KATEDRA INFORMATIKY
PŘÍRODOVĚDECKÁ FAKULTA
UNIVERZITA PALACKÉHO

SOFTWAROVÉ INŽENÝRSTVÍ

VLADIMÍR SKLENÁŘ

VÝVOJ TOHOTO UČEBNÍHO TEXTU JE SPOLUFINANCOVÁN
EVROPSKÝM SOCIÁLNÍM FONDEM A STÁTNÍM ROZPOČTEM ČESKÉ REPUBLIKY

Olomouc 2007

Abstrakt

Text poskytuje úvod do softwarového inženýrství pro studenty kombinované formy studia. Je orientován především na objektově orientovaný přístup k tvorbě software. Výklad je založen na vysvětlení principů iterativního a inkrementálního softwarového procesu označovaného zkratkou RUP (Rational Unified Process).

Cílová skupina

Kombinovaná forma studia, ale lze využít i pro prezenční formu

Obsah

1 Co je softwarové inženýrství

Studijní cíle: Cílem této kapitoly je porozumět tomu, čím se softwarové inženýrství zabývá.

Klíčová slova: Softwarové inženýrství, metoda, profese.

Potřebný čas: 2 hodiny

1.1 Definice a předmět zájmu

Softwarové inženýrství se zabývá všemi aspekty tvorby a údržby software

Softwarové inženýrství je inženýrská disciplína zaměřená na všechny aspekty tvorby software od počátečních stádií specifikování systému až po udržování systému během jeho využívání. V této definici jsou podstatné dvě fráze:

- *Inženýrská disciplína.* Inženýrství je spojováno se systematickým a disciplinovaným aplikováním vědeckých znalostí při návrhu a tvorbě cenově efektivních řešení praktických problémů.
- *Všechny aspekty tvorby software*. Softwarové inženýrství se nezabývá pouze činnostmi spojenými se samotným vývojem software. Mezi předměty jeho zájmu patří mimo jiné také řízení softwarového projektu, organizace řešitelského týmu a komunikace mezi jeho členy a některé ekonomické otázky, například stanovení ceny software.

Cílem softwarových inženýrů je pokud možno co nejefektivnějším způsobem dosáhnout vytvoření kvalitního softwarového produktu. Při své práci postupují systematickým a organizovaným způsobem, který je dán zvolenou metodou a používají přitom vhodné nástroje, které jejich činnosti podporují nebo automatizují (vývojová prostředí, CASE, nástroje pro testování, ...). Pod pojmem **metoda** nebo **technika** přitom chápeme formální proceduru, která vede k vytvoření požadovaného výsledku. Jsou s ní spojeny především:

- *Terminologie.* Metoda zavádí a definuje význam pojmů, které se při jejím aplikování používají (např. třída, objekt, asociace, ...). Je to nutné pro to, aby si zúčastnění řešitelé vzájemně rozuměli.
- *Notace.* Způsob, jakým jsou zaznamenávány informace a výsledky, které při aplikování dané metody získáme nebo vytvoříme. V softwarovém inženýrství mají často tvar diagramů zachycujících různé modely řešeného systému.
- *Proces.*Popis činností, které se provádějí, jejich návaznost a doporučení pro jejich provádění.

V tomto textu se budeme zabývat především objektově orientovaným postupem při tvorbě software využívajícím jazyk UML (Unified Modeling Language).

Průvodce studiem

Pojem softwarové inženýrství byl poprvé použit v roce 1968 na konferenci zabývající se tzv. softwarovou krizí. V té době se tvorba velkých softwarových projektů vyznačovala celou řadou nedostatků a zaostávala za rychlým rozvojem hardwarových technologií. Neúměrně velké procento započatých projektů skončilo neúspěchem.Projekty byly dokončovány se zpožděním a jejich plánovaný rozpočet býval značně překročen. Od té

doby došlo ke značnému pokroku, ale zdaleka nelze říci, že již bylo dosaženo ideálního stavu

1.2 Vztah k informatice

Informatika se zabývá teorií, softwarové inženýrství jejím aplikováním

Softwarové inženýrství je zaměřeno na praktické problémy a při jejich řešení aplikuje výsledky jiných věd. Mezi nimi zaujímá významné postavení informatika (computer science). Ta se mimo jiné zabývá teoretickým aspekty spojenými s využíváním počítačů a softwarových systémů. Zatímco například programovací jazyky jsou pro informatiku předmětem zájmu, pro softwarové inženýrství je to nástroj, který využívá při navrhování a implementování řešení. Celá řada poznatků z informatiky je pro softwarové inženýry podstatná. Bez jejich zvládnutí a schopnosti je prakticky použít by nemohli úspěšně vykonávat svou profesi.

1.3 Softwarové inženýrství jako profese

Tak jako jiné profese i softwarové inženýrství si postupně vytvoří pravidla spojená s jejím vykonáváním. Jedná se především o stanovení znalostí a dovedností , které by si měl softwarový inženýr osvojit, a stanovení zásad chování, které by měl při vykonávání profese dodržovat.

Softwarový inženýr by měl při své práci dodržovat etická pravidla

Softwaroví inženýři musí akceptovat, že na jejich prací jsou kladeny i jiné nároky než jen „pouhé“ aplikování technických znalostí. Chtějí-li být respektováni jako profesionálové musí dodržovat etická a morální pravidla spojená s profesní zodpovědností. Neměli by používat svých dovedností a schopností k nečestným účelům nebo způsobem, který poškodí pověst profese. Mezi oblasti, kde akceptovatelné chování není vždy stanovené zákonem patří například

- *Zachování důvěrnosti***.** Je nutné respektovat důvěrnost vzhledem k zaměstnavateli i klientům bez ohledu na to, zda o tom byla sepsána formální smlouva
- *Respektování kvalifikace.* Není akceptovatelné mylné informování o stupni kvalifikace a vědomé přijmutí práce, která dosažené kvalifikaci neodpovídá.
- *Dodržení autorských práv*. Je nutné respektovat zákony o intelektuálním vlastnictví . Je třeba zajistit, že intelektuální vlastnictví zaměstnavatelů a klientů je chráněno.
- *Nezneužívání počítačů*. Není akceptovatelné používání technických dovedností ke zneužití počítačů jiných osob. Zneužití může mít celou řadu podob, od relativně nezávadného (hraní her na počítači zaměstnavatele) až po velmi nebezpečné (šíření virů).

Při stanovení etických standardů a jejich prosazování hrají důležitou roli profesní společnosti a organizace. Například uznávané organizace ACM (Association for Computing Machinery) a IEEE (Institute of Electrical and Electronic Engineers) vydaly společný dokument „Software Engineering Code of Ethics and Profesinal Practices“. Podle něj by softwaroví inženýři měli dodržovat následujících osm principů:

- Pracovat v souladu s veřejným zájmem
- Sledovat nejlepší zájmy klientů a zaměstnavatelů
- Vytvářet produkty dosahující nejvyššího možného profesionálního standardu
- Dodržovat nezávislost a integritu při profesionálním posuzování
- Manažéři a vedoucí projektů by měli prosazovat etický přístup k řízení vývoje a udržování software.

- Posilovat reputaci profese v souladu s veřejným zájmem
- Udržovat korektní vztahy s kolegy
- Dbát na celoživotní vzdělávání a dodržování etického přístupu

Očekávané znalosti a dovednosti softwarových inženýrů

Postupně také dochází k sjednocení názorů na to, jaké znalosti a dovednosti by softwaroví inženýři měli mít. V dokumentu Software Engineering Body of Knowledge (SWEBOK) je popsán rozsah znalostí a dovedností, které se očekávají u softwarového inženýra po 4 letech praxe. Náplň univerzitních kurzů softwarového inženýrství a znalosti očekávané u absolventa vysoké školy stanovuje dokument SEEK (Software Engineering Education Knowledge), který opět společně vytvořily ACM a IEEE.

Shrnutí

Softwarové inženýrství se zabývá všemi aspekty tvorby software. Využívá při tom poznatky jiných disciplín, především informatiky a matematiky.

Pojmy k zapamatování

- Softwarové inženýrství,
- Metoda.
- Etický kodex

Kontrolní otázky

Čím se zabývá softwarové inženýrství?
Je vytváření a šíření počítačových virů v souladu s etickým kodexem?

2 Softwarový proces

Studijní cíle: Pochopit pojem softwarový proces a seznámit s druhy softwarových procesů, které jsou dnes v praxi používány.

Klíčová slova: Softwarový proces, životní cyklus, vodopád, iterace, verze, agilní metody.

Potřebný čas: 5 hodin

2.1 Definice

Softwarový proces je logicky související množina aktivit vedoucí k vytvoření nebo úpravě softwarového produktu. Tyto aktivity zahrnují především specifikování požadavků, návrh a implementování produktu, jeho testování, nasazení a další evoluci.

Softwarový proces nám poskytuje návod pro efektivní vývoj a udržování software. Při jistém stupni zjednodušení si jej můžeme představit jako mechanismus, jehož úkolem je zajistit, aby na základě vznesených požadavků vzniknul softwarový produkt, který je splňuje.

Softwarový proces stanovuje kdo, co, kdy a jak má udělat

Disciplinované používání stanoveného softwarového procesu snižuje rizika nečekaných problémů a zvyšuje předvídatelnost celého postupu prací. Jednotliví vývojáři ví, jak mají postupovat a mohou se spolehnout na to, že ostatní postupují stejně. Softwarové procesy mohou být upravovány na základě dobrých nebo špatných předchozích zkušeností. Umožňují tak zaznamenat úspěšné postupy a předávat je ostatním.

