Slezská univerzita v Opavě

Filozoficko-přírodovědecká fakulta v Opavě

METODIKY VÝVOJE SOFTWARE

Studijní opora předmětu „UI/AI050 – Metodiky vývoje software“

Mgr. Jiří Martinů

Opava 2017

POKYNY KE STUDIU

METODIKY VÝVOJE SOFTWARE

Pro studium problematiky metodik vývoje software jste obdrželi studijní balík obsahující:

- toto skriptum určené pro distanční (kombinované) i prezenční studium
- přístup do e-learningového portálu Moodle
- harmonogram průběhu semestru a rozvrh prezenční části

Prerekvizity

Pro studium tohoto předmětu se nepředpokládají žádné předchozí znalosti.

.

Cílem předmětu

je seznámit studenta s metodikami návrhu a implementací softwarového projektu. Po prostudování modulu by měl student chápat smysl využívání metodik vývoje software a modelování, měl by být schopen orientace v různých metodikách, měl by umět popsat základní vlastnosti probíraných metodik a měl by rozumět rozdílu mezi agilními a rigorózními metodikami.

Pro koho je předmět určen

Modul je zařazen do bakalářského studia oboru Aplikovaná informatika studijního programu B1802 – Aplikovaná informatika, ale může jej studovat i zájemce z kteréhokoliv jiného oboru.

Skriptum se dělí na části, kapitoly, které odpovídají logickému dělení studované látky, ale nejsou stejně obsáhlé. Předpokládaná doba ke studiu kapitoly se může výrazně lišit, proto jsou velké kapitoly děleny dále na číslované podkapitoly a těm odpovídá níže popsaná struktura.

Při studiu každé kapitoly doporučujeme následující postup:

Čas ke studiu: xx hodin

Na úvod kapitoly je uveden čas potřebný k prostudování látky. Čas je orientační a může vám sloužit jako hrubé vodítko pro rozvržení studia celého předmětu či kapitoly. Někomu se čas může zdát příliš dlouhý, někomu naopak. Jsou studenti, kteří se s touto problematikou ještě nikdy nesetkali a naopak takoví, kteří již v tomto oboru mají bohaté zkušenosti.

Cíl: Po prostudování tohoto odstavce budete umět

- Popsat …
- Definovat …
- Vyřešit …

Následují cíle, kterých máte dosáhnout po prostudování této kapitoly – konkrétní dovednosti, znalosti.

Výklad

Kapitola pokračuje výkladem studované látky, zavedením nových pojmů, jejich vysvětlením; v případě potřeby je látka doplněna obrázky, tabulkami, odkazy, apod.

Shrnutí pojmů

Na závěr kapitoly jsou zopakovány hlavní pojmy a části kapitoly, které si v ní máte osvojit. Pokud něčemu ještě nerozumíte, vraťte se k výkladové části kapitoly.

Otázky

Pro ověření, že jste dobře a úplně látku kapitoly zvládli, máte k dispozici několik teoretických otázek.

Úspěšné a příjemné studium s tímto učebním textem Vám přeje autor.

Jiří Martinů

OBSAH

1 ÚVOD DO PROBLEMATIKY: SPECIFIKACE POJMŮ METODOLOGIE, METODIKA (CÍL METODIK), METODA, ROZDĚLENÍ METODIK PRO VÝVOJ SW. PRIMÁRNÍ DŮVODY MODELOVÁNÍ. ŽIVOTNÍ CYKLY VÝVOJE SW. LEHKÉ A TĚŽKÉ METODIKY.

V této kapitole se seznámíme se základními pojmy oblasti metodik vývoje software a s důvody, proč se metodiky a modelování používají. Dozvíme se rovněž, podle jakých kritérií metodiky dělíme a objasníme pojem *životní cyklus* vývoje software.

Čas ke studiu: 2 hodiny

Cíl: Po prostudování této kapitoly budete umět

- Definovat základní pojmy z oblasti metodik vývoje software a modelování.
- Objasnit důvody modelování.
- Objasnit pojem *životní cyklus* při vývoji software.
- Rozdělit metodiky vývoje software podle různých kritérií.
- Objasnit rozdíly mezi tzv. *lehkou* a *těžkou* metodikou.

Výklad

1.1 Základní terminologie

- **metoda** (z řeckého met-hodos – cesta za něčím) je **návod či postup** získávání poznatků. Jinými slovy jde o **prostředek poznání**. Odpovídá na otázku: **JAK** dosáhnout vytýčeného cíle? Zaměřuje se na určitou etapu vývoje SW.
- **metodika** obecně je nauka o metodě nebo pracovní postup. Jedná se o doporučený souhrn etap, přístupů, zásad, postupů, pravidel, metod, technik, nástrojů, dokumentů, metod řízení, atd. Odpovídá na otázky typu **CO, KDY KDO, PROČ**. Ve vývoji software **představuje metodika souhrn doporučených praktik a postupů,**

pokrývajících celý životní cyklus vytvářené aplikace. Pro řešení dílčích problémů se v rámci nasazení metodik uplatňují specifické postupy – metody.

- **metodologie** je **vědní disciplína**, zabývající se metodami, jejich tvorbou a aplikací.
- **technika – takto nazýváme** postupy kroků, jak se dobrat k výsledku, např. normalizace datového modelu, prototypování.
- **nástroj –** prostředek k provedení něčeho, k zobrazení výsledku apod. Jedná se např. o modely systému, CASE nástroje.
- **projekt –** množina souvisejících činností určených pro splnění daného cíle. Projekt musí mít přiděleny zdroje (lidské zdroje, finanční zdroje, čas…) a je nutné zajistit jeho řízení (manažer). Projekt má rovněž svého zadavatele (odběratele).

1.2 Rozdělení metodik pro modelování SW

Metodiky je možné klasifikovat podle různých kritérií a přístupů, jako např. podle zaměření, rozsahu metodiky, váhy metodiky, typu řešení, domény, přístupu, atd. Stručný přehled rozdělení metodik je uveden v následující tabulce, v dalším textu je pak uveden jejích bližší popis:

Tab. 1.1 – Rozdělení metodik podle různých kritérií

Kritérium	Kategorie/ vysvětlení
zaměření metodiky	globální projektové
rozsah metodiky	fáze – role - dimenze
váha metodiky	velikost metodiky x hustota metodiky
typ řešení	vývoj nového řešení, rozšíření stávajícího řešení, integrace řešení, implementace typového řešení, užití řešení například formou ASP
doména	ERP, CRM, SCM, EAI…
přístup k řešení	Strukturovaný, objektově orientovaný…

1.2.1 Přehled jednotlivých kritérií rozdělení metodik

V této podkapitole uvádíme stručný popis rozdělení metodik podle kritérií uvedených v tabulce 1.

