Vypracované kontrolní otázky z VIS

Přednáška 2 - Doména, kompetence

Popište, co se rozumí pojmem informační systém, co řeší a uveďte příklady.

Informační systém je interakce mezi lidmi, procesy a daty. Řeší propojení technologií a lidských aktivit tak, aby podpořil procesy v organizaci. Jeho účelem je zpracovávat informace. Např. bankovní systém, rezervační systém, školní systém.

Popište, co se rozumí pojmem doména informačního systému a uveďte příklady.

Doména je skupina souvisejících věcí z pohledu zákazníka. Do domény patří data, procesy i uživatelé. Např. do domény bankovnictví patří pojmy účet, transakce, kreditní karta, smlouva. Do domény rezervace letenek patří letenka, zavazadlo, sedadlo, letiště.

Na jaké otázky si musíme odpovědět při realizaci informačního systému? Uveďte příklady odpovědí na jednotlivé otázky v kontextu nějakého konkrétního informačního systému.

Příklady pro školní systém.

- Co? Ukládáme informace ve formě dat. Např. jméno žáka, známka žáka, atd.
- Jak? Procesy v systému. Např. zápis docházky.
- Kde? Místa, kde se pracuje se systémem. Např. učitelé přistupují k systému ze školních počítačů, rodiče odkudkoliv pomocí mobilního zařízení.
- Kdo? Role uživatelů. Např. učitel, žák, rodič, ředitel.
- Kdy? Události v systému. Např. učitel je upozorněn na začátku hodiny, aby zapsal docházku.
- Proč? Důvody vzniku systému. Např. zefektivnění komunikace mezi rodiči a učiteli.

Z jakých pohledů (míry abstrakce) se můžeme dívat na informační systém? A v jakých rolích?

- Vizionář - vize, rámec systému.
- Vlastník projektu - business model.
- Architekt - systémový model.
- Návrhář - technologický model.
- Programátor - detailní reprezentace části systému.
- Uživatel, pracovník údržby systému - systém za provozu.

Co se rozumí architekturou informačního systému a co zahrnuje?

Architektura je organizace komponent systému, jejich vzájemných vztahů a vztahů k okolím systému. Zahrnuje:

- Principy návrhu a vývoje.
- Pohled vývojáře na globální strukturu systému a chování, propojení a synchronizaci jeho částí.
- Přístup k datům a toky dat.
- Fyzické rozmístění komponent.

Jaké jsou rozdíly mezi statickou, dynamickou a mobilní architekturou informačního systému? Uveďte příklady.

- Statická - struktura systému je dána návrhem a nemění se za běhu.
- Dynamická - komponenty systému a vazby mezi nimi mohou vznikat a zanikat za běhu systému dle pravidel daných návrhem.
- Mobilní - rozšiřuje dynamickou architekturu o mobilní prvky, které se přesouvají.

Co obsahuje struktura každého informačního systému? Uveďte příklady.

- Komponenty - softwarový balíček nebo služba s rozhraním.
- Konektory - komunikační kanály propojující komponenty.
- Konfigurace - konkrétní způsob propojení komponent pomocí konektorů.

Popište, co se rozumí pojmem komponenta informačního systému. Uveďte příklady.

Komponenta je softwarový balík, modul nebo služba s rozhraním. Poskytuje funkčnost a zapouzdřuje data i chování. Např. modul pro věrnostní program.

Jaký je rozdíl mezi architekturou a návrhem informačního systému?

Architektura vychází především z nefunkčních požadavků. Popisuje, jaký má systém být.

Návrh vychází pouze z funkčních požadavků. Popisuje, co má systém poskytovat.

Jak se správně postupuje při stanovení architektury a návrhu informačního systému?

1. Identifikace požadavků.
2. Dekompozice systému do komponent.
3. Přidělení požadavků k jednotlivým komponentám.
4. Kontrola. Všechny požadavky musí být přiděleny.

