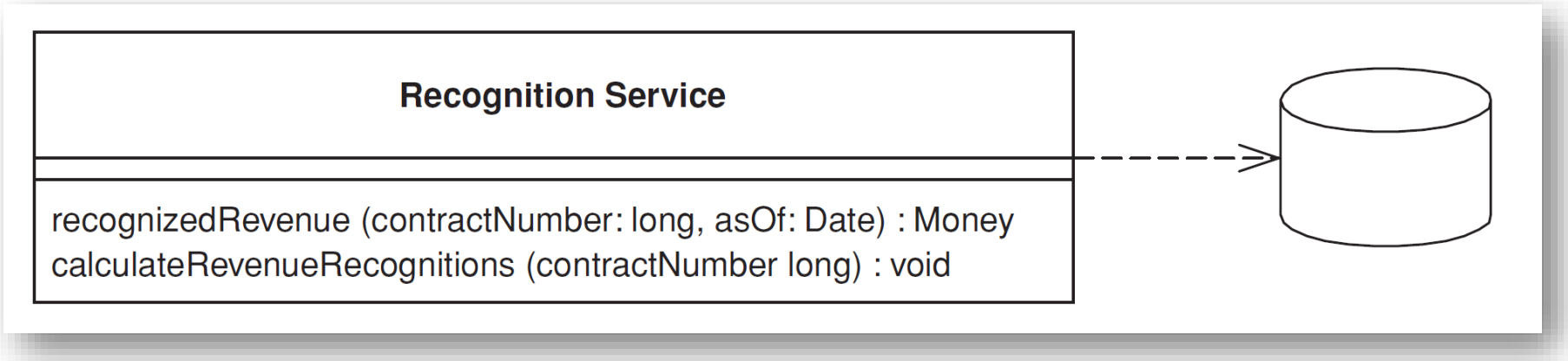
- Adam Bien. Real World Java EE Patterns – Rethinking Best Practices. 2009.