- **Kritérium zaměření metodiky:**

Globální - metodiky vývoje SW v rámci celého podniku či organizace. Patří mezi ně tzv. enterprise metodiky, např. MMDIS, Enterprise Unified Process (EUP), atd.

Projektové – v rámci vývoje daného SW se zabývají pouze určitou částí SW.

- **Kritérium rozsahu:**

Rozeznáváme metodiky, které se zabývají **celým životním cyklem** vývoje SW (např. MMDIS, atd.) a metodiky, zabývající se jen **určitou částí** (etapou) vývoje SW. Posledním uvedeným metodikám říkáme **dílčí metodiky**.

- **Kritérium váhy (podrobnosti) metodiky:**

Těžké metodiky – vyžadují podrobný popis a jsou tzv. rigorózní, tzn. přísné, precizní, přesné.

Lehké metodiky – jejich vlastností je, že musí být „barely sufficient“ (Cockburn), volně přeloženo jako „sotva (téměř) dostatečné“ anebo „a little less than just enough“ (Highsmith), což je možno volně přeložit jako „trochu méně než dostatečné“.

Mezi lehké metodiky patří tzv. **agilní** metodiky, o kterých bude řeč dále.

- **Kritérium přístupu k vývoji:**

Podle kritéria přístupu k vývoji můžeme metodiky rozdělit na:

Strukturované metodiky – založené na principech strukturovaného programování

RAD (Rapid Application Development) – založený na iterativním přístupu

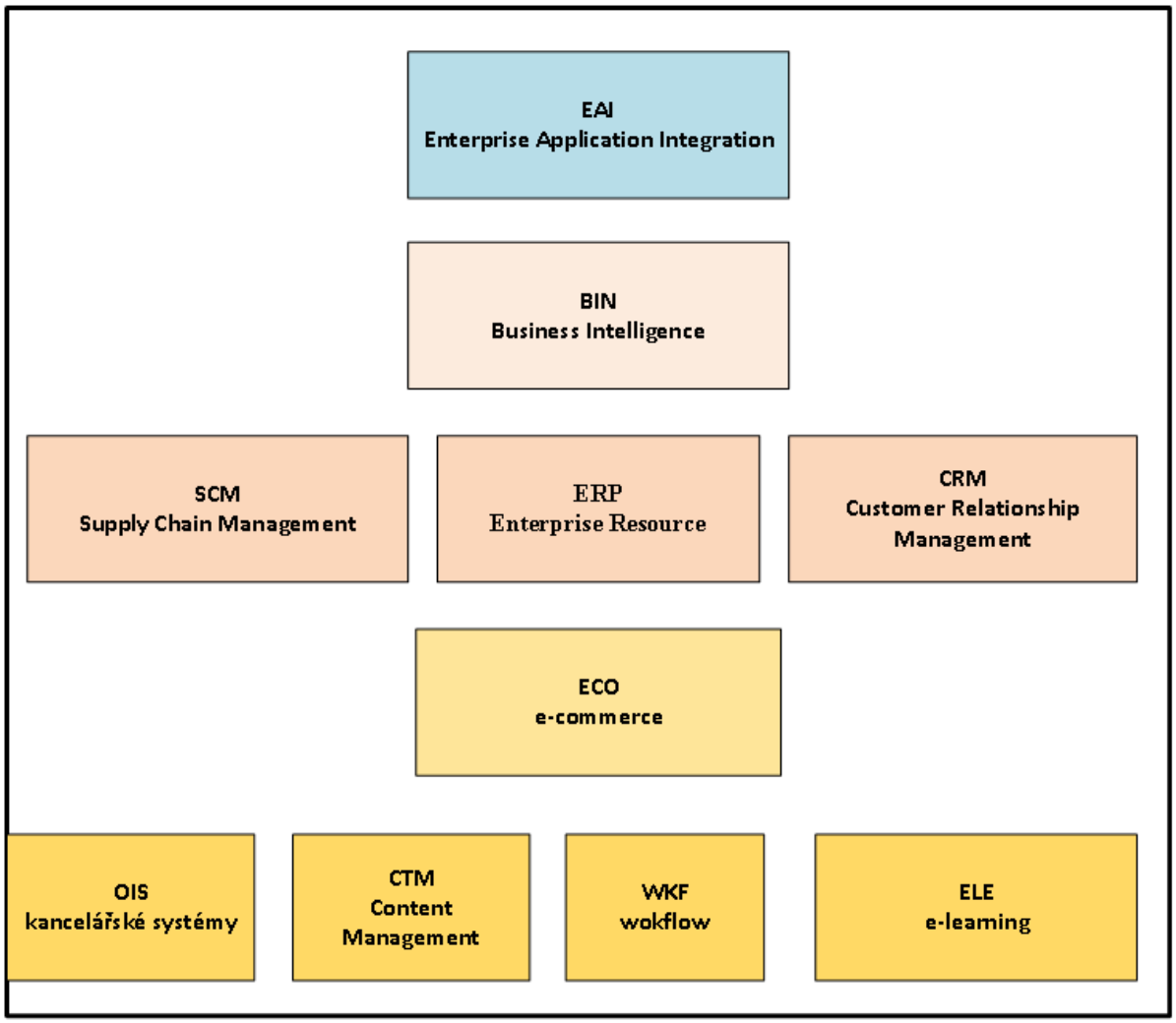
Objektově orientované metodiky – založené na principech objektově orientovaného programování

- **Kritérium způsobu vývoje:**

Toto kritérium dělení metodik zahrnuje zejména:

  - konvenční metodiky s **životním cyklem typu *vodopád*** (viz dále)
  - metodiky **přírůstkového** a **iterativního** vývoje

- **Kritérium domény:**



Obrázek 1-1Rozdělení metodik vývoje software podle kritéria domény

1.3 Primární důvody modelování

Důvodů pro modelování při vývoji SW je celá řada. Modelování slouží k návrhu vztahů mezi dílčími částmi navrhovaného SW systému (např. modulů, objektů), jejich přehlednému zobrazení, návrhu spolupráce a interakce mezi nimi, k návrhu vstupů a výstupů jednotlivých komponent a celkové analýze vyvíjeného SW.

Díky vyspělým modelovacím a analytickým nástrojům poskytuje modelování účinný způsob, jak zabránit vzniku a zdlouhavému a neefektivnímu odstraňování chyb vycházejících z „nedomyšlených“ částí kódu. Odhaluje slabá místa navrhovaného systému.