Které kompetence obsahuje informační systém? Uveďte příklady těchto kompetencí.

- Komunikace s uživatelem. Např. vyplnění formuláře.
- Zpracování informací. Např. výpočet věku z vloženého data narození.
- Trvalé uchování dat. Např. uložení data narození v databázi.

Přednáška 3 - Vzory doménové logiky

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

Návrhový vzor je návod, který opakovaně vede k vytváření úspěšných řešení známého problému. Obsahuje:

- Výstižný název.
- Popis problému.
- Souvislosti. Např. s ostatními návrhovými vzory.
- Příklady použití vzoru. UML diagramy. Úryvky zdrojového kódu.

Např. Builder, Composite, Observer, Transaction Script, Table Data Gateway, Lazy Load.

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

Třívrstvá architektura rozděluje systém do tří vrstev.

- Prezentační vrstvu, která komunikuje s uživatelem.
- Doménovou vrstvu, která zpracovává informace a dočasně je uchovává ve formě dat.
- Vrstvu přístupu k datům, která trvale ukládá data.

Podstatou je zaměnitelnost implementací jednotlivých vrstev a minimalizace závislostí mezi vrstvami.

Logická architektura rozděluje software systému do vrstev. Stává se fyzickou, pokud software každé vrstvy běží na jiném zařízení.

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

Podstatou je oddělení Modelu a View tak, aby se zdrojový kód Modelu znovu využil při vytváření nového způsobu reprezentace.

MVC se liší tím, že neřeší přístup k databázi a není lineární, tzn. směr závislostí není stejný. Controller i View jsou závislé na Modelu.

Co řeší skupina návrhových vzorů pro doménovou logiku?

Řeší umístění stejných dat a algoritmů doménové logiky do tříd a služeb. Také řeší způsob zpracování požadavků z prezentační vrstvy.

Popište podstatu jednotlivých vzorů pro doménovou logiku (Transaction Script, Domain Model, Table Module, Service Layer).

- Transaction Script - jeden požadavek z prezentační vrstvy zpracovává jedna funkce (transakce), která validuje a zpracovává data a přistupuje k databázi.
- Domain Model - jeden požadavek může zpracovávat více objektů, které zapouzdřují data i chování.
- Table Module - jedna instance třídy dle tohoto vzoru řídí doménovou logiku pro všechny řádky jedné tabulky z databáze.
- Service Layer - zapouzdřuje doménovou logiku do samostatné vrstvy a poskytuje ji přes rozhraní více klientům - druhům prezentační vrstvy.

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

Liší se v umístění stejných dat a algoritmů do tříd a služeb.

Transaction Script

- Vhodný pro jednoduchou doménovou logiku, protože je nejjednodušší na implementaci.
- Nevhodný pro složitou logiku, protože dochází k duplikaci kódu mezi jednotlivými skripty.

Domain Model

- Vhodný pro složitou logiku, jelikož dokáže jednoduše implementovat polymorfismus.
- Nevhodný pro jednoduchou logiku kvůli vyšším nárokům na implementaci.

Table Module

- Vhodný pro logiku, která je úzce svázána s tabulkami z databáze.
- Nevhodný při využití polymorfismu, protože je nesnadné jej implementovat pomocí tohoto vzoru.

Service Layer

- Vhodná pro aplikace s více klienty z prezentační vrstvy.
- Nevhodná pro aplikace s jedním klientem kvůli zvýšeným nárokům na implementaci.

Přednáška 4 - Vzory mapování

Co společně řeší skupina návrhových vzorů pro práci s datovými zdroji?

Řeší zapouzdření logiky přístupu k datům a její oddělení od doménové logiky tak, aby na sobě byly nezávislé.

Popište podstatu jednotlivých vzorů pro práci s datovými zdroji (Table Data Gateway, Row Data Gateway, Active Record, Data Mapper).