Z jiného úhlu pohledu můžeme říci, že softwarový proces stanovuje kdo, co, kdy a jak má udělat. S touto jednoduchou, ale výstižnou definicí jsou spojeny následující pojmy:

- **Role** definují funkce, ve kterých vystupují lidé zapojení do softwarového procesu (analytik, programátor, tester, projektový manažér apod.). S každou rolí jsou spojeny artefakty, za které zodpovídá a aktivity, které má vykonávat. Jedna osoba může vystupovat ve více rolích a naopak jednu roli může zastávat více osob.
- **Artefakt** je označení pro entity, které jsou v rámci softwarového procesu vytvářeny, modifikovány nebo využívány. Může to být například textový dokument, zdrojový kód, komponenta apod. Mezi nejdůležitější artefakty patří modely.
- **Aktivita** je činnost prováděná pracovníky v jednotlivých rolích při realizování softwarového procesu. Jejich výsledkem jsou nově vytvořené nebo modifikované artefakty. S aktivitami bývají spojeny doporučení, jak při jejich provádění postupovat.
- **Fáze softwarového procesu** je časový interval, kterému odpovídá stanovený stupeň rozpracování (dokončení) softwarového díla
- **Milník** stanovuje, které artefakty mají být v daném čase dokončeny. Slouží jako kontrolní bod umožňující rozhodnout, zda je možné přejít do další fáze procesu.

V rámci softwarového procesu tedy lidí vystupující v některé z rolí provádějí aktivity vedoucí k vytvoření artefaktů, za které daná role zodpovídá. Přitom mohou využívat jako vstupy artefakty, které již byly vytvořeny v logicky předcházejících aktivitách.

2.2 Druhy softwarových procesů

Softwarových procesů může být celá řada.. V podstatě se dá říci, že každá softwarová firma má svůj vlastní konkrétní softwarový proces, který odpovídá jejím specifickým podmínkám. Vliv

na stanovení podrobností softwarového procesu má například počet zaměstnanců a jejich zkušenosti, druh a rozsah řešených projektů, dostupnost nástrojů pro podporu vývoje atd.. Mezi používanými softwarovými procesy můžeme ale rozpoznat několik modelů softwarového procesu. Model softwarového procesu je jeho abstraktní reprezentací. Obsahuje jeho základní charakteristiky a principiální popis používaných postupů. Většina v současné době používaných softwarových postupů jsou založeny na následujících dvou modelech. Pro správné pochopení základního rozdílu mezi nimi je nutné si uvědomit, co znamená pojem fáze softwarového procesu. Myslí se tím časový interval, kterému odpovídá stanovený stupeň rozpracování (dokončení) softwarového díla.

- **Vodopádový přístup** je historicky starší, vznikal v sedmdesátých letech minulého století. Chápe jednotlivé vykonávané aktivity jako fáze procesu. Můžeme tedy konstatovat, že projekt je v daném okamžiku ve fázi specifikování požadavků, analýzy, návrhu, implementace, testování nebo předávání, Přechod do další fáze je možný teprve v okamžiku, kdy jsou činnosti spojené s aktuální fází zcela dokončeny. Tedy například k implementaci se přistupuje až poté, co jsou kompletně specifikovány všechny požadavky na vytvářené softwarové dílo, je ukončena analýza problému a kompletně navrženo softwarové řešení. Tento na první pohled přímočarý a snadno pochopitelný postup se ale v praxi příliš neosvědčil. Hlavním důvodem je fakt, že v současné době je nereálné očekávat, že příslušnou činnost, například specifikování požadavků, provedeme najednou úplně a bezchybně. Je nutné si uvědomit, že život přináší neustále změny, které je nutné do řešeného systému zapracovávat. Většina vývojářů má zkušenost s tím, že budoucí uživatel si na některé z požadovaných funkcí vzpomene až v okamžiku, kdy s vytvářeným softwarovým dílem začne pracovat. Jejich zapracování může vést k návratu o několik fází zpět a přepracování řady dříve vyhotovených artefaktů. Obecně přitom platí, že odstranění problému je tím dražší, čím později je odhalen.

Vodopád spojuje s jednotlivými fázemi některou z vykonávaných aktivit

- **Iterativní přístup** je založen na opakovaném vykonání jednotlivých aktivit v takzvaných iteracích. Pro každou iteraci je vybrána malá skupina funkcí, které jsou během ní kompletně začleněny do vznikajícího softwarového díla. Provedou se tedy všechny potřebné aktivity od konkretizování požadavků na vybrané funkce až po jejich testování. Během každé iterace tedy vznikne nová verze software, která je rozšířena o novou funkcionalitu. Někdy to bývá charakterizováno tak, že vývoj probíhá jako posloupnost minivodopádů. V každé iteraci dochází k rozšíření a zpřesnění informací o řešeném problému. Při tomto způsobu práce je reakce na změnu daleko pružnější a k odhalení případných chyb a problémů dojde dříve než u vodopádového přístupu. Taktéž budoucí uživatelé mají daleko větší možnost ovlivňovat výslednou podobu produktu, protože si vyvíjený systém mohou vyzkoušet podstatně dříve. Z předchozího textu je patrné, že při iterativním přístupu nemůžeme spojovat pojmy fáze a aktivita. Nemůžeme tedy tvrdit, že se produkt se nachází např. ve fázi analýzy. Podrobněji se rozdělením iterativního softwarového procesu budeme zabývat v kapitole RUP.

Iterativní proces probíhá v iteracích, během nichž proběhnou všechny aktivity a vznikne nová verze produktu

2.3 Agilní techniky

Většina softwarových procesů používaných v sedmdesátých až devadesátých letech minulého století předpokládala striktní dodržování naplánovaného postupu a povinné vytváření celé řady dokumentů. To bylo opodstatněné v případě velkých projektů, na kterých pracovaly desítky vývojářů, popřípadě u projektů, které vyžadovaly vysoký stupeň spolehlivosti (např. řídící systémy moderních letadel). V případě menších projektů ale docházelo k tomu, že režie spojená s dodržováním stanoveného postupu a vytvářením všech požadovaných artefaktů mohla být příliš velká. Často spotřebovala více času než samotné programování a testování. Nespokojenost

s tímto stavem vedla několik zkušených softwarových vývojářů na konci devadesátých let k navržení nového přístupu k tvorbě software, který bývá označován termínem agilní metody. Jejich cílem je umožnit vývojářům soustředit se především na vytváření samotného software, a ne na jeho navrhování a dokumentování. Agilní metody jsou založeny na iterativním přístupu a předpokládají, že požadavky na vyvíjený systém se mohou během vývoje měnit. Snaží se o co nejrychlejší dodání fungujícího (i když ne zcela hotového) software zákazníkům. Ti pak po seznámení s ním předkládají nove, popřípadě změněné požadavky na systém, které se zohlední při dalších iteracích.

Základní myšlenky, ze kterých agilní metody vycházejí byly v roce 2001 formulovány v manifestu agilního vývoje software (agile manifesto). Mezi klíčové principy patří:

- Lidé a jejich interakce jsou důležitější než procesy a nástroje
- Fungující software je důležitější než podrobná dokumentace
- Spolupráce se zákazníkem je důležitější než uzavřené smlouvy
- Reagování na změny je důležitější než dodržování plánu.

Agilní metody vyžadují aktivní účast zákazníků během všech vykonávaných činností. Postup prací není podrobně plánován dopředu a je aktualizován na konci každé iterace. Vytvářejí se pouze ty artefakty, které jsou pro daný projekt užitečné. Jsou vhodné spíše pro menší projekty na nichž pracuje 2-5 vývojářů. Na kvalitu vývojářů a jejich spolupráci kladou vysoké nároky.

V současné době existuje několik agilních metod. Mezi nejznámější patří extrémní programování a Scrum.

Shrnutí

Softwarový proces je logicky související množina aktivit, která vede k vtvoření nebo úpravě softwarového produktu. Mezi nejznámější modely softwarového procesy patří procesy označované termínem vodopád a iterativní softwarové procesy.

Pojmy k zapamatování

- Softwarový proces
- Role, aktivita, artefakt.
- Fáze, milník
- Vodopád
- Iterace

Kontrolní otázky

1. *Jaký je rozdíl mezi vodopádem a iterativním softwarovým procesem?*
2. *Vysvětlete pojem iterace.*

3 Jazyk UML

Studijní cíle: V této kapitole se seznámíme s jazykem UML, historií jeho vzniku a jeho významem v rámci softwarového procesu. Dále se seznámíme s nejčastěji používanými diagramy.

Klíčová slova: Modelování, diagram, případ užití, třída, asociace, generalizace.

Potřebný čas: 6 hodin

Vývoj současných softwarových systému je natolik složitý proces, že lidská mysl není schopna uchopit najednou všechny jeho podrobnosti. Proto je nutné softwarové systémy modelovat. Modely jsou zjednodušením nebo abstrakcí reality, kterou reprezentují. Pro úplné zachycení všech podrobností o problematice, pro kterou chceme vytvořit softwarový produkt, je potřeba vytvořit více modelů. Každý z nich zaznamenává pouze ty prvky řešeného systému, které jsou podstatné pro stanovený úhel pohledu, například pohled na statickou strukturu problémové domény nebo pohled na dynamické chování některého objektu apod.. Neexistují přesná kritéria, která by nám umožnila jednoznačně určit, který model je lepší nebo vhodnější než druhý. Podstatné je to, jak přispívají k pochopení řešeného systému a podporují výměnu informací mezi jednotlivými řešiteli.

Pro využívání modelování v praxi softwarových firem je nutná existence standardu, který zajistí, že všichni řešitelé budou chápat prvky obsažené v modelech stejným způsobem a budou používat stejnou terminologii a notaci. V současné době jako takový standard slouží jazyk označovaný zkratkou UML (Unified Modeling Language, unifikovaný modelovací jazyk).