Modelování poskytuje rovněž podpůrný prostředek pro tvorbu dokumentace. Usnadňuje spolupráci mezi jednotlivými členy vývojového týmu jasným vymezením úkolů.

Modelování vymezuje vznik a životní cykly objektů, procesů a dalších komponent a vede k vytvoření správně navrženého, přehledného, bezpečného a dobře zdokumentovaného SW produktu.

1.4 Životní cykly vývoje SW

Pojem životní cyklus je nedílnou součástí každého projektu zaměřeného na vývoj SW:

Pod tímto pojmem rozumíme návaznost jednotlivých etap procesu vývoje. Etapy si můžeme představit jako jednotlivé části skládačky (např. kostky, které skládáme do nějakého výsledného tvaru). Každá část skládačky má své definované vlastnosti, jakými jsou např. vstupy a výstupy, data dokončení, atd. Bez hotové části skládačky neleze přidat její další díl, neboli bez ukončené etapy nelze pokračovat v další etapě. Výsledný tvar utvořený z těchto jednotlivých dílů pak tvoří životní cyklus vývoje SW.

Existuje celá řada obecně uznávaných etap životního cyklu vývoje SW. Zde uvádíme 2 nejpoužívanější:

(Řepa, V. Analýza a návrh informačních systémů. Ekopress, Praha 1999, ISBN 80-86119-13-0. str. 17-19):

- cíl etapy (proč etapa má být provedena a co je jejím výsledkem),
- účel a obsah etapy (popis role etapy v celém vývoji systému, shrnutí činností prováděných v etapě),
- předpoklady zahájení etapy (kdy je možné začít s pracemi v rámci dané etapy),
- kritéria ukončení etapy (kdy je možné prohlásit etapu za ukončenou),
- klíčové dokumenty etapy (seznam dokumentů, vyprodukovaných nebo upravených v dané etapě, které musí být schváleny vedením projektu),
- kritické faktory etapy (na co je třeba v etapě dávat největší pozor, faktory, které mohou způsobit problémy při vývoji),
- činnosti etapy (seznam a popis činností, které se v etapě provádějí),
- návaznosti činností v etapě (graficky vyjádřená návaznost a souběžnost provádění jednotlivých činností etapy).

(Kendall, K.E. Systém Analysis and Design. Prentice Hall, 1991):

- zachycení požadavků na systém (týká se funkčnosti, designu, návaznosti na jiné systémy, integrace s ostatními systémy, reakční doby atd.),
- tvorba konceptuálního modelu (zachycení skutečností v rámci modelu),
- tvorba implementačního modelu (jedná se již o konkrétní návrh IS),

- implementace a zavedení,
- testování,
- udržování systému a provoz
- stažení systému z užívání.

Shrnutí pojmů 1.1.

V této kapitole jsme se seznámili se základní terminologií z oblasti vývoje software. Objasnili jsme pojmy jako:

- metoda
- metodika
- metodologie
- technika
- nástroj
- projekt

Uvedli jsme si rozdělení metodik do kategorií dle následujících kritérií:

- zaměření metodiky
- rozsah metodiky
- váha metodiky
- typ řešení
- doména
- přístup k řešení

V další části jsme se naučili rozlišovat pojmy lehká a těžká metodika a seznámili se s pojmem životní cyklus a důvody modelování.

Otázky 1.1.

1. Objasněte pojmy metoda, metodika, projekt, technika, nástroj
2. Jaké jsou hlavní vlastnosti projektu, co musí splňovat?
3. Proč provádíme modelování?

4. Co znamená termín „životní cyklus“ vývoj SW?
5. Jaká znáte kritéria rozdělení metodik?

2 VODOPÁDOVÝ PŘÍSTUP K TVORBĚ SW: PRINCIP MODELU, ŽIVOTNÍ CYKLUS, SPECIFIKACE, PLÁN, FÁZE VÝVOJE, MOŽNOSTI POUŽITÍ, NEVÝHODY.

V této kapitole se seznámíme s vodopádovým modelem vývoje SW, jeho vlastnostmi, výhodami a nevýhodami.

Čas ke studiu: 2 hodiny

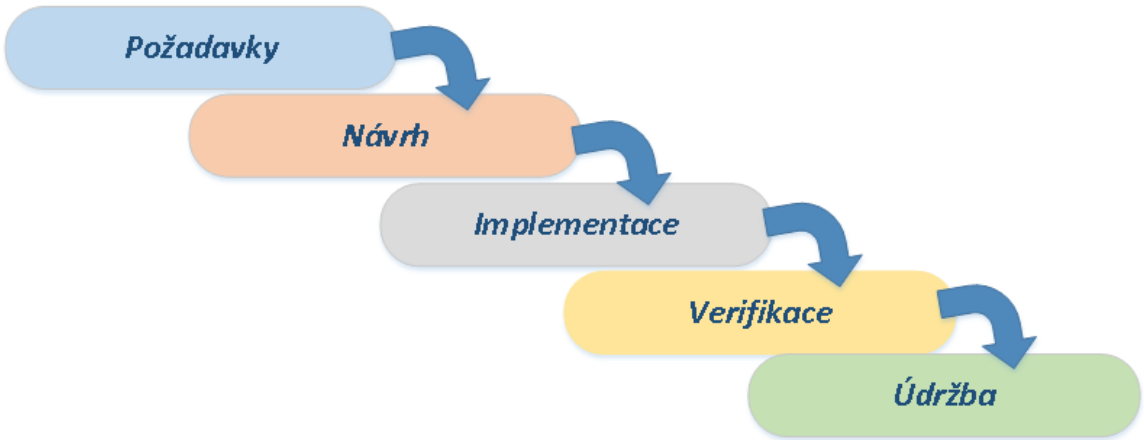
Cíl: Po prostudování této kapitoly budete umět

- Definovat pojem vodopádový model.
- Objasnit hlavní vlastnosti vodopádového modelu.
- Vyjmenovat fáze vodopádového modelu.
- Objasnit výhody a nevýhody vodopádového modelu.

Výklad

2.1 Model vodopád

znázorňuje určitý **idealizovaný** stav – posloupnost na sebe navazujících etap **bez cyklických návratů zpět**. V praxi by bylo vhodné jej dodržovat, většinou to však není možné, proto má tento model význam spíše teoretický a slouží jako základní myšlenkový postup pro studium etap životního cyklu.