- Table Data Gateway - jedna instance třídy dle tohoto vzoru obsluhuje přístup k datům všech řádků jedné tabulky z databáze.
- Row Data Gateway - každá instance třídy dle tohoto vzoru obsluhuje přístup k datům jednoho řádku tabulky.
- Active Record - podobný Row Data Gateway, přidává doménovou logiku.
- Data Mapper - vrstva mapperů, které převádějí data mezi relačním a objektovým schématem. Odstraňuje závislost mezi doménovou vrstvou a vrstvou přístupu k datům.

V čem se liší jednotlivé vzory pro práci s datovými zdroji? Kdy, kde a proč je použít / nepoužít?

Liší se v umístění logiky přístupu k datům do tříd, v tom zdali obsahují doménovou logiku a jaký typ závislostí je mezi nimi a okolními vrstvami.

Table Data Gateway

- Vhodná pro jednoduchou doménovou logiku pro nenáročnost implementace.
- Nevhodná při spojování tabulek.

Row Data Gateway

- Vhodná pro sdílení dat mezi transakčními skripty.
- Nevhodná při duplikaci kódu mezi transakčními skripty, pak je lepší využít Active Record a duplikovaný kód uložit tam.

Active Record

- Vhodný pro doménovou logiku s operacemi přímo mapovanými na tabulky.
- Nevhodný při využití polymorfismu pro nesnadnou implementaci.

Data Mapper

- Vhodný pro vyvíjení objektového a relačního schématu odděleně, protože odstraňuje závislosti mezi nimi.
- Nevhodný pro jednoduchou doménovou logiku, protože zvyšuje nároky na implementaci.

Se kterými vzory pro doménovou logiku by se mohly použít některé ze vzorů pro práci s datovými zdroji a proč? Uveďte příklady.

- Table Data Gateway - s Table Module, protože oba vzory přímo mapují jednu tabulku z databáze.
- Row Data Gateway - s Transaction Scriptem, protože umožňuje sdílet přístup k datům více skriptům.
- Active Record - s Transaction Scriptem, aby zabránil duplikaci kódu mezi více skripty.
- Data Mapper - s Domain Modelem, protože oba umožňují pracovat s polymorfismem.

Přednáška 5 - Objektově relační chování

Pro každý ze čtyř návrhových vzorů pro práci s datovými zdroji si promyslete a napište kousíček zdrojového kódu, ze kterého bude poznat, o který vzor jde.

Table Data Gateway

```
public class ProductTDG {
    public void Delete(int id) {...}
    public IDataReader Find(int id) {...}
    public void Insert(int id, string name) {...}
    public void Update(int id, string name) {...}
}
```

Row Data Gateway

```
public class ProductRDG {
    public int ID { get; set; }
    public string Name { get; set; }

    public void Delete() {...}
    public static ProductRDG Find(int id) {...}
    public void Insert() {...}
    public void Update() {...}
}
```

Active Record

```
public class ProductAR : ProductRDG {
    public decimal ComputeDiscount() {...}
    public bool IsAvailable() {...}
}
```

Data Mapper

```
public class ProductMapper {
    private Product product;

    public void Delete() {...}
    public ProductMapper(Product product) => this.product = product;
    public void Save() {...}
}
```

Co společně řeší návrhové vzory pro objektově relační chování?

Řeší snížení počtu přístupů k databázi, čímž se zvýší výkon aplikace a stane se více responzivní.

Kdy bychom měli zvážit použití vzoru Unit of Work a proč? Na příkladu vysvětlete, jak jej použít.

Unit of Work bychom měli použít tehdy, když se objekty v systému mění často a systém není kritický, díky čemuž si můžeme dovolit odložit ukládání změn do databáze.

Např. zákazník vytvoří objednávku, ta se uloží na server, nikoliv však na databázi. Při přidání položky se data objednávky změní, ale stále není uložena na databázi. Teprve když se zákazník odhlásí nebo potvrdí objednávku, potom je objednávka zapsána do databáze. Pokud objednávku zrušil předtím, smaže se pouze ze serveru, jelikož v databázi není.