3.1 UML a jeho historie

UML je univerzální jazyk pro vizuální modelování systémů. Je spojován především s modelováním objektově orientovaných softwarových systémů, může být ale použit i v jiných oblastech.

Jednotlivé modely jsou graficky prezentovány pomocí diagramů. UML nabízí 13 druhů diagramů. Je tak umožněno zaznamenání různých pohledů na řešený systém. Při vytváření diagramů se předpokládá využití nástrojů typu CASE (computer-aided software engineering). V současné době jich existuje celá řada.

Jazyk definuje pouze sémantiku pojmů používaných při modelování (třída, asociace, stav, apod.) a grafickou syntaxi pomocí níž je v jednotlivých diagramech zaznamenáváme. UML jako takové nestanovuje softwarový proces, který má být použit. Často je však spojován s procesem označovaným zkratkou UP (Unified Process, unifikovaný proces), popřípadě jeho komerční verzí RUP (Rational Unified Process) dodávanou firmou IBM Rational.

Jazyk UML začal vznikat v polovině devadesátých let. V té době existovalo několik různých metodik a s nimi spojených notací pro podporu objektově orientovaného modelování. Přestože v principu používaly stejný přístup, odlišovaly se od sebe používanými pojmy a grafickými symboly pomocí nichž byly v diagramech zaznamenávány. To vedlo k tomu, že spolupráce týmů používajících různé metodiky byla velmi obtížná. Autoři nejpoužívanějších metod (Booch, Rumbaugh, Jacobson) se proto spojili ve firmě Rational Corporation a začali pracovat na tvorbě UML. V roce 1997 sdruženi OMG (Object Management Group) přijalo verzi 1.1 jako standard. V poměrně krátké době byl jazyk UML vývojářskou veřejností přijat.

UML je neustále rozvíjen a zpřesňován. V současné době je aktuální verze 2.1. Ve verzi 2 bylo zavedeno mnoho nových prvků, které rozšířily vizuální vyjadřovací možnosti jazyka. Současně

byla zpřesněna a vylepšena sémantika jazyka s cílem dosáhnout jednoznačnosti při interpretování jednotlivých modelů. To bylo nutné především proto, aby mohly vznikat nástroje, které umožní automatickou transformaci vytvořených modelů až do finálního kódu. Mohl by pak být realizován vývoj software tak, jak jej definuje architektura MDA (Model Driven Architecture). Její podstatou je představa, že při vývoji softwarového systému je rozhodující vytvoření jeho modelu. Automatické nástroje pak zajistí jeho transformaci do spustitelného kódu.

3.2 Diagramy UML

Diagramy nám umožňují graficky reprezentovat naše porozumění řešení. Diagramy nejsou modely, většinou graficky prezentují pouze některé prvky celého modelu. Poskytují tak částečný pohled na model. Výběr prvků z modelu, které do konkrétního diagramu umístíme, je dán tím, jakou část řešení problému chceme pomocí diagramu prezentovat. Jeden prvek modelu (např. třída) může být umístěn ve více diagramech. UML poskytuje 13 typů diagramů. Ne všechny se používají stejně často. Některé z nich, například diagram tříd, se používají prakticky v každém projektu. Jiné pouze ve speciálních případech. V následujícím textu se stručně seznámíme s těmi, které patří k těm častěji používaným.

3.2.1 Diagram případů užití

UML podporuje softwarové procesy, které jsou řízené uživatelskými požadavky na systém. Pro jejich zaznamenání poskytuje následující pojmy.

- **Aktér** specifikuje roli, ve které vystupuje konkrétní subjekt (osoba nebo jiný systém), který s modelovaným systémem vstupuje do interakce. Je to prvek, který vyvolává některé funkcionality modelovaného systému a řídí jejich průběh, není ale jeho součástí.
- **Případ užití** (use case) definuje jeden možný způsob využití systému některým z aktérů. Popisuje posloupnost interakcí mezi aktérem a systémem vedoucí k dosažení cíle, který uživatel vyvoláním případu užití sledoval.

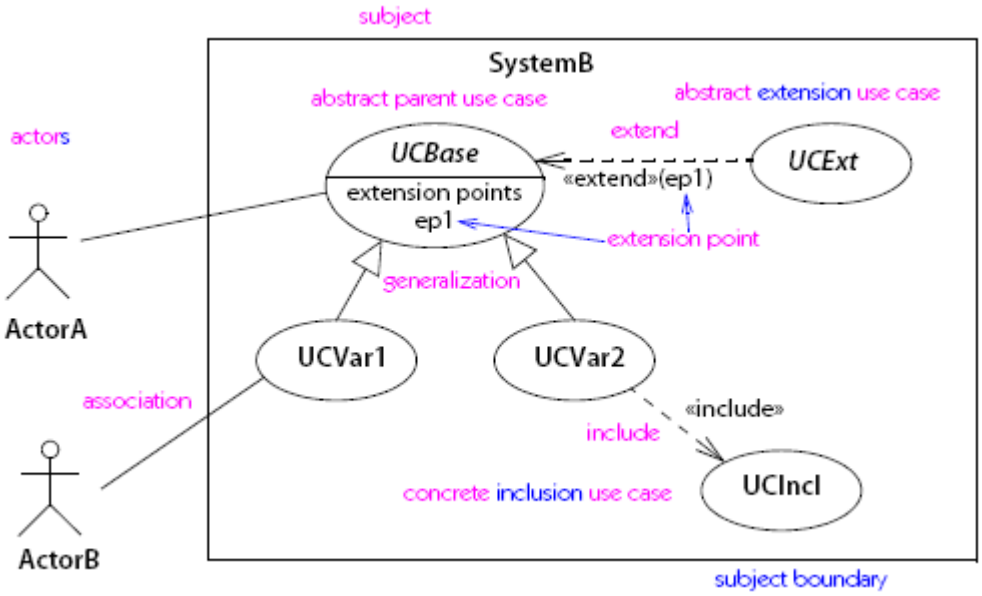
Diagram případů užití obsahuje aktéry, případy užití a jejich vzájemné vztahy. Poskytuje nám tedy informaci o tom, jakou funkcionalitu sytém nabízí, kdo ji muže vyvolávat a také stanovuje hranice modelovaného systému (aktér je mimo hranice modelovaného systému, případy užití jsou jeho součástí). Jedná se o statický pohled na systém. V diagramu není obsažena informace o tom, jak jednotlivé případy užití probíhají. To musí být popsáno v samostatném textovém dokumentu. Vztahy mezi případy užití se týkají situací, kdy se některé činnosti opakují ve více případech užití. Jedná se o následující vztahy:

Relace **include** říká, že při každém vykonávání jednoho případu užití se taktéž vykoná scénář spojený s jiným případem užití.

Relace **extend** umožňuje rozšířit chování základního případu užití o vyvolání dalšího případu užití. Musí být přitom stanoveny pozice v základním průběhu (místo rozšíření), kde k tomu dojde. Rozšiřující případy užití se nevykonávají při každém vyvolání základního případu užití. Pro jejich vyvolání musí být splněny stanovené podmínky. Tímto způsobem se vypořádáme s výjimečnými případy.

Relace zobecnění (**generalizace/specializace**) vyjadřuje vztah mezi obecnějším případem užití a jeho speciálním případem.

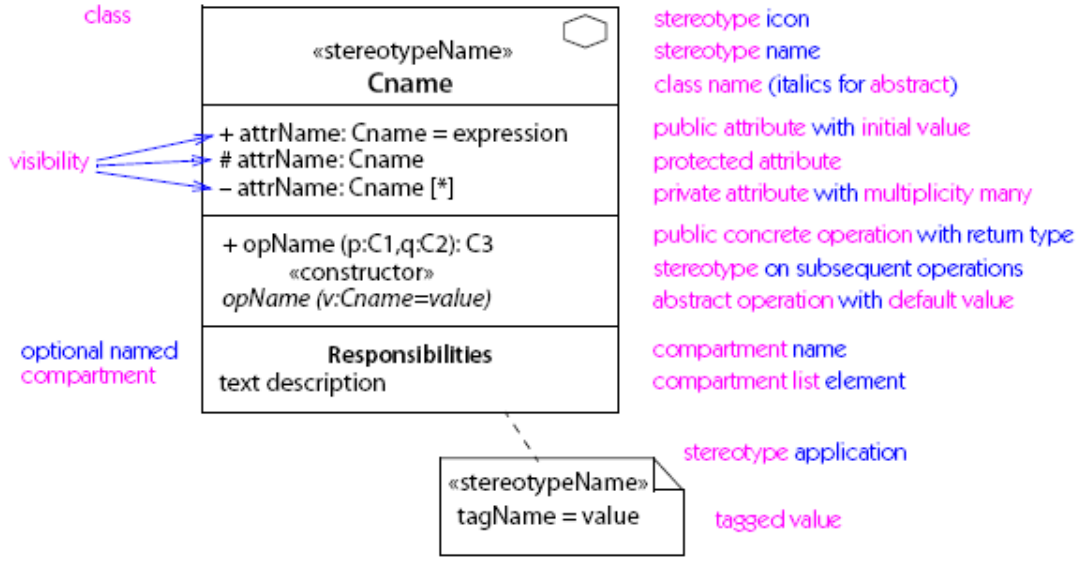
Přehled syntaxe spojené s diagramem případů užití je na následujícím obrázku



3.2.2 Třídní diagram

Třídní diagram obsahuje třídy, které se v dané problémové doméně vyskytují a jejich vztahy. Reprezentuje statický pohled na strukturu systému.