Obrázek 2-1Model vodopád

Jak vyplývá z obrázku 1, model vodopád má následující fáze (etapy):

- specifikace zadání, stanovení požadavků
- analýza a návrh SW
- implementace – nasazení SW
- verifikace – testování a integrace
- provoz a údržba

2.2 Výhody a nevýhody vodopádového modelu

Výhody vodopádového modelu

- jednoduchý z hlediska řízení
- při stálých požadavcích: nejlepší struktura výsledného produktu

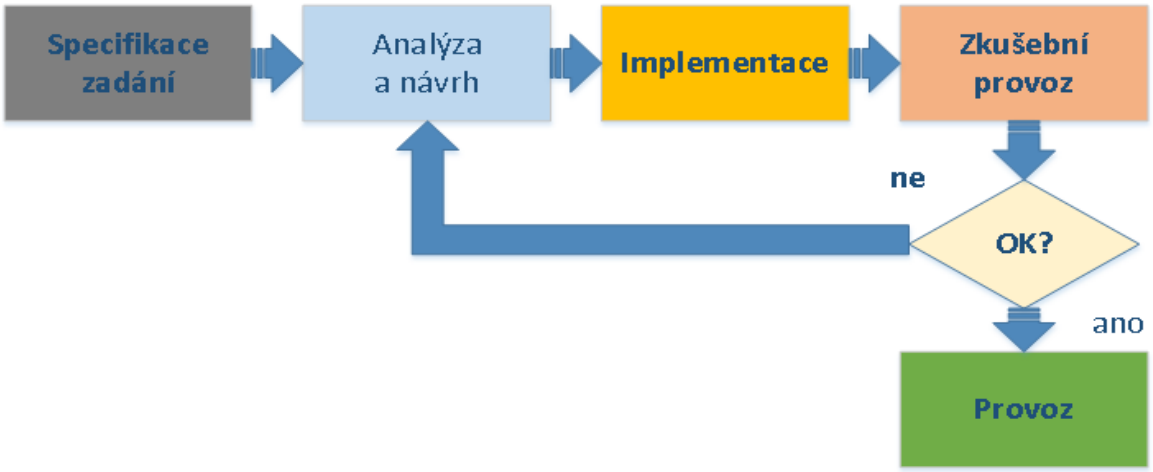
Nevýhody vodopádového modelu

- zákazník není schopen přesně stanovit veškeré požadavky předem
- při změnách požadavků má tento model dlouhou dobu realizace
- zákazník vidí spustitelnou verzi až v závěrečných fázích projektu, z čehož vyplývá, že např. nedostatky jsou odhaleny příliš pozdě (fáze verifikace) a jejich odstranění vede k navýšení čerpání zdrojů.

Dalšími typy modelů vývoje SW jsou modely **výzkumník** a **prototyp**.

2.3 Model výzkumník

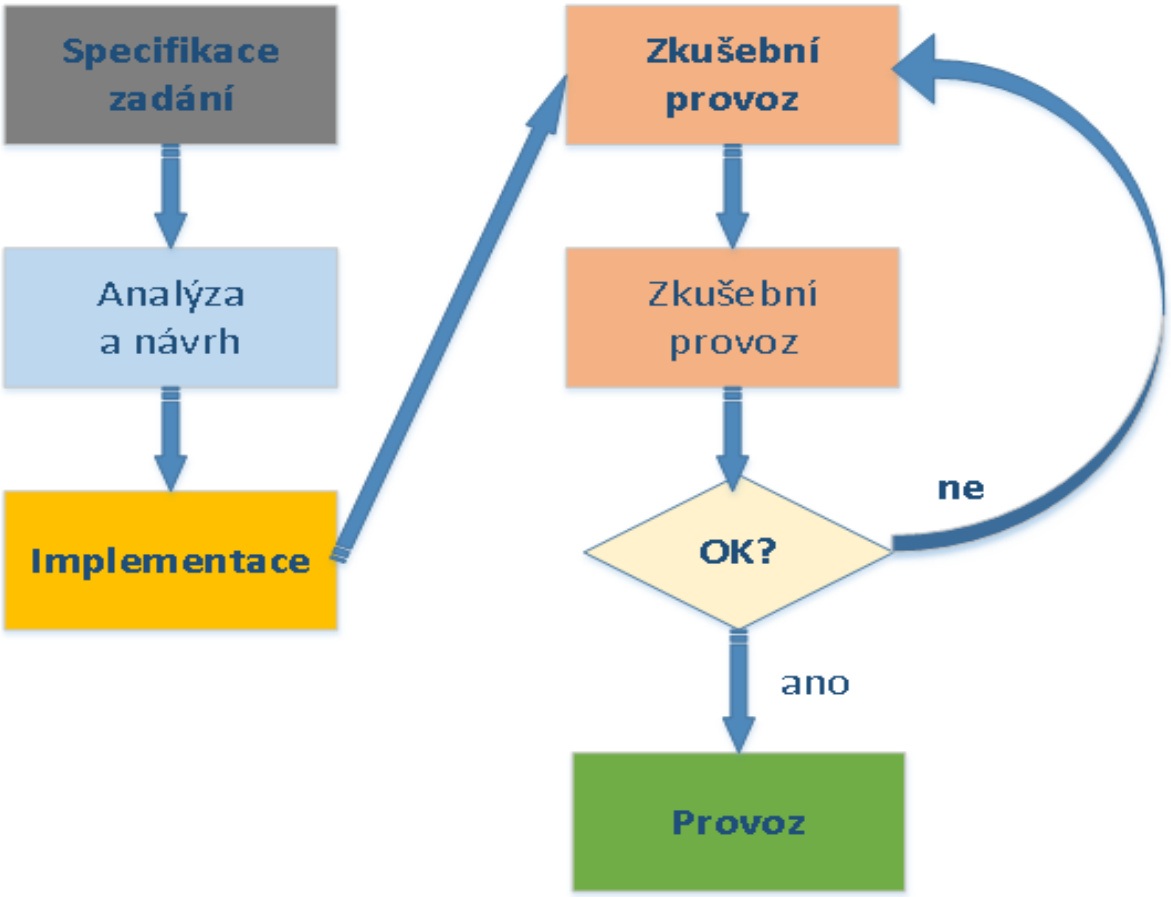
je uváděn spíše jako negativní případ přístupu k vývoji IS. Jeho použití svědčí o tom, že řešitelský tým neovládá dobře danou problematiku, získává postupně zkušenosti v oblasti, pro kterou je IS určen. Za takovýchto okolností je doba etap těžce plánovatelná. U rozsáhlejších IS lze takový přístup použít (bez negativních následků) jen stěží.