Kdy bychom měli zvážit použití vzoru Identity Map a proč? Na příkladu vysvětlete, jak jej použít.

Identity Map využijeme tehdy, když záznamy v databázi mění svůj stav a existuje šance, že dvě části systému by chtěli pracovat se stejným záznamem.

Např. editor chce upravit článek v novinách. Systém načte záznam článku do objektu. O chvíli později chce druhý editor pracovat na stejném článku. Jelikož článek je již v paměti, nebude se jeho záznam opět načítat z databáze a systém bude pracovat se stejným objektem, se kterým pracuje první editor.

Kdy bychom měli zvážit použití vzoru Lazy Load a proč? Na příkladu vysvětlete, jak jej použít.

Lazy Load využijeme tehdy, když jsou objekty ve složitých kompozicích a tvoří kolekce, které jsou vlastněny jinými objekty.

Např. při načtení objednávky nemusíme načítat její položky. Položky načteme až při zobrazení detailu objednávky.

Popřemýšlejte, se kterými vzory pro práci s datovými zdroji byste mohli společně použít některé ze vzorů pro objektově relační chování a proč? Zkuste najít příklady.

Unit of Work

Lze použít s libovolným vzorem kromě Table Data Gateway, pokud zároveň nepoužíváme Domain Model, protože poté nemáme k dispozici objekt, který bychom mohli registrovat.

Identity Map

Vhodná pro Row Data Gateway a Active Record, kteří ji používají ve statické metodě při hledání záznamů v databázi. Klíčem mapy může být ID a hodnotou je Row Data Gateway nebo Active Record.

Pokud používáme abstraktní šablonový Mapper, ze kterého dědí ostatní Mappery, pak může být umístěna tam.

Lazy Load

Vhodná pro Row Data Gateway a Active Record. Ti můžou mít kolekci objektů uloženou jako NULL a teprve, když je poprvé potřeba, ji vyplní daty z databáze.

Přednáška 6 - Objektově relační struktury

V jakých situacích byste použili vzor Identity Field a proč? A kdy naopak ne? Napište fragment kódu, ze kterého bude patrné, proč jste tento vzor využili.

Identity Field je vhodné využít, když se stav záznamu ukládá v objektu a primární klíč záznamu je možné uložit ve stejném objektu. Důvodem je přímé mapování objektu na relační schéma. Nevhodné pro objekty B, které vlastní jiný objekt A pomocí kompozice. Pak postačí pro identifikaci B cizí klíč objektu A.

```
CREATE TABLE Products(
    id INT NOT NULL PRIMARY KEY,
    name VARCHAR(256) NOT NULL
);
```

```
public class ProductRDG {
    private int id; // Identity Field
    public string Name { get; set; }

    public void Delete() {...}
    public void Insert() {...}
    public void Update() {...}
}
```

Jaký je rozdíl mezi vzory Foreign Key Mapping a Associate Table Mapping? Napište dva fragmenty kódu, ze kterých bude patrné, že jste použili tyto vzory.

Rozdíl je ten, že Foreign Key Mapping nelze využít k reprezentaci vztahu M:N mezi dvěma tabulkami z databáze, zatímco Associate Table Mapping lze využít u všech typů vztahů. Další rozdíl je ten, že Associate Table Mapping dokáže uložit i atributy vztahu.

Foreign Key Mapping

```
public class Customer {...}

public class Order {
    private Customer customer;
}
```

Associate Table Mapping

```
public class Student {...}
public class Subject {...}

public class StudentSubject {
    private Student student;
    private Subject subject;
    private int grade;
}
```

V čem se liší vzor Dependent mapping od vzoru Data mapper? Kdy je vhodné jej použít?