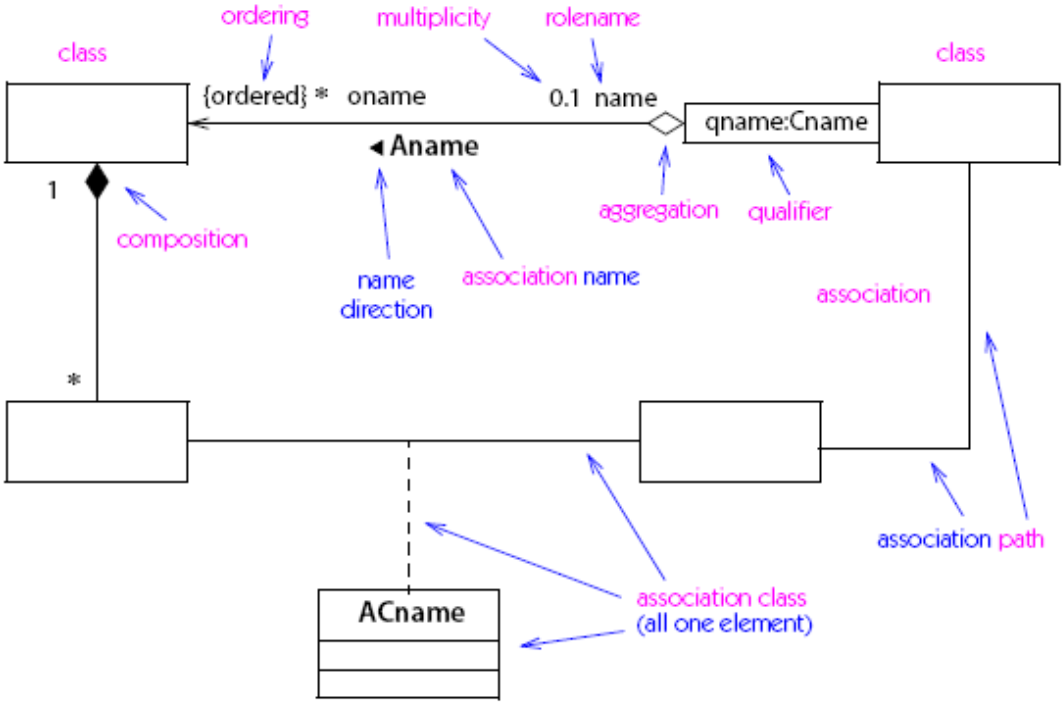
Třída je v UML reprezentována obdélníkem, který může být rozdělen do čtyř pruhů. Jsou postupně určeny pro zaznamenání jména třídy, jejich atributů a operací a zodpovědností, které má třída naplňovat. Čtvrtá část určená pro zaznamenání zodpovědností není většinou využívána. Přesná syntaxe je uvedena na následujícím obrázku.



Pro **atributy třídy** tedy můžeme zaznamenat jejich název, typ, počáteční hodnotu a viditelnost (+ public, # protected, – private). Pro **operace tříd** můžeme zaznamenat jejich jméno, názvy parametrů a jejich typy, typ návratové hodnoty a viditelnost.

Pojem **stereotyp** je určen k bližšímu zpřesnění pojmu, u kterého je uveden. Na obrázku je například uveden u jedné z operací stereotyp `<<constructor>>`. To nám poskytuje o dané operaci další informace. UML definuje řadu stereotypů, ale dává také uživatelům možnost definovat své vlastní stereotypy. Tím mohou uživatelé rozšiřovat výrazové možnosti jazyka a zavádět pojmy vyskytující se v daném prostředí.

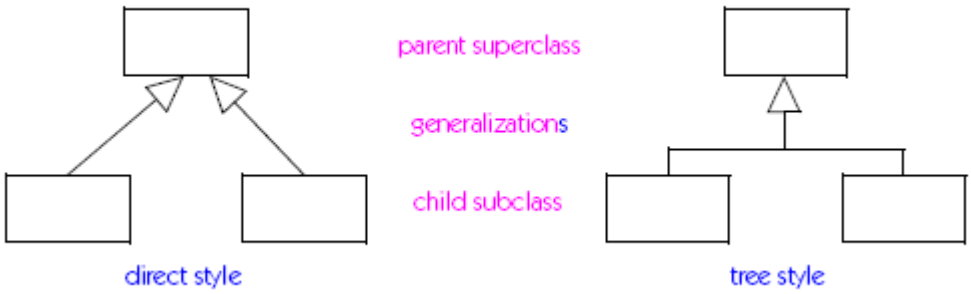
Možné vztahy mezi objekty jako instancemi tříd jsou na následujícím obrázku



Mezi základní pojmy patří

- **Asociace** říká, že jeden objekt má odkaz na druhý. Technicky to znamená, že mu může zaslat zprávu. Silnější formou tohoto vztahu jsou vztahy **agregace** a **kompozice**, které vyjadřují vztah celek-část, to je, že jeden objekt obsahuje jiný objekt. Rozdíl mezi nimi je ten, že v případě kompozice nemůže objekt, který vystupuje jako součást jiného vystupovat samostatně, to znamená, že jeho existence je spojena s existencí objektu, ve kterém je obsažen.
- **Četnost (násobnost)** zaznamenává kolik instancí asociované třídy se může napojit na jednu instanci třídy výchozí.
- **Role** umožňuje zaznamenat to, jak objekt chápe instance asociované třídy
- **Asociační třída** se zavádí v případě, že s některou asociací jsou spojeny další informace, popřípadě činnosti.

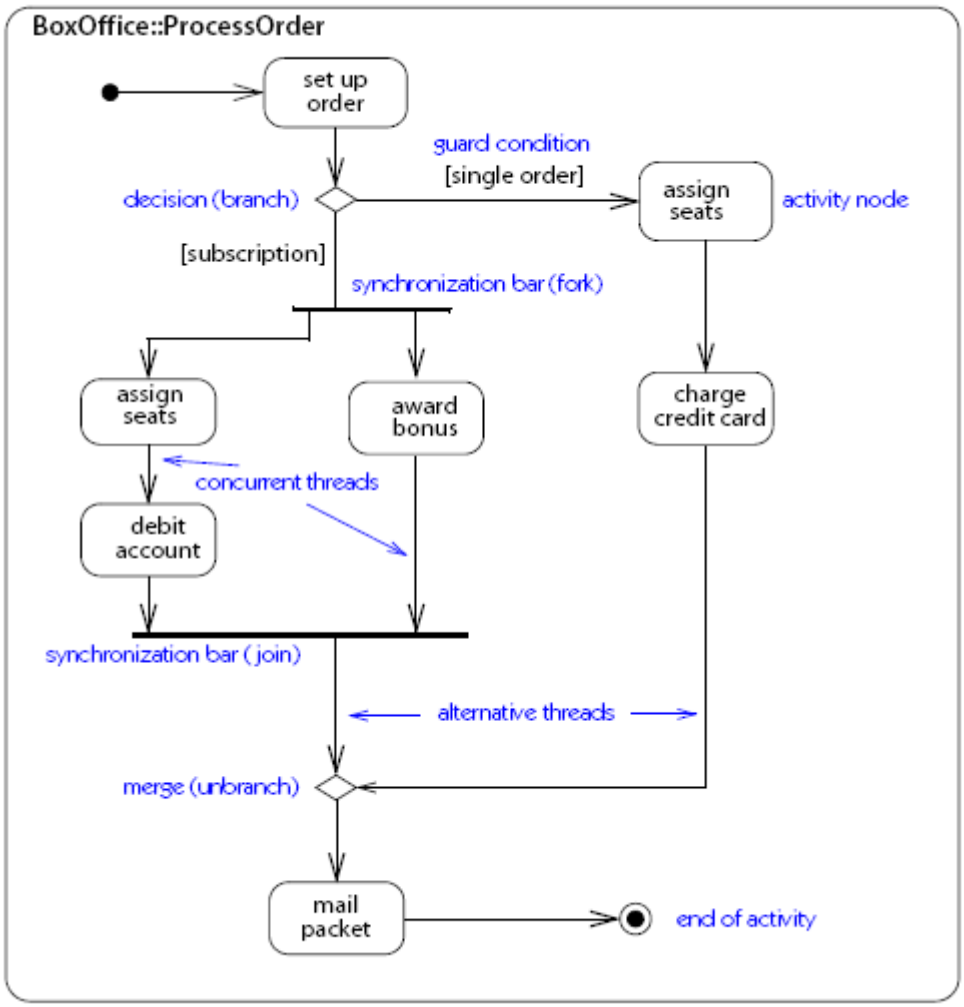
Vztahy mezi třídami jsou na následujícím obrázku



Základním pojmem je tu pojem **generalizace/specializace.** Vyjadřuje skutečnost, že jedna třída je speciálním případem jiné.