Obrázek 2-2 Model výzkumník

2.4 Model prototyp

jde o aplikaci plánovaného a řízeného inkrementálního přístupu k vývoji IS. Pod pojmem „prototyp“ rozumíme částečnou implementaci produktu (části IS) v logické nebo fyzické formě, která by měla reprezentovat všechna vnější rozhraní systému. Uživatelé IS se s prototypem seznamují a testují jej. Na základě jejich připomínek je upřesňována specifikace požadavků na systém, prototyp je upravován a dále doplňován až do podoby výsledného systému. Na rozdíl od typu „výzkumník“ zde vycházíme z řádné analýzy a návrhu systému, jehož součástí je i rozsah prototypu v jednotlivých stupních vývoje. Stupně vývoje prototypu jsou plánované a jeho vývoj je řízen podle předem stanovených pravidel.



Obrázek 2-3 Model prototyp

Shrnutí pojmů 2.1.

V této kapitole jsme si objasnili pojem vodopádový model.

Uvedli jsme jeho hlavní etapy:

- specifikace zadání, stanovení požadavků
- analýza a návrh SW
- implementace – nasazení SW
- verifikace – testování a integrace
- provoz a údržba

Dozvěděli jsme se také, jaké jsou jeho výhody a nevýhody a důvody, proč vodopádový model není v praxi příliš využíván.

Otázky 2.1.

1. Kdy se používá model vodopád?
2. Jaké znáte fáze modelu vodopád?
3. Jaké jsou výhody/nevýhody modelu vodopád?

3 ITERAČNÍ/EVOLUČNÍ PŘÍSTUP K TVORBĚ SW: PRINCIP MODELU, ITERACE, ZPĚTNÁ VAZBA, FÁZE ŽIVOTNÍHO CYKLU. SROVNÁNÍ VODOPÁDOVÉHO A ITERAČNÍHO PŘÍSTUPU.

V této kapitole se seznámíme s iterativním přístupem k vývoji software a se spirálovým modelem a jeho vlastnostmi. Uvedeme rovněž hlavní etapy spirálového modelu a rozdíly mezi vodopádovým a spirálovým modelem vývoje SW.

Čas ke studiu: 2 hodiny

Cíl: Po prostudování této kapitoly budete umět

- Definovat pojem iterace (iterační přístup).
- Objasnit hlavní vlastnosti spirálového modelu.
- Vyjmenovat základní etapy spirálového modelu.
- Uvést hlavní rozdíly vodopádového a spirálového modelu.

Výklad

3.1 Iterace

Iterace = opakování. Model zahrnuje vývoj SW v iteracích. Tento iterativní přístup má následující vlastnosti:

- v průběhu každé iterace je vytvořen reálný výsledek. Zadavatel má po každé iteraci příležitost zhodnotit a zkontrolovat výsledek a srovnat jej se svými požadavky. To má za následek rychlejší a pružnější odhalení chyb ve specifikaci
- zadavatel je účastníkem vývoje, členem týmu. Podílí se minimálně na kontrolních bodech, tzv. **milestones**.
- Iterativní přístup klade vyšší nároky na řízení v porovnání např. s vodopádovým (neiterativním) modelem.

- Vzhledem k vyšším nárokům na řízení a změny během jednotlivých iterací má iterativní model potenciálně horší výslednou strukturu.

Příkladem modelu s iterativním přístupem je **spirálový model**.

3.2 Spirálový model

Spirálový model je odvozen od vodopádového modelu. Zásadním způsobem však mění přístup a oproti vodopádovému modelu má dvě odlišné vlastnosti:

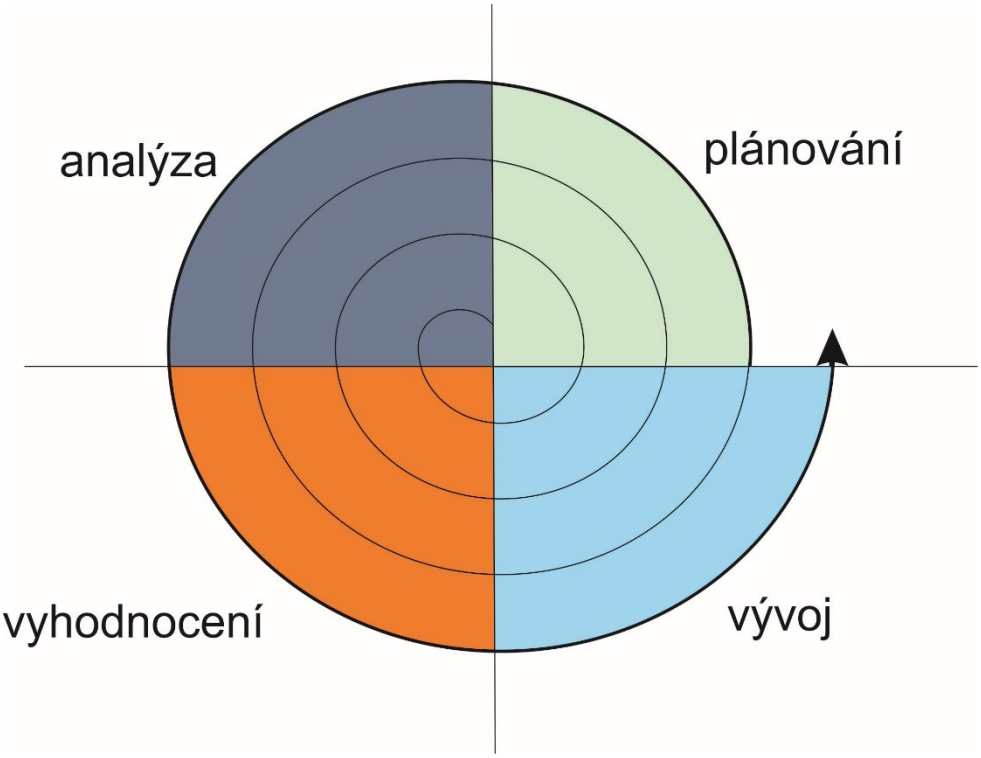
- iterativní přístup
- podrobná analýza rizik

Pokud u spirálového modelu chceme přejít do další fáze (etapy), je nutné provést důslednou analýzu možných problémů a rizik. Analýza rizik je prováděna v každém cyklu a je určujícím faktorem pro další směr projektu. Pod pojmem riziko se rozumí libovolná událost či situace, která by mohla mít dopad na projekt, a která by jej mohla ohrozit. Každému riziku je v průběhu analýzy přiřazena jeho nebezpečnost a pravděpodobnost výskytu.