Data Mapper ukládá logiku přístupu k datům jedné třídy do samostatné Mapper třídy. Dependent mapping spojuje logiky přístupu k datům dvou tříd do jedné a odstraňuje tak jednu Mapper třídu. Pokud využíváme Row Data Gateway, pak závislá třída nebude obsahovat žádnou logiku přístupu k datům a vše obsahuje třída vlastníka.

Vhodné při výskytu kompozice. Tzn. objekt B vlastní právě jedna instance objektu A a objekt B neexistuje samostatně.

Napište fragment kódu, ze kterého bude patrné, že jste použili vzor Dependent mapping.

```
public class Track {
    public string Name { get; }
    ...
}

public class Album {
    public int ID { get; }
    public List<Track> Tracks { get; }
    ...
}

public class AlbumMapper {
    private Album album;

    public AlbumMapper(Album album) => this.album = album;

    public void UpdateTracks() {
        var db = Database.GetInstance();
        db.ExecuteStatement($"DELETE FROM Tracks WHERE album_id = {album.ID}");

        foreach(var track in album.Tracks)
            db.ExecuteStatement($"INSERT INTO Tracks (album_id, name) VALUES
({album.ID}, {track.Name})");
    }
    ...
}
```

Vymyslete alespoň dva příklady vhodné pro použití vzoru Embedded value. Co je podstatou tohoto vzoru? Proč a kdy je vhodné jej použít?

Např. ukládání hodnoty peněz spolu s měnou, nebo ukládání časového období.

Podstatou je mapování objektu z paměti do více polí tabulky z databáze, která reprezentuje jiný objekt.

Vhodné pro hodnotové objekty, které nemají identitu a zároveň se chceme dotazovat nad daty těchto objektů. Pokud nepotřebujeme dotazování, pak je vhodnější použít Serialized LOB.

Vymyslete alespoň dva příklady vhodné pro použití vzoru Serialized LOB. Co je podstatou tohoto vzoru? Proč a kdy je vhodné jej použít?

Např. ukládání grafu zaměstnanců a oddělení, do kterých patří. Dalším příkladem je ukládání obrázků do databáze.

Podstatou je ukládání grafu objektů jako sekvence bytů (serializace) v poli tabulky z databáze.

Jelikož databáze většinou neumožňují vytvářet dotazy nad částmi LOB, pak je vhodné jej využít pro objekty, nad kterými se neprovádí dotazy a načítají se pokaždé společně jako jeden celek. Také vhodné pro statické číselníky, které se málokdy mění.

Přednáška 7 - Mapování dědičnosti a obecné vzory

Popište podstatu vzorů Single / Class / Concrete Table Inheritance a v jakých situacích je vhodné je použít.

Single Table Inheritance

Reprezentuje hierarchii tříd jako jedinou tabulku v databázi. Třídu, ke které patří záznam z tabulky lze rozlišit pomocí pole navíc, které pojmenujeme např. "type".

Vhodná, pokud se chceme vyhnout spojování tabulek nebo pokud očekáváme přesun dat v hierarchii tříd mezi potomky a předky. Takové změny totiž neovlivní schéma v databázi.

Class Table Inheritance

Každá třída v hierarchii má tabulku v databázi obsahující data, která tato třída přidává. Data, která dědí jsou v tabulkách předků. Sloupec pro primární klíč je duplikován.

Vhodná, pokud nechceme plýtvat místem. V tabulkách se nenachází sloupce, které by nebylo možné využít, protože patří jiné třídě.

Concrete Table Inheritance

Každá konkrétní třída v hierarchii má svou vlastní tabulku v databázi. Data, která dědí jsou v této tabulce duplikována jako nové sloupce.

Vhodná, pokud se chceme vyhnout spojování tabulek a uzamykání tabulek předků.