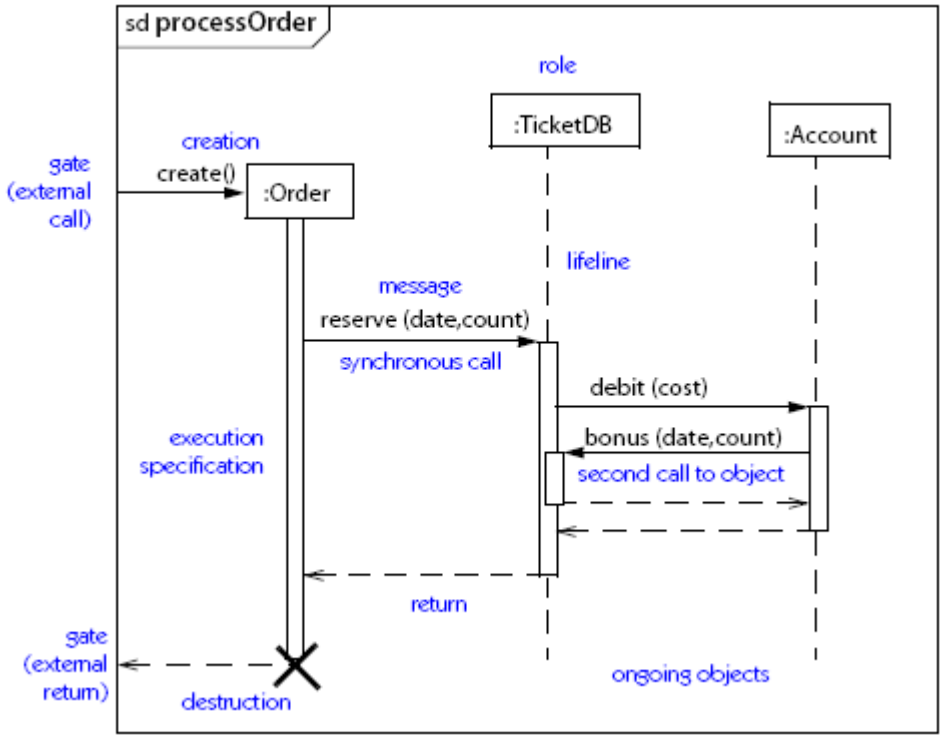
3.2.3 Diagram aktivit

Diagram aktivit je jedním z diagramů, které umožňují zachytit chování systému. Obsahuje aktivity, které jsou v systému vykonávány a jejich časovou návaznost. Umožňují zaznamenat paralelismus, to je situace, kdy jsou některé aktivity vykonávány souběžně. Jeho vytvoření je užitečné například v situacích, kdy se s daným problémem seznamujeme a potřebujeme si ujasnit, jaké činnosti v něm probíhají, jak za sebou následují a jaké datové hodnoty si předávají.



3.2.4 Sekvenční diagram

Sekvenční diagram zaznamenává interakci mezi objekty podílejícími se realizování některé funkcionality systému. Nejčastěji bývá vytvářen pro zaznamenání toho, jak systém realizuje případy užití. Zaznamenává zprávy, které si objekty zasílají, a to v podobě, která zdůrazňuje především jejich časovou návaznost.



Shrnutí

UML slouží k objektově orientovanému modelování softwarových systémů. Poskytuje 13 druhů diagramů. Mezi nejčastěji používané patří diagram případů užití, třídní diagram, diagram aktivit a sekvenční diagram.

Pojmy k zapamatování

- Model
- Třída, atribut, operace
- Asociace, agregace, kompozice
- Četnost, role
- aktivita

Kontrolní otázky

1. *Jaké charakteristiky můžeme spojit s asociací?*

2. *Pro které situace se vytváří sekvenční diagram?*

4 RUP (Rational Unified Process)

Studijní cíle: Seznámení se základními charakteristikami RUP.

Klíčová slova: Fáze, iterace, milník, disciplína.

Potřebný čas: 4 hodiny

Rational Unified Process (RUP) je příkladem moderního iterativního a inkrementálního modelu softwarového procesu. Vzniknul na konci devadesátých let jako komerční verze metody UP (Unified Process). Základní principy UP byly formulovány při vytváření jazyka UML ve firmě Rational. Za duchovního otce UP bývá označován Ivar Jacobson. V současné době je firma Rational součástí koncernu IBM.

RUP patří v současné době mezi poměrně rozšířené a známé modely softwarového procesu. Je podporován celou řadou softwarových nástrojů.

4.1 Fáze, iterace a disciplíny v RUP

RUP rozděluje životní cyklus softwarového produktu do čtyřech fází.

- **Zahájení** (Inception). Cílem této fáze je stanovit rozsah projektu a posoudit podmínky pro jeho realizaci. Během ní se řešitelé zaměřují především na:
    - Porozumění tomu, co se má vytvořit. Identifikují se všechny externí prvky systému (aktéři) a jejich hlavní požadavky na vznikající systém (klíčové případy užití)
    - Stanovení alespoň jedno možné řešení. Zvolí se alespoň jedna potenciální architektura, která umožní systém vytvořit. Implementují a ověří se rizikové prvky této architektury.
    - Zjištění případných rizik. Na konci fáze zahájení musí existovat hrubá představa o problémech, které mohou při tvorbě systému nastat.
    - Ujasnění ceny, časového plánu, potřebných zdrojů a ekonomické hodnoty projektu.
    - Stanovení softwarového procesu a používaných nástrojů

  Hlavním výsledkem této fáze je dokument vize.

- **Rozpracování** (Elaboration). Cílem této fáze je vyřešit hlavní problémy spojené vytvářením softwarového systému a implementovat základní prvky architektury systému. Během ní se řešitelé zaměřují především na:
    - Podrobné rozpracování klíčových požadavků a ověření, že zvolená architektura je pro jejich implementaci vhodná.
    - Navrhnout, implementovat a otestovat základní prvky architektury. Funkčnost na aplikační úrovni nemusí být úplná, ale rozhraní mezi základními stavebními bloky zvolené architektury musí být naimplementovány. Na konci této fáze musí být architektura ve stabilním tvaru.
    - Zpřesnění plánu vytváření celého systému

- **Konstrukce** (Construction). Cílem této fáze je detailní návrh všech částí systému, jejich implementování a testování. Během ní se praktický dokončí implementace systému. Řešitelé zaměřují především na:
    - Popsání a realizování zbývajících případů užití
    - Navržení databáze
    - Integraci a testování systému
    - Beta nasazení a zpětnou vazbu

- **Nasazení** (Transition). Tato fáze je zaměřena především na přesun systému od vývojářů k jeho uživatelům a zajištění jeho korektního a stabilního fungování v reálném prostředí. Řešitelé zaměřují především na:
    - Akceptační testy a z nich vyplývající požadavky na změny
    - Vyškolení uživatelů
    - Konvertování databází

Během každé fáze může proběhnou několik iterací. Jedna iterace nemůže být rozdělena do více fází. Během jedné iterace se vykonají všechny disciplíny a vznikne nová verze produktu obsahující některou novou funkcionalitu. Doba trvání jedné iterace je pevně stanovená a podle rozsahu projektu a velikosti řešitelského týmu trvá 1-6 týdnů. RUP nepovoluje prodloužení doby jedné iterace. Iterace se ve stanovené době ukončí, i když se nepodařilo dosáhnout požadovaných cílů. Je nutné cíle přehodnotit a začít novou iteraci. Volba cílů by měla být řízena tak, aby se pokud možno co nejdříve řešily největší problémy a snížil se stupeň rizika spojeného s projektem. RUP proto bývá označován jak vývoj řízený riziky (risc driven development).

Během každé iteraci dojde k rozšíření a zpřesnění některých artefaktů. Většina artefaktů je tedy vytvářena postupně a zcela dokončeny jsou až ve fázi nasazení.

RUP předpokládá provádění následujících disciplín:

- **Byznys modelování**
- **Specifikování požadavků**
- **Analýza a návrh**
- **Implementace**
- **Testování**
- **Nasazení**
- **Správa konfigurací a změn**
- **Řízení projektu**
- **Příprava prostředí pro projekt**

Předpokládá se, že v každé iteraci se provede v nějaké míře každá z disciplín. V různých fázích se ale jednotlivým disciplínám věnuje různá míra pozornosti. Například specifikování požadavků se věnuje největší pozornost ve fázi zahájení a na počátku fáze rozpracování.

4.2 Motivační příklad

V následujících kapitolách se budeme podrobněji zabývat některými z disciplín a způsobem jejich realizování v rámci RUP. Vedle obecného výkladu si některé prováděné aktivity budeme definovat na konkrétním příkladu malé aplikace. Jejím cílem je vytvořit softwarové prostředí podporující činnosti spojené s vypisováním požadavků pro udělení zápočtů a hodnocením jejich plnění studenty.

Uživatelské zadání je následující.

Požaduje se vytvoření softwarové aplikace, která umožní vypsání požadavků pro udělení zápočtu v jednotlivých předmětech a evidovat jejich plnění studenty. Jednotliví vyučující budou mít možnost zaznamenat požadavky pro předměty, které vyučují. Požadavky mohou mít různý charakter (příklady, účast, vystoupení na semináři, …). Některé mohou být spojeny s odevzdáním řešení (vypracované příklady, ...). Mohou být také spojeny s termínem splnění.

Řešení vypsaných požadavků budou moci studenti odevzdávat v elektronické podobě (upload) a učitel bude mít možnost odevzdaná řešení ohodnotit (schválit, přidělit nějaký počet bodů, odmítnout se zdůvodněním, …). Učitel bude mít také možnost zaznamenat plnění těch požadavků, které nemá smysl odevzdávat (účast, body za písemnou práci, …).

Jednotliví studenti budou mít přehled o tom, které požadavky již splnili a které ne. Učitel získá informace pro udělení zápočtu. Udělení ale nebude systém realizovat automaticky, musí jej potvrdit (zadat) vyučující. Učitel bude mít navíc možnost souhrnných přehledů za celý předmět. Vedení katedry bude mít možnost souhrnných přehledů za celou katedru.

V případě nesrovnalostí (stížností) aplikace umožní vedení katedry přístup k zaznamenaným informacím o odevzdaných řešeních a jejich ohodnocení.

Systém nemá nahrazovat nebo duplikovat STAG. Nebude vyžadovat zaznamenávání informací, které již jsou ve STAGu. Údaje zaznamenané ve STAGU se jednorázově načtou na počátku semestru. Na druhou stranu systém umožní sekretářce katedry výpis studentů, kterým již byl zápočet udělen, aby mohla tuto informaci zaznamenat do STAGu.

Aplikace bude provozována dlouhodobě. Bude uchovávat informace o jednotlivých předmětech. Umožní případné uznání požadavku vyřešeného v předchozím semestru.