3.3 Srovnání vodopádového a spirálového modelu

Vodopádový model se stal téměř nevyhovujícím v případech, kdy bylo nutné stanovit úplnou a přesnou specifikaci požadavků. Lepším přístupem na počátku je pouze stanovení rámce architektury a funkčnosti navrhovaného systému, což umožňuje např. spirálový model. V průběhu vývoje jsou pak upřesňovány a rozpracovávány jednotlivé detaily. Tento iterativní přístup, který lze chápat jako cyklické opakování, se v době svého vzniku stal přelomovým přístupem v chápání životního cyklu vývoje software.

Spirálový model je možné rozdělit na čtyři základní etapy. Pro lepší představu si jednotlivé etapy můžeme znázornit ve formě kvadrantů, obrázek 2:

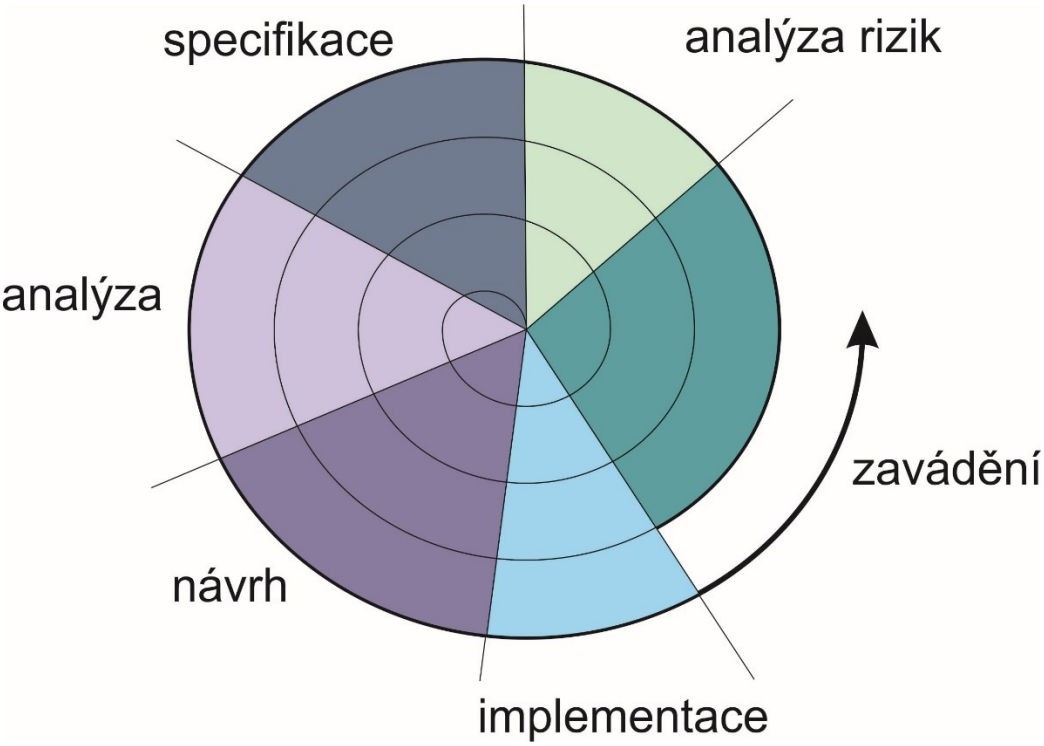


Obrázek 3-1Spirálový model

- **analýza** - v této etapě se stanovují cíle, alternativy a rozsah cyklu (iterace).
- **vyhodnocení** – v této etapě jsou vyhodnoceny alternativy, řešení rizik a jejich identifikace.
- **vývoj** – tato etapa slouží k vývoji produktu a ke kontrole očekávaných výsledků.
- **plánování** – je etapa, ve které se plánuje následující iterace. Na počátku každého cyklu jsou identifikovány subjekty mající vliv na průběh iterace, včetně jejich podmínek ovlivňujících úspěch iterace. Po dokončení cyklu je provedena revize a předání.

Při každém průchodu spirálou dochází k rozpracovávání dané problematiky na stále vyšším stupni.

Kromě toho základního dělení na 4 etapy se můžeme setkat i s dělením spirály na podrobnější etapy – specifikace, analýza, návrh, implementace, zavádění, analýza rizik. Princip zůstává, samozřejmě, stejný jako u základního spirálového modelu – při každém průchodu etapou dochází k detailnějšímu rozpracování a zjemňování návrhu. Graficky lze tento model znázornit jako na obrázku 3-2.



Obrázek 3-2 Jiný typ spirálového modelu

Shrnutí pojmů 3.1.

V této kapitole jsme si objasnili další z modelů – spirálový model a uvedli si jeho hlavní vlastnosti a výhody v porovnání s vodopádovým modelem. Uvedli jsme si čtyři hlavní etapy spirálového modelu:

- analýza
- vyhodnocení
- vývoj
- plánování

Otázky 3.1.

1. Co znamená pojem iterativní přístup?
2. Jaké jsou nejdůležitější rozdíly mezi vodopádovým a spirálovým modelem?
3. Uveďte hlavní etapy spirálového modelu a popište, co se během těchto etap provádí.

4 METODIKA UP: MODELOVACÍ PROCES UP (UNIFIED PROCESS), STRUKTURA JAZYKA UML (UNIFIED MODELING LANGUAGE), NEJPOUŽÍVANĚJŠÍ DIAGRAMY JAZYKA UML, DALŠÍ PRVKY UML, VZTAH UP A UML.

V této kapitole se seznámíme s modelovacím procesem UP, strukturou a významem jazyka UML, a uvedeme si diagramy jazyka UML využívané k modelování vývoje software. Zvládnutí kapitola bude, vzhledem k počtu diagramů UML a celkovému obsahu, časově náročnější – doporučujeme proto rozdělit si učivo do menších celků, učit se postupně a zkusit si rovněž diagramy prakticky nakreslit.

Čas ke studiu: 6 hodin

Cíl: Po prostudování této kapitoly budete umět

- Objasnit principy metodiky UP.
- Uvést, k čemu se využívá UML.
- Vyjmenovat základní diagramy UML a objasnit jejich účel.
- Orientovat se ve vztahu mezi UP a UML.

Výklad

4.1 Metodika UP

Metodika UP – Unified Process (unifikovaný nebo jednotný proces) vychází z průmyslového standardu SEP – Software Engineering Process. UP je jen zkrácené označení průmyslového standardu USDP – Unified Software Development Process (unifikovaný proces vývoje software). Jednoduše řečeno, jedná se o generický proces pro jazyk UML – Unified Modeling Language (unifikovaný modelovací jazyk). Jde pouze o **základní, generickou a otevřenou metodiku, kterou je možné uzpůsobit jakémukoli rozsahu projektu**.