Napište fragment kódu, ze kterého bude patrné, že jste použili vzor Single Table Inheritance.

```
public abstract class BankAccount {
    private int ID;
    private decimal balance;
    ...
}

public class CheckingAccount : BankAccount {
    private int checksUsed;
    ...
}

public class SavingsAccount : BankAccount {
    private decimal interest;
    ...
}
```

```
CREATE TABLE BankAccounts (
    id INT NOT NULL PRIMARY KEY,
    type VARCHAR(32) NOT NULL,
    balance DECIMAL(13, 2) NOT NULL,
    checks_used INT NULL,
    interest DECIMAL(13, 2) NULL
);
```

Jaký je rozdíl mezi vzory Class a Concrete Table Inheritance? Napište dva fragmenty kódu, ze kterých bude patrné, že jste použili tyto vzory.

Class Table Inheritance

- Musíme vytvořit tabulku i pro abstraktní třídy.
- Nevytváříme nové sloupce pro zděděná data kromě primárního klíče.

Concrete Table Inheritance

- Tabulky reprezentují pouze konkrétní třídy.
- Zděděná data jsou duplikovány jako stejné sloupce mezi všemi tabulkami.

Společný kód pro doménovou vrstvu

```
public abstract class BankAccount {
    private int ID;
    private decimal balance;
    ...
}

public class CheckingAccount : BankAccount {
    private int checksUsed;
    ...
```

```
}

public class SavingsAccount : BankAccount {
    private decimal interest;
    ...
}
```

Databáze s Class Table Inheritance

```
CREATE TABLE BankAccounts (
    id INT NOT NULL PRIMARY KEY,
    balance DECIMAL(13, 2) NOT NULL
);

CREATE TABLE CheckingAccounts (
    id INT NOT NULL PRIMARY KEY,
    checks_used INT NOT NULL
);

CREATE TABLE SavingsAccounts (
    id INT NOT NULL PRIMARY KEY,
    interest DECIMAL(13, 2) NOT NULL
);
```

Databáze s Concrete Table Inheritance

```
CREATE TABLE CheckingAccounts (
    id INT NOT NULL PRIMARY KEY,
    balance DECIMAL(13, 2) NOT NULL,
    checks_used INT NOT NULL
);

CREATE TABLE SavingsAccounts (
    id INT NOT NULL PRIMARY KEY,
    balance DECIMAL(13, 2) NOT NULL,
    interest DECIMAL(13, 2) NOT NULL
);
```

Vymyslete a popište příklad na využití vzoru Gateway.

Služba pro rozesílání e-mailů může mít komplikované rozhraní, které je pro systém nepotřebné v celém svém rozsahu. Např. posílání příloh, když systém potřebuje posílat pouze text.

Přístup k této službě lze zapouzdřit v nové třídě EmailGateway. Pokud rozhraní této nové třídy uložíme samostatně jako IEmailGateway, pak lze nahradit konkrétní e-mailovou službu bez toho, aby se musel změnit kód, který ji používá. Důvodem této změny může být např. nižší cena.

Vymyslete a popište příklad na využití vzoru Mapper.

Konkretizací vzoru Mapper je Data Mapper, který propojuje doménovou vrstvu a vrstvu přístupu k datům. Díky tomu se mohou objektová a relační schémata vyvíjet nezávisle.

Vymyslete a popište příklad na využití vzoru Layer Supertype. Napište fragment kódu využívající dědičnost, ze kterého bude patrné, že jste použili tento vzor.

Pokud všechny záznamy z tabulky používají pro identifikaci umělý klíč ID, pak je možné data i logiku pro identifikaci vyčlenit do abstraktního předka pro všechny objekty doménové vrstvy.

```
public abstract class DomainObject {
    public int ID { get; set; }
}

public class Order : DomainObject {...}
public class Product : DomainObject {...}
```

Vymyslete a popište příklad na využití vzoru Service Stub (Mock Object). Napište fragment kódu, ze kterého bude patrné, že jste použili tento vzor.