Aplikace bude určena především pro využívání na katedře informatiky a měla by zohledňovat zavedené zvyklosti. Přístup k aplikaci bude možný pomocí internetového prohlížeče. Jednotlivým uživatelům se budou zobrazovat pouze ty informace, které se jich týkají. Zadávání informací bude realizováno především výběrem z nabízených položek. Budou dodrženy přístupová práva jednotlivých uživatelů.

Shrnutí

RUP je iterativní a inkrementální model softwarového procesu. Dělí postup prací na softwarovém produktu d čtyřech fází. Během každé fáze se může vykonat několik iterací. Doba iterace je pevně stanovena a nesmí se překročit. Během jedné iterace se vykonají v nějaké míře všechny discipliny a vznikne nová verze produktu.

Pojmy k zapamatování

- Fáze, iterace, verze
- Disciplína

Kontrolní otázky

1. *Co je náplní jednotlivých fází RUP?*
2. *Kterým disciplínám je věnována největší pozornost ve fázi zahájení a kterým ve fázi konstrukce?*

5 Specifikování požadavků

Studijní cíle: V této kapitole se seznámíme s jednotlivými druhy požadavků na softwarový systém, způsoby jejich zjištění a dokumenty, ve kterých jsou zaznamenány.

Klíčová slova:

Potřebný čas:

Inženýrství požadavků (requirements engineering) je termín označující aktivity spojené se zjišťováním, dokumentováním a udržováním kolekce požadavků na softwarový systém.

Správné stanovení požadavků je rozhodující pro konečný úspěch projektu

Každý požadavek reprezentuje vlastnost, kterou musí systém splňovat. V případě softwarových systémů požadavky stanovují, co by měl systém dělat (požadované funkce a služby) a definují podmínky a omezení pro jeho provozování (výkonnost, spolehlivost, odezva apod.) a implementování (platforma, vývojové nástroje, dodržované standardy, apod.).

Správné stanovení požadavků je jedním klíčových předpokladů pro úspěch celého projektu. Řada studií potvrdila, že nesprávné stanovení požadavků a nedostatečné zapojení uživatelů do procesu jejich specifikování jsou jednou z hlavních příčin konečného neúspěchu celého projektu. Pokud totiž systém neposkytuje uživatelům to, co chtějí, je prakticky bezcenný. Sebelepší provedení následujících aktivit (navržení a implementování systému) nemůže tento fakt změnit.

5.1 Definice a význam požadavků

V podstatě rozlišujeme požadavky na

Požadavky nám sdělují, co bychom měli vytvořit, nikoli jak bychom to měli udělat

Funkční požadavky. Stanovují funkce a služby, které bude systém poskytovat, jak bude reagovat na konkrétní vstupní údaje a jak se bude chovat v konkrétních situacích. V některých případech může být naopak stanoveno, co systém dělat nesmí. V rámci UP jsou základním prostředkem pro specifikování funkčních požadavků případy užití.

Nefunkční požadavky stanovují podmínky a omezení, které musí být při provozování a implementování systému splněny. Často se aplikují na systém jako celek. Vedle požadavků na samotný produkt (výkonnost, kapacitní nároky, spolehlivost apod.) mohou např. zahrnovat i organizační požadavky (dodržované standardy a postupy práce, způsob nasazení apod.) a požadavky dané platnou legislativou (důvěrnost, rozsah a doba zálohování dat apod.)

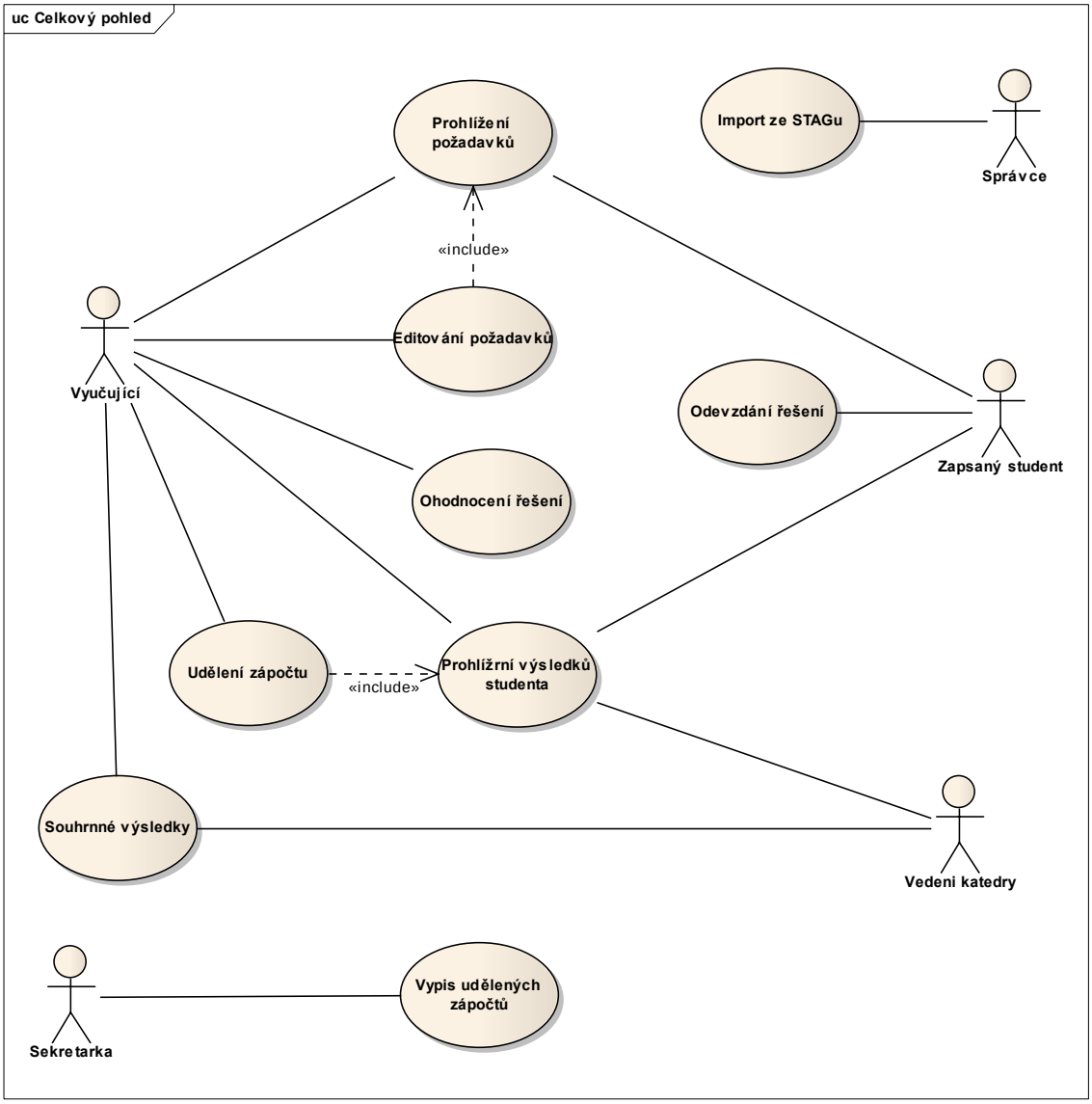
5.2 Zjišťování požadavků

Při zjišťování požadavků je pozornost zaměřena na získání maximálního množství informací o tom, co uživatelé od nového vytvářeného systému očekávají a jaké činnosti vykonávají. Mezi doporučované techniky patří:

- Studium dokumentů
- Konzultace se všemi zúčastněnými
- Dotazníky
- Pozorování

5.3 Role, artefakty a Aktivity v UP

RUP předpokládá, že funkční požadavky na řešený systém jsou zaznamenány pomocí případů Vytvoření modelu případů užití je tedy v této disciplíně jednou z klíčových aktivit. Podílí se n



5.4 Specifikování případů užití

Textový zápis stanovuje šablona

Případ užití: **Odevzdání řešení požadavku studentem**

1. **Stručný popis**

Tento případ užití umožní studentům odevzdat řešení vypsaného požadavků. Předpokládá se, že řešení je obsaženo v souboru, který student vytvoří před započetím tohoto případu užití. Soubor musí splňovat stanovená omezení (velikost, přípona, ...).

2. **Vstupní podmínky**

Před zahájením tohoto případu užití musí být student přihlášen. Musí mít také připravený soubor obsahující řešení některého požadavku.

3. **Tok událostí**

Případ užití začíná, když student zvolí nabídku Odevzdání řešení.

3.1. Základní tok

Odevzdání řešení

1. Systém zobrazí seznam názvů předmětů, na které je student zapsaný
2. Student vybere předmět
3. Systém zobrazí všechny požadavky, které byly k vybranému předmětu vypsány. Odlišně se zobrazí ty požadavky, na které již student řešení odevzdal. Mezi nimi budou dále rozlišitelné ty, které již byly učitelem ohodnoceny a uznány jako splněné.
4. Student vybere požadavek, ke kterému chce odevzdat řešení, a zvolí „odevzdat“
5. Systém zobrazí standardní dialog pro volbu souboru
6. Student vybere soubor obsahující řešení a volbu potvrdí
7. Systém zkontroluje, zda soubor splňuje stanovené podmínky. Pokud ano, pak je přenese na server. Systém pak zaznamená, že student řešení odevzdal, zobrazí o tom zprávu.

3.2. **Alternativní toky**

3.2.1 Změna odevzdaného souboru

Kroky 1. – 3. proběhnou stejně jako při základním toku událostí

4. Student vybere požadavek, ke kterém již řešení odevzdal, a který není splněný.

5. Systém zobrazí obsah souboru s posledním odevzdaným řešením.

6. Student zvolí „opravit“

Kroky 5. - 7. proběhnou stejně jako při základním toku událostí. Systém zaznamená, že student odevzdal další verzi řešení. Předchozí verze ponechá na serveru.