Metodika UP musí být přizpůsobena cílům a podmínkám daného vývoje, tzn. používaným normám, šablonám, nástrojům, atd.

Unifikovaný proces (UP) znamená:

- řízení požadavky a případy užití.
- řízení rizikem.
- základ na architektuře.
- iterativní a přírůstkový proces vývoje SW produktu.

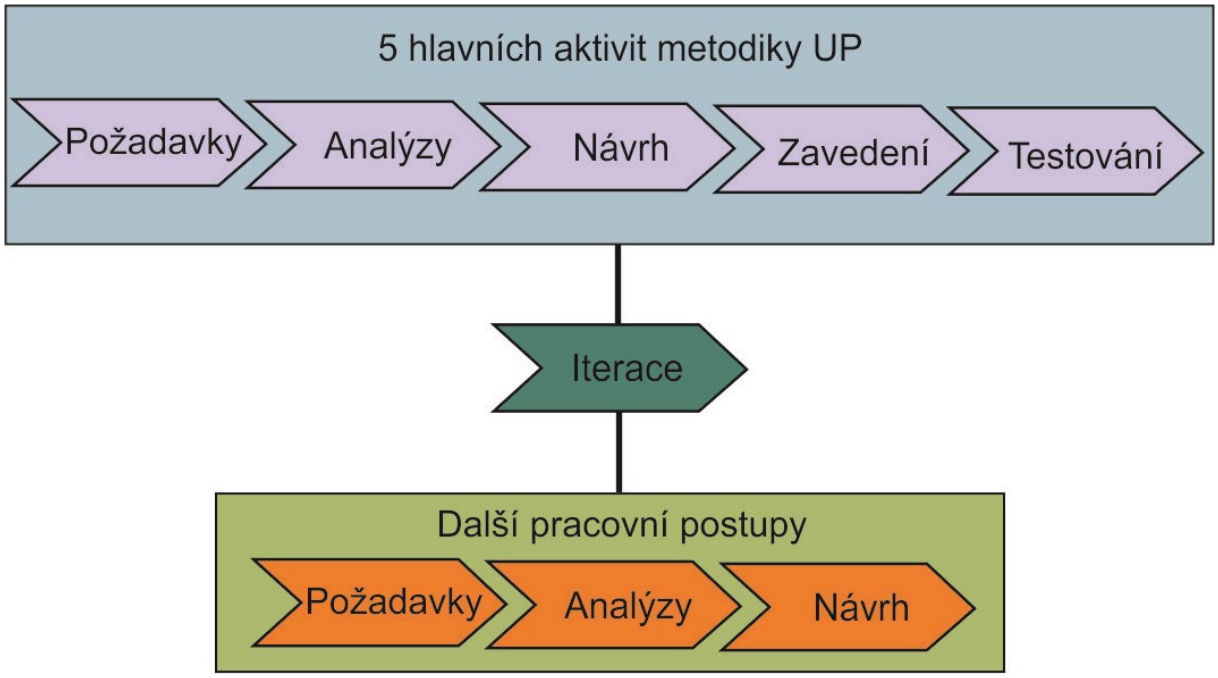
UP je rozdělen do jednotlivých iterací, z nichž každá prochází pěti základními pracovními procesy:

- stanovení požadavků
- analýza
- návrh
- implementace
- testování
- iterace v UP

Iterace je klíčovým prvkem UP. Každou iteraci v UP můžeme chápat jako samostatný dílčí projekt, který má své etapy (jako každý projekt). Etapy iterace v UP jsou tvořeny následujícími činnostmi (aktivitami):

- plánování
- analýza a návrh
- implementace
- integrace a testování
- interní nebo externí uvedení

Jednotlivé iterace mohou probíhat i paralelně. To dovoluje souběžnost a flexibilitu prací u velkých projektů.



Obrázek 4-1 Hlavní aktivity iterací v UP

Iterace a přírůstky

Každá iterace v UP vytváří tzv. **baseline**, tj. základní linii. Základní linie představuje souhrn schválených a revidovaných prvků.

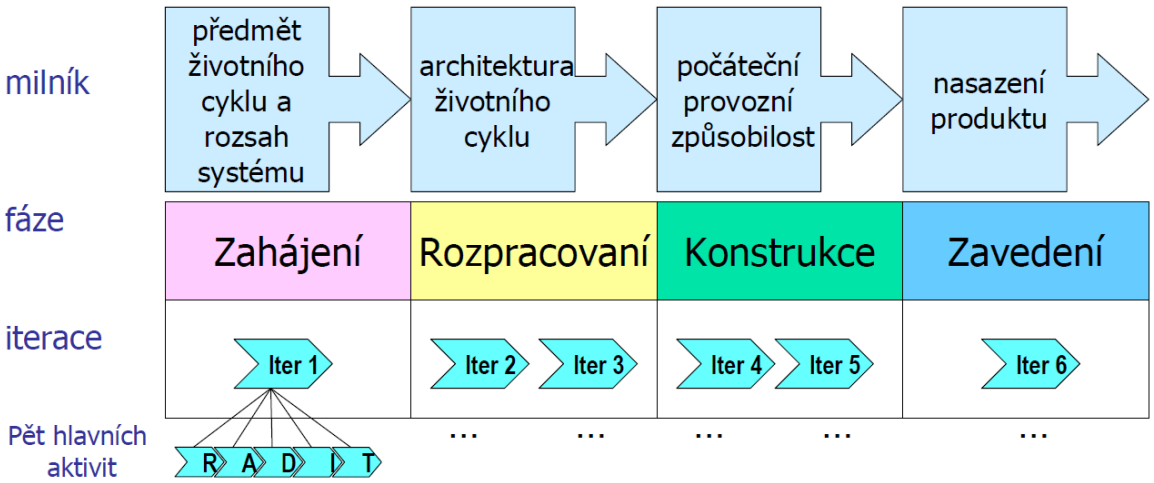
Skýtá určitý základ pro následné přezkoumání a vývoj.

Iteraci je možné měnit pouze prostřednictvím formálních metod správy změn a konfigurace.

A co jsou přírůstky?

Přírůstky jsou rozdílem mezi dvěma následnými základními liniemi. Proto je „metodika iterací a přírůstků“ častým synonymem pro UP.