Platby v systému řeší webová služba. Přístup k této službě zapouzdříme pomocí rozhraní IPaymentGateway. Poté vytvoříme dvě implementace tohoto rozhraní. Jednu z nich pro testování dle vzoru Service Stub, která se nebude připojovat na web a vrací předem známý výsledek.

```
public interface IPaymentGateway {
    public PaymentResult Pay(int transactionID);
}

public class PaymentServiceStub : IPaymentGateway {
    public PaymentResult Pay(int transactionID) {
        return PaymentResult.SUCCESS;
    }
}

public class WebPaymentGateway : IPaymentGateway {
    public PaymentResult Pay(int transactionID) {
        return
App.GetInstance().Redirect($"https://paymentservice.com/{transactionID}").GetPay
mentResult();
    }
}

// Použití v klientském kódu.
var res = Environment.GetPaymentGateway().Pay(transactionID);
```

Přednáška 8 - Jak vyvíjet v týmu

Popište, co se rozumí vodopádovým modelem a jaké jsou jeho výhody a nevýhody.

Vodopádový model je vývojová metodika, která rozděluje vývojový proces do fází, které jdou za sebou a přistoupit k následující fázi lze pouze při dokončení fáze předchozí. Fázemi jsou specifikace požadavků, návrh, implementace, ověření a údržba.

Výhody jsou bezproblémová záměna lidí díky důkladné dokumentaci, včasné odhalení chyb a stabilita projektu díky jednoduchosti jeho řízení.

Nevýhody jsou neschopnost reagovat na změny požadavků, neschopnost změnit technologie, které se v průběhu vývoje ukázaly jako nesprávné volby. Dalším problémem je, že se nemůžeme vrátit k předchozí fázi a každou fázi nemůžeme dovést do dokonalosti.

Popište, co se rozumí iterativním a inkrementálním vývojem. Uveďte příklad.

Iterativní vývoj je opakovaný proces vodopádového modelu, tedy fází specifikace požadavků, návrh, implementace a testování. Každá iterace přidává funkčnost a na jejím konci je spustitelný výstup.

Např. zákazník původně požaduje pro svůj e-shop funkci okamžité platby po potvrzení objednávky. Později se však rozhodne, že platbu lze odložit. V následující iteraci se tento požadavek specifikuje a implementuje.

Inkrementální vývoj umožňuje vyvíjet různé části systému různě rychle, a to dokonce paralelně. Vývojáři se mohou vracet k již hotovým částem systému a zlepšovat je.

Např. doménová vrstva a vrstva k přístupu k datům se mohou vyvíjet odděleně a spojí se až v okamžiku jejich dokončení.

Co se rozumí UP (unified process)? Na jakých principech, charakteristikách a fázích je postaven? Uveďte příklady.

UP je komerční robustní metodika vývoje.

Principy

- Založena na procesech a rozsáhlé dokumentaci. To umožňuje bezproblémovou výměnu vývojářů.
- Iterativní a inkrementální vývoj. Umožňují reagovat na nové požadavky a vracet se k již hotovým částem systému a zlepšovat je.
- Use-case driven. V každé části vývoje jsou využívány případy užití.
- Architecture-centric. Volba technologie je důležitá, protože se na ni systém stává závislým. Udržuje architekturu stabilní.
- Risk-focused. Rizika a nejistoty se řeší v rané fázi projektu.

Fáze

- Zahájení - cílem je prokázat realizovatelnost projektu.
- Rozpracování - cílem je prokázat, že vývojový tým má schopnost systém vybudovat.
- Konstrukce - cílem je vytvoření beta verze.

- Předání - cílem je oprava chyb nalezených zákazníkem a konečné nasazení systému do provozu.

Jaké jsou čtyři základní charakteristiky agilního softwarového vývoje, které ho odlišují od vodopádového a dalších robustních přístupů?

- Lidi a jejich vztahy nad procesy a nástroji.
- Funkční software nad rozsáhlou dokumentací.
- Spolupráce se zákazníkem nad předem sjednanou smlouvou.
- Reakce na změnu požadavků nad plněním plánu.

Na jakých principech a praktikách je založeno tzv. extrémní programování?