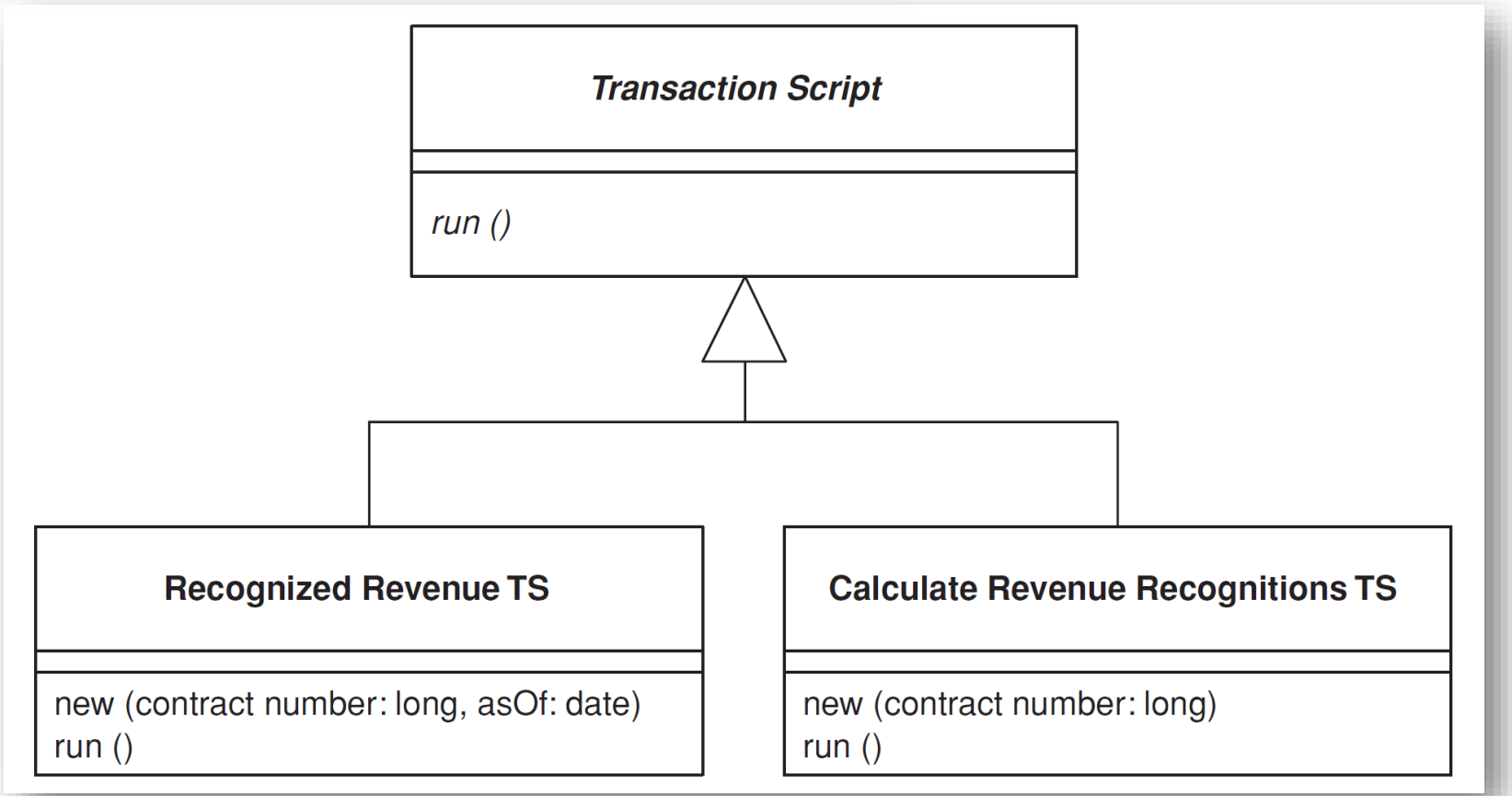
Vzory pro doménovou logiku

- Transaction script
- Domain model
- Table module
- Service layer

Transaction Script

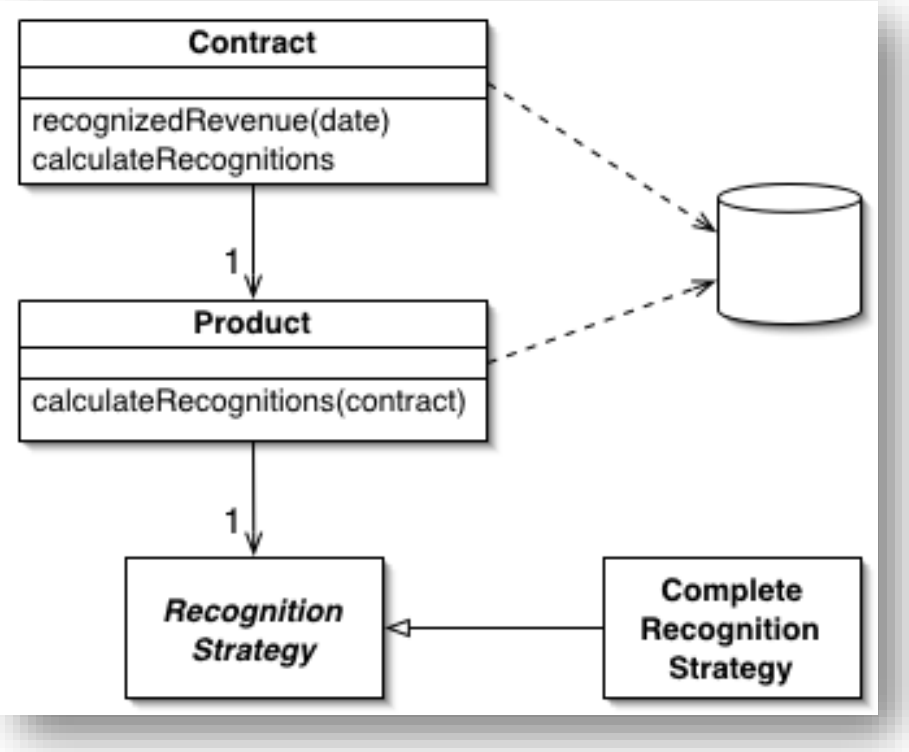
- Organizes business logic by procedures where each procedure handles a single request from the presentation.


Recognition Service
recognizedRevenue (contractNumber: long, asOf: Date) : Money
calculateRevenueRecognitions (contractNumber long) : void


Transaction Script
run ()
Recognized Revenue TS
new (contract number: long, asOf: date)
run ()
Calculate Revenue Recognitions TS
new (contract number: long)
run ()

Domain Model

- An object model of the domain that incorporates both behavior and data.