4.2 Struktura UP



Obrázek 4-2 Struktura UP. Zdroj: http://fei.mtrakal.cz/materialy_public/7.semestr/%5B2010-2011%5DINPSW_Simerda/prednasky/02_UPIntroduction.pdf

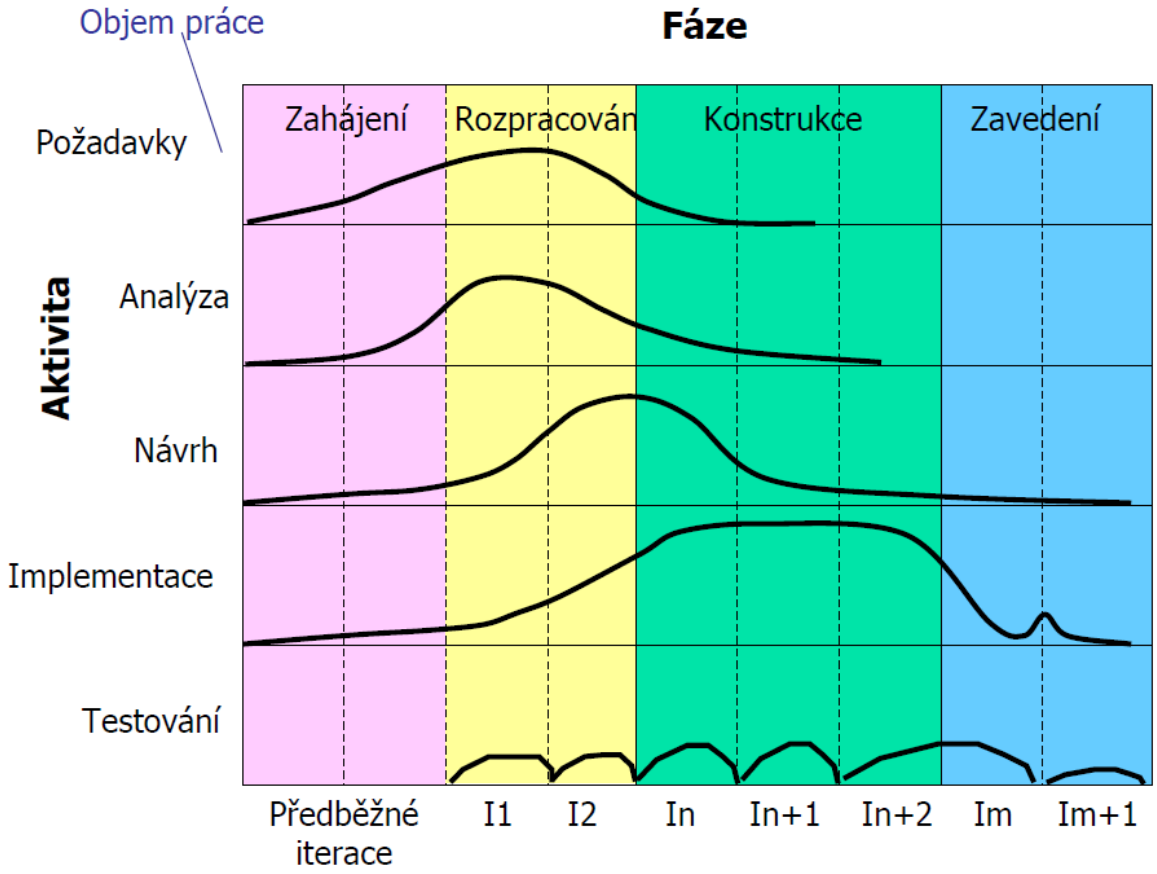
Každá etapa (fáze) může být tvořena jednou či více iteracemi. Kolik takovýchto iterací bude, se vždy odvíjí od velikosti daného projektu. Každá etapa je ukončena tzv. milníkem (*milestone*).

U každé z etap je nezbytné zvážit:

- soustředění se na základ pracovního postupu
- cíl(-e) etapy
- konečný milník etapy

Jak je zřejmé z obrázků 4-2 a4-3, každá aktivita v UP má několik fází:

- zahájení
- rozpracování
- realizace (konstrukce)
- zavedení



Obrázek 4-3 Etapy (fáze) metodiky UP. Zdroj: http://fei.mtrakal.cz/materialy_public/7.semestr/%5B2010-2011%5DINPSW_Simerda/prednasky/02_UPIntroduction.pdf

Podívejme se nyní na tyto fáze podrobněji.

Činnosti fáze zahájení:

Požadavky – zvážení důvodů projektu, jeho přínosů a nejdůležitějších požadavků.

Analýza – při této činnosti je analyzována a stanovena proveditelnost.

Návrh – navrhují se určité technické prototypy.

Implementace – určují a vytváří se koncepce technických prototypů.

Testování – ve fázi Zahájení se testování neprovádí.

Cíle fáze zahájení:

- stanovení proveditelnosti projektu, vytvoření koncepcí a technických prototypů
- tvorba obchodních případů
- určení klíčových požadavků a přínosu systému

- určení kritických rizik

Milník fáze zahájení:

- definovány klíčové požadavky, které musí být odsouhlaseny investorem
- definovány systémové vlastnosti
- tvorba spustitelného architektonického základu
- odhad rizik
- obchodní případy
- investor souhlasí s cílem projektu

Činnosti fáze rozpracování:

Požadavky – upřesnění rozsahu systému a požadavků, které na něj jsou a budou kladeny.

Analýza – při této činnosti se stanovuje, co budeme vytvářet.

Návrh – vytvoření stabilní architektury.

Implementace – vytvoření spustitelného architektonického základu.

Testování – testování spustitelného architektonického základu.

Cíle fáze rozpracování:

- vytvoření spustitelného architektonického základu
- další upřesnění odhadu rizika a definice požadavků a vlastností kvality
- určení klíčových požadavků pro 80% funkčních požadavků
- vytvoření přesného plánu konstrukční fáze
- formulace nabídky zahrnující všechny zdroje – prostředky, čas, vybavení, personál a náklady

Milník fáze rozpracování:

- vytvoření robustního spustitelného architektonického základu
- je evidován odhad rizik
- byl vytvořen plán projektu do takové hloubky, aby umožnil vytvoření nabídky

- obchodní případ byl porovnán s plánem
- uživatelé odsouhlasili pokračování

4.2.1 Stručné shrnutí k metodice UP

UP zahrnuje řízení rizik a případů užití, soustředění se na architekturu, iterace a přírůstky, vytvoření softwarového produktu.

UP má 4 etapy (fáze):

- zahájení