3.2.2 Nevyhovující soubor

V případě, že zvolený soubor nesplňuje stanovené podmínky, systém zobrazí zprávu a umožní uživateli zvolit opakovaní výběru souboru.

3.2.3. Zrušení akce

V kterémkoliv okamžiku muže student zvolit „zrušit“ a případ užití se ukončí

4. **Výstupní podmínky**

Nejsou žádné výstupní podmínky svázané s tímto případem užití.

5. **Body rozšíření**

Nejsou žádné body rozšíření svázané s tímto případem užití.

6 Analýza a návrh

Studijní cíle: V této kapitole se seznámíme především s artefakty a postupy, které RUP spojuje s disciplínou analýza a návrh.

Klíčová slova: Analýza, problémová doména, doména řešení

Potřebný čas: 4 hodiny

Cílem této disciplíny je analyzovat a navrhnout systém, který bude v definovaném implementačním prostředí vykonávat funkce specifikované v popisech případů užití.
Výsledný systém musí splňovat všechny specifikované požadavky a jeho struktura musí umožnit eventuální změny funkčních požadavků nebo omezení daných prostředím. Činnosti spojené s touto disciplínou jsou prováděny především ve fázi rozpracování.

Analýza je zaměřena především na porozumění problému a jeho řešení. Spočívá v tvorbě modelů, jež zachycují podstatné požadavky a charakteristické rysy požadovaného systému.Většina aktivit se týká tvorby modelů, které zachycují požadované chování výsledného produktu. To zahrnuje také nalezení podstatných prvků problémové domény podílejících se na realizování chování specifikovaného v případech užití, jejich vzájemné vztahy a způsob komunikace mezi nimi formou zasílání zpráv.

Omezujeme se pouze na třídy, které jsou součástí slovníku problémové domény. Zajímají nás především prvky podstatné pro stanovení struktury systému a jeho chování. Měli bychom se vyvarovat začlenění tříd, kterým neodpovídá některý pojem v problémové doméně a které jsou tedy spojeny se samotným softwarovým řešením. Tím bychom se předčasně zaměřili na konkrétní způsob návrhu a implementace a omezili bychom obecnost analytického modelu. Důležité je, aby analytickému modelu rozuměli všechny zainteresované osoby, to znamená nejenom vývojáři, ale také uživatelé.

. A by nás mohlo To by é nemají odpovídající strhlavní Cílem je

Výstupem je

Analytické třídy

Realizace případů užití

. a podíl na jejich zaznamenání ve formě analytických tříd. Většina prací spojených s

Postup při analýze

Hledání analytických tříd

V UP je zahrnuto do aktivity Analýza případů užití

Charakteristika dobrých analytických tříd

Reprezentují abstrakci problémové domény, modelují její důležité aspekty.

Měly by mapovat pojmy skutečného světa a jejich názvy by jim měly odpovídat

Je s ní spojena malá, ale správně definovaná a logicky související množina odpovědností (tj. je soudržná – high cohesion). Má jeden jasně stanovený účel a ten řeší pokud možno úplně, a ničím jiným se nezabývá.

Obsahuje malé množství vazeb na jiné třídy (low coupling)

Chyby

Funktoid – třída s jedinou operací

„všeumějící třída“ – zahrnuje velké množství různých zodpovědností. V extrémním případě reprezentuje celý systém.

Příliš mnoho vazeb na jiné třídy – každý komunikuje s každým

Postupy používané při nalezení analytických tříd a jejich vztahů

Existuje více různých přístupů k vytvoření diagramu tříd. Volba vhodného přístupu závisí mimo jiné na velikosti a typu vyvíjeného systému, zkušenostech a schopnostech řešitelů a pracovních postupech zavedených v dané organizaci. Jeden z možných přístupů je založen na navrhování realizace případů užití. Při této činnosti zvažujeme pro každý jednotlivý případ užití, které třídy jsou potřebné pro zajištění funkcionality zaznamenané v jeho popisu. Výčet těchto tříd spolu s mechanismem jejich spolupráce při realizovaní případu užití je v UML označován pojmem spolupráce (collaboration). Analytický model je tak postupně vytvářen zapracováváním jednotlivých realizací případu užití.

Jiný přístup je založen na vytvoření diagramu tříd reprezentujícího doménový model dané problematiky.Ten se vytváří jako celek, a ne po jednotlivých případech užití.

Budování doménového modelu je v podstatě iterativní proces, při kterém se postupně v jednotlivých iteracích vylepšuje a zpřesňuje. Nikomu se nepodaří jej vytvořit napoprvé. Možné kroky, kterými při jednotlivých iteracích můžeme projít popsal Rumbaugh již v roce 1991 ve své metodice OMT. Jsou to

Identifikovat objekty a třídy, které je popisují

Identifikovat atributy u tříd

Identifikovat vztahy mezi objekty (asociace) a mezi třídami (generalizace-specializace)

stanovit zodpovědnosti jednotlivých tříd

stanovit operace a atributy, které zajistí realizaci přidělených zodpovědností

Analýza podstatných jmen a sloves

Procházíme dokumenty, které popisují řešenou problematiku. Jsou především dokumenty, které již v průběhu řešení vznikly, jako je například vize systému, slovník pojmů, popis případů užití, ...). Užitečné však mohou být i další dokumenty, například odborné články, předpisy, směrnice, ...). V těchto dokumentech si pak všímáme podstatných jmen a slovesných vazeb. Podstatná jména a fráze tvořené podstatnými jmény pak zvažujeme jako možné kandidáty na analytické třídy a jejich atributy. Slovesa a slovesné fráze pak mohou odpovídat odpovědnostem a operacím tříd. Musíme však přitom být uvážliví a uvědomovat si, že v textu může být řada synonym nebo homonym, a ty pak mohou vést ke zvažování duplicitních nebo nepravých analytických tříd. Rumbaugh ve své knize doporučuje v prvním kroku zvolit větší množství tříd a v následujících krocích je „prosévat“. Pro prosévání doporučuje následující kritéria

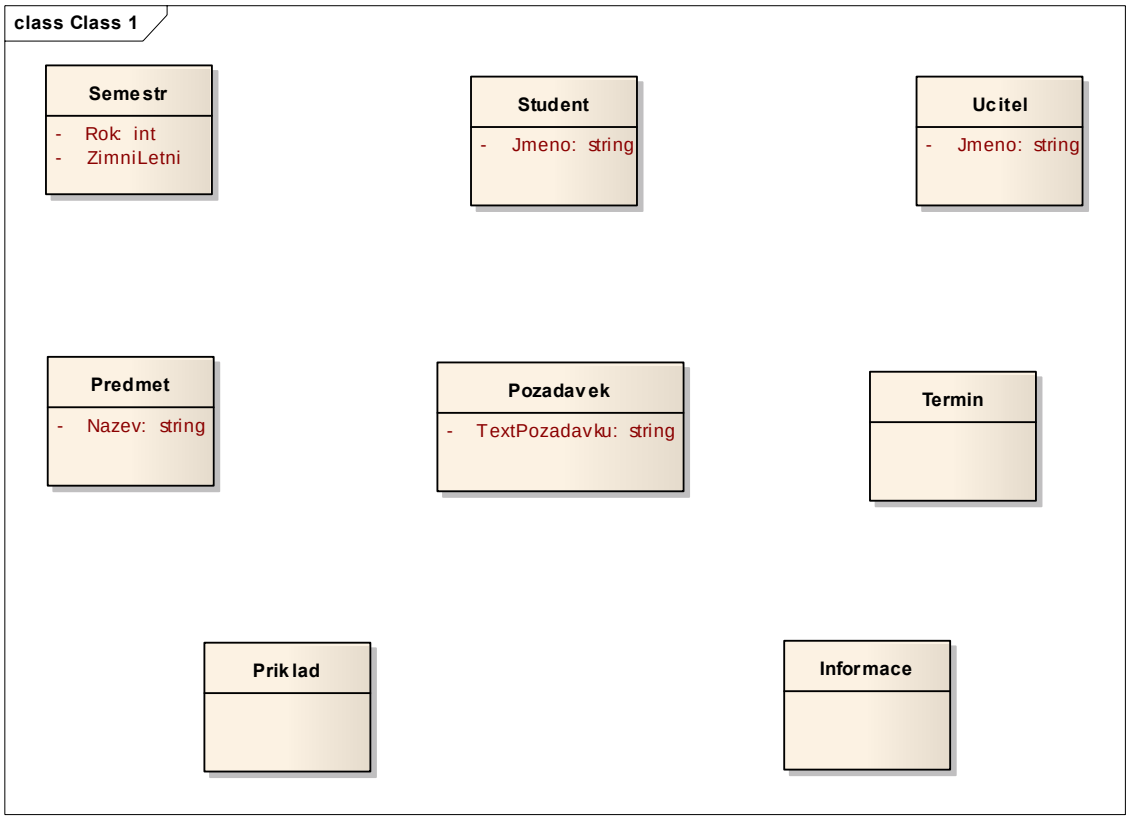
Zvážit, zda podstatné jméno reprezentuje třídu nebo pouze atribut některé třídy (barva auta)