Principy

- Jednoduchost řešení problému. Nejdříve vytvoříme nejjednodušší řešení a postupně jej měníme v komplexní.
- Častá zpětná vazba od testů i zákazníků.
- Uvítání změn požadavků.
- Odvaha. Např. vyřadit kód, který se vyvíjel dlouho, ale nyní se ukázal jako nevhodný.

Praktiky

- Využívání všech osvědčených metod.
- Párové programování.
- Posouzení kódu (Code Review) programátorem, který jej nepsal.

Které nejdůležitější charakteristiky má SCRUM?

- Agilní metodika vývoje.
- Sprint - iterace s délkou 1 měsíce, na jejímž výstupu je funkční přírůstek systému.
- Rozděluje zaměstnance do rolí.
    - Product owner - zajišťuje komunikaci mezi zákazníkem a vývojovým týmem.
    - Vývojový tým.
    - SCRUM master - vede a motivuje vývojový tým. Je odpovědný za odstranění rizik a nejistot.
- Používá User Story namísto Use Case. User Story není zaměřena na detaily a je myšlena k vyvolání konverzace.
    - User Story jsou uloženy v backlogu. Každý sprint řeší část backlogu.
- Pravidelné schůzky vývojového týmu (každodenní nebo 2-3x do týdne).
- Zaměřený na rizika.

Popište proces typický pro tzv. testy řízený softwarový vývoj. Uveďte příklad.

1. Napsat testy.
2. Ujistit se, že testy neprojdou.
3. Napsat zdrojový kód systému.
4. Spustit testy. Pokud alespoň jeden neprojde pak zpět ke kroku 3.
5. Zlepšení kódu (refaktoring).
6. Opakování pro další část systému.

Např. při přidání funkčnosti odložení platby objednávky v systému e-shopu se musíme ujistit pomocí testů, že okamžitá platba stále funguje tak, jak byla dříve naprogramovaná.

Přednáška 9 - Doménově specifické jazyky

Co je doménově specifický jazyk? Proč a kdy je dobré ho využít? Uveďte příklady.

Jedná se o programovací jazyk zaměřený na konkrétní doménu s omezenou výrazovostí.

Je dobré jej využít pro zrychlení vývoje systému v dané doméně a pro zefektivnění komunikace mezi programátory a doménovými experty.

Např. HTML pro vytváření vzhledu webových stránek, SQL pro práci s relační databází a XPath pro práci s XML.

Co je cílem využití doménově specifického jazyka a jaké vlastnosti by měl mít?

Cíle

- Zrychlení vývoje.
- Zmenšení šance výskytu chyb.
- Zefektivnění komunikace mezi programátory a doménovými experty.

Vlastnosti

- Jednoduchý na porozumění, díky čemuž je po krátkém učení čitelný i pro ty, kteří nejsou programátory.
- Zmenšuje množství kódu oproti obecným jazykům, díky čemuž zrychluje vývoj a zmenšuje šanci výskytu chyb.

Jaký je rozdíl mezi externím a interním doménově specifickým jazykem? Uveďte příklady.

Externí DSL je oddělen od hlavního jazyka aplikace, ve které se používá. Např. Regex, SQL, XPath.

Interní DSL je součástí knihovny hlavního jazyka aplikace. Např. jazyk pro formátování času a data, JMock.

S jakými problémy se můžeme setkat při návrhu doménově specifického jazyka? Uveďte příklady.

- Při využívání více jazyků může být těžké najít nové vývojáře, kteří rozumí všem.
- Vytváření vlastního DSL nese sebou i náklady na jeho údržbu.
- Pokud se nový jazyk neujme, nebo pokud je výhradně používán ve firmě, která ho vyvinula, pak je velmi těžké najít nové vývojáře, kteří budou ochotni se jej naučit.
- DSL je abstrakce a schovává nízkoúrovňové části systému. Opomenutí těchto částí může vést ke snížení výkonu systému při nesprávném použití DSL.