Dva různé styly pro doménový model

- **Jednoduchý** – pracuje s jednoduchými logikami a třídy odpovídají tabulkám v databázi

- **Úplný** – pracuje s komplexními logikami a model tříd se liší od databázového modelu

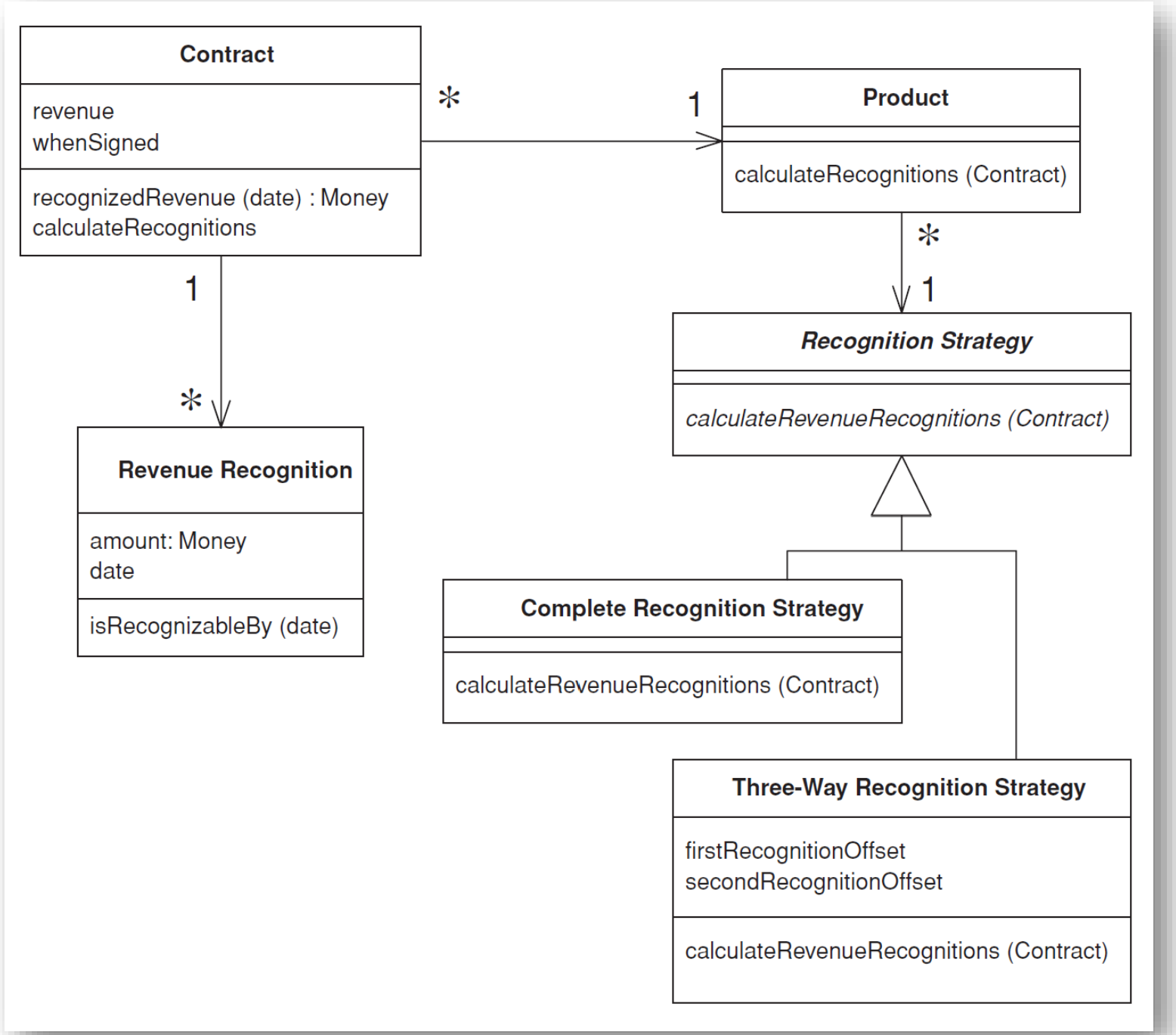

Contract
revenue
whenSigned
recognizedRevenue (date) : Money
calculateRecognitions
*
1
Product
calculateRecognitions (Contract)
*
1
1
*
Recognition Strategy
calculateRevenueRecognitions (Contract)
Revenue Recognition
amount: Money
date
isRecognizableBy (date)
Complete Recognition Strategy
calculateRevenueRecognitions (Contract)
Three-Way Recognition Strategy
firstRecognitionOffset
secondRecognitionOffset
calculateRevenueRecognitions (Contract)

Table Module

- A single instance that handles the business logic for all rows in a database table or view.