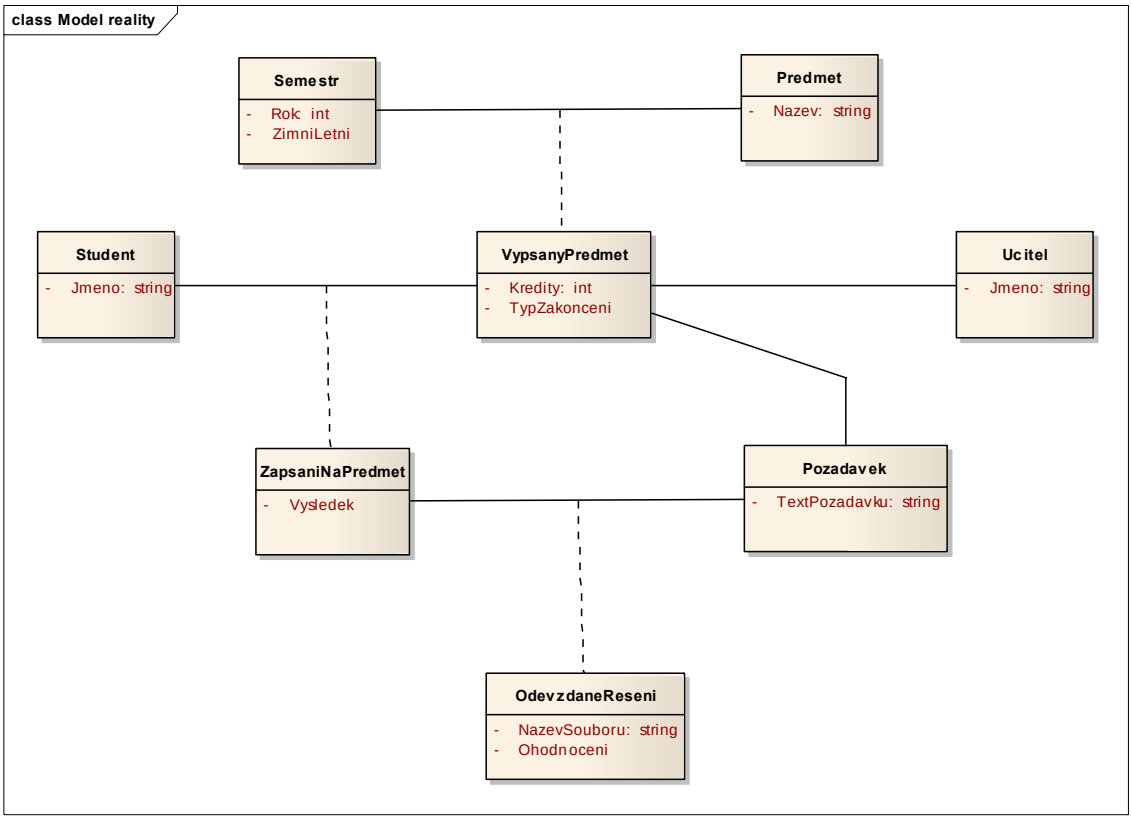
Odstranit redundantní pojmy (zákazník, uživatel)

Odstranit vágní pojmy

Odstranit pojmy reprezentující operaci nebo událost (založení, vrácení, ...)

Odstranit třídy reprezentující celý systém




class Model reality
Semestr
- Rok: int
- ZimniLetni
Predmet
- Nazev: string
Student
- Jmeno: string
VypsanyPredmet
- Kredity: int
- TypZakonceni
Ucitel
- Jmeno: string
ZapsaniNaPredmet
- Vysledek
Pozadavek
- TextPozadavku: string
OdevzdaneReseni
- NazevSouboru: string
- Ohodnoceni

7 Implementace

8 Testování

Studijní cíle: Cílem této kapitoly je popsat aktivity vykonávané při testování v rámci softwarového procesu RUP.

Klíčová slova: Testování, testér, testovací případ, plán testů, výsledky testů.

Potřebný čas: 4 hodiny

Cílem testování je ověřování kvality vyvíjeného produktu. Při jeho provádění zkoumáme všechny atributy kvality, které u dobrého software očekáváme. V rámci testování se ověřuje především:

- funkčnost – dělá aplikace to, co je požadováno?
- spolehlivost – umožňuje aplikace dlouhodobé korektní provozování?
- výkonnost – je zajištěna akceptovatelná doba reakce při vykonávání jednotlivých případů užití? Při jakém zatížení systému je možno aplikaci provozovat?

V současné době je kladen na testování velký důraz a požaduje se, aby testování probíhalo během všech fází softwarového procesu. Za velkou chybu se považuje zahájení testování až v okamžiku, kdy je implementace systému téměř dokončena. Některé metodiky požadují, aby se jako první navrhly a implementovaly testy a až poté se přistoupilo k implementaci samotné aplikace. Důvodem je skutečnost, že cena spojená s odstraněním zjištěné chyby je tím nižší, čím dříve se odhalí.

Testovací aktivity jsou nedílnou součástí softwarového procesu a stejně jako jiné aktivity by měly být plánovány, navrhovány a vykonávány členy vývojářského týmu jako součást každé iterace. Jediná testovací aktivita, která je vykonávána až ve fázi nasazení jsou akceptační testy, které provádí cílový uživatel.

8.1 Úrovně testování

Existuje celá řada testů. Následující výčet zahrnuje ty nejčastěji používané:

- **Testování jednotek (Unit-testing)** – jednotka je nejmenší část systému, kterou vývojář vytváří. Je to typicky výsledek činnosti jednoho programátora a je uložena v jednom souboru. Různé programovací jazyky chápou pojem jednotka různě. V C++ a Javě je to třída, v C je to funkce. V méně strukturovaných jazycích to může být celý program.
- **Integrační testy**. V rámci integračních testů se ověřuje vzájemná spolupráce jednotek v rámci většího celku, například subsystému nebo komponenty. Je možné, že při izolovaných testech jednotlivých jednotek na chyby nenarazíme, avšak v situacích, kdy se vzájemně volají mohou nastat problémy, například, když volající strana použije jako parametr proměnnou jiného typu, než volaná strana očekává.
- **Testy systému** se zaměřují na chyby, které se projevují na nejvyšší úrovni integrace. Mohou zahrnovat například testování bezpečnosti, spolehlivosti a dostupnosti, výkonnosti, zálohování a obnovení, jazykovou lokalizaci atd.
- **Akceptační testy** stanovuje zákazník. Jejich účelem je umožnit předpokládaným uživatelům posoudit, zda systém odpovídá jejich potřebám a očekáváním. Mohou mít

například podobu pilotního testu, při kterém je sytém experimentálně ověřován u zákazníka při běžném provozu. Do této kategorie patří také alfa a beta testování.

8.2 Techniky testování

Techniky testování mohou být klasifikovány podle různých kritérií. Nejznámější je dělení založené na informacích, které při návrhu testů použijeme:

- **Black box testování** je strategie, při níž je testování založeno pouze na znalosti specifikace požadavků na systém. Nepředpokládají se žádné informace o vnitřní struktuře software a jeho implementace. Ověřuje se především, zda systém poskytuje správné výstupy pro všechny možné vstupy. Podstatné pro hodnověrnost testů je volba reprezentativní kolekce vstupních dat. Ty musí zahrnovat nejenom běžně se vyskytující vstupní hodnoty, ale také hodnoty výjimečné, které vyžadují zvláštní ošetření. Taktéž musí být ověřena korektní reakce systému na chybné vstupní hodnoty.
- **White box** testování předpokládá znalost způsobu implementace a dostupnost zdrojové kódu. Snažíme se ověřit, že jednotlivé větve programu pracují podle očekávání. Při návrhu a provádění testů se také snažíme dosáhnout úplného pokrytí zdrojového kódu, tj. že každý řádek byl při některém z testů vykonán.

V současné době existuje celá řada nástrojů, které testování podporují a umožňují jeho automatizaci.

8.3 Role, aktivity a artefakty

RUP zavádí pro testování následující role:

- **Manažér testů** zodpovídá za celkový průběh testování. Plánuje potřebné zdroje a řeší podstatné problémy.
- **Analytik testů** je zodpovědný za identifikování a definování požadovaných testů, monitorování jejich průběhů a vyhodnocování výsledků.
- **Návrhář testů** role stanovuje používané techniky, nástroje a postupy při provádění požadovaných testů.
- **Tester** je zodpovědný za provádění stanovených testů a zaznamenání jejich výsledků.

V malé organizaci může všechny tyto role vykonávat jeden člověk.

Mezi klíčové artefakty patří:

- **Testovací případy (test case)** popisují způsob ověření jednoho požadavku na systém. Pokud se jedná o funkční požadavek je většinou spojen s odpovídajícím případem užití. Může být ale také určen k ověření stanovených podmínek a omezení, například dodržení požadovaných standardů. Testovací případy popisují jednotlivé kroky, které má tester vykonat, vstupní data, která má přitom použít a očekávané výstupy.
- **Testovací skripty** obsahují instrukce pro vykonávání testů. Mohou být vykonávány automaticky nebo manuálně.
- **Plán testů** obsahuje informace o účelu a cílech testování v rámci konkrétního projektu. Identifikuje strategie a zdroje, které budou při implementování a vykonávání testů použity.

- **Vyhodnocení testů** představuje souhrnnou analýzy výsledků testů.Vytváří se alespoň jednou v rámci každé iterace.

9 Správa verzí a řízení změn

Studijní cíle: Cílem kapitoly je pochopit činnosti spojené s touto disciplínou

Klíčová slova: Verze, změna,

Potřebný čas: 2 hodiny

10 Závěr

11 Seznam literatury

[Som04] SOMMERVILLE, I.: *Software Engineering 7*, Addison-Wesley, 2004, ISBN 0-321-21026-3

[Rob03] ROBILLARD, P., KRUCHTEN, P., D'ASTOUS, P.: *Software Engineering Process with the UPEDU*, Addison-Wesley, 2003, ISBN 0-201-75454-1.

[Arl07] ARLOW, J., NEUSTADT, I.: *UML 2 a unifikovaný proces vývoje aplikací*. Computer Press, 2003, ISBN 80-7226-947-X.

[Kad04] KADLEC, V.: *Agilní programování.* Computer press, 2004, ISBN 80-251-0342-0.

12 Seznam obrázků

13 Rejstřík