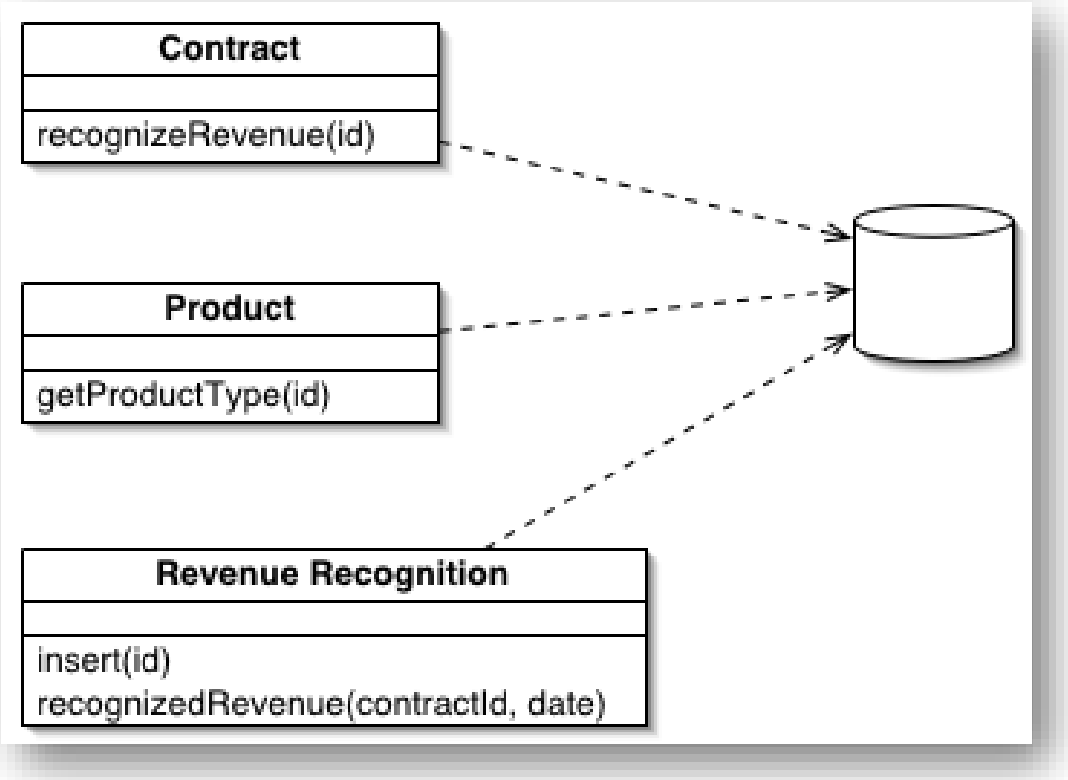
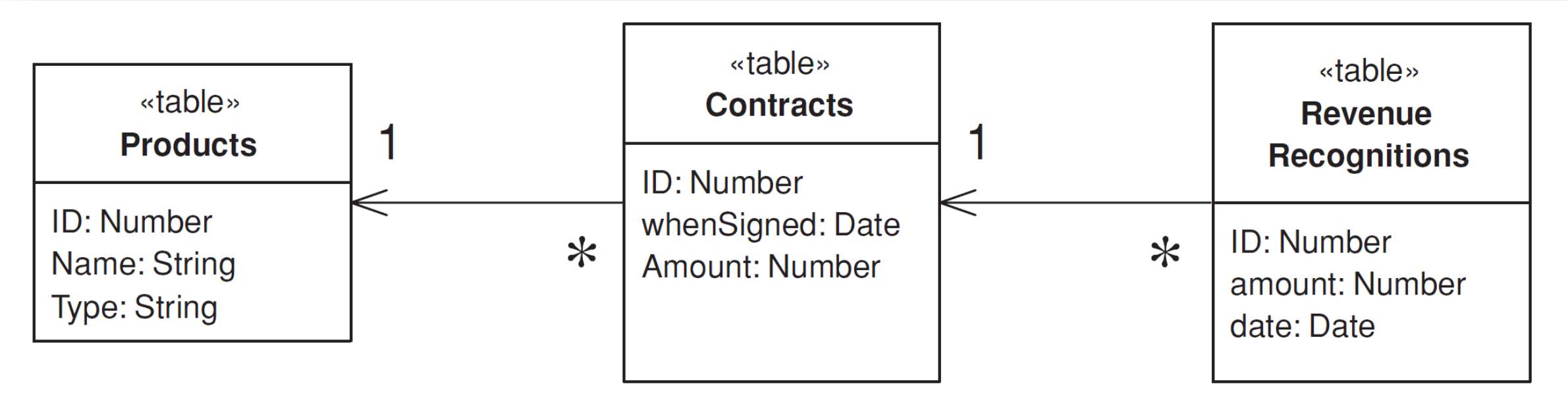


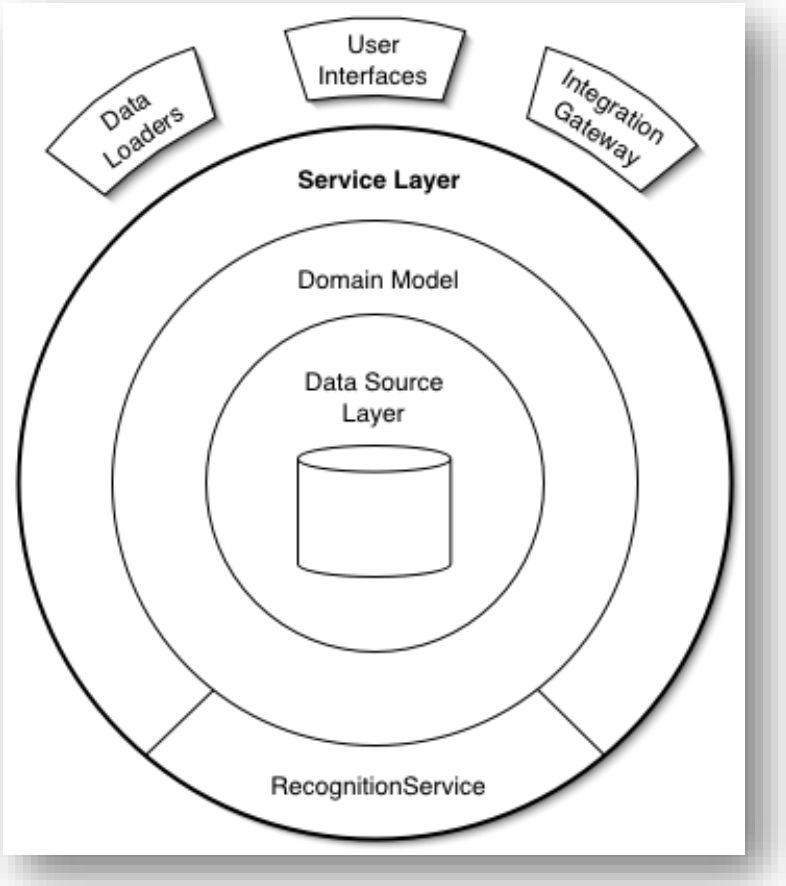
Schéma pro *Table Module*



- Jedna instance třídy dle vzoru Table Module odpovídá jedné tabulce v DB.
  - anEmployeeModule.getAddress(long employeeID)

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.

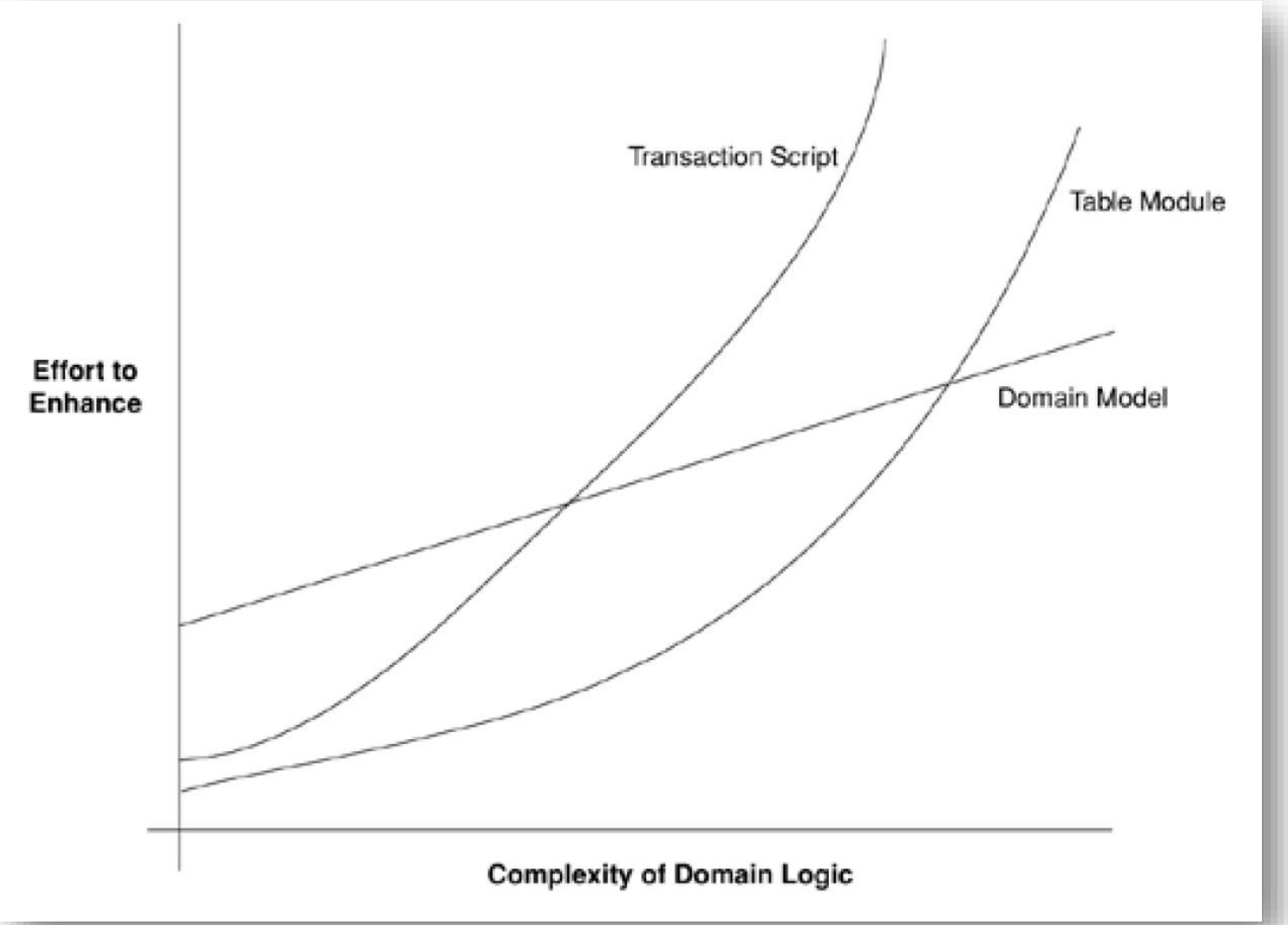


Service Layer ukrývá byznys logiku

- **Doménová logika**, která se týká problémové oblasti (např. výpočetních strategií)

- **Aplikační logika**, která souvisí s odpovědnostmi informačního systému (zajištění tzv. workflow).

Který a kdy použít?



Projekt: Specifikace požadavků

- Funkční požadavky ve formě use-case modelu
- Use-case diagram (celý systém)
- Use-case (strukturovaný popis pro 3 scénáře)
- Diagram aktivit (po jednom k use-case a přes use-case)

Úkoly na cvičení

- Re-implementuje úkol z předchozího cvičení podle vzorů z přednášky (Transakční skript, Doménový model, Modul pro tabulku).

Kontrolní otázky

1. Co se rozumí pojmem návrhový vzor? Co každý vzor obsahuje? Uveďte příklady.

2. Popište podstatu třívrstvé architektury. Jaký je rozdíl mezi fyzickou a logickou třívrstvou architekturou?

3. Co je podstatou vzoru MVC? V čem se liší od třívrstvé architektury?

4. Co řeší skupina návrhových vzorů pro doménovou logiku?

5. Popište podstatu jednotlivých vzorů pro doménovou logiku (Transaction script, Domain model, Table module, Service layer).

6. V čem se od sebe liší jednotlivé vzory pro doménovou logiku? Kdy, kde a proč je použít/nepoužít? Uveďte příklady.

K přečtení…

- Martin Fowler. *Patterns of Enterprise Application Architecture.* Addison-Wesley Professional, 2003 [110-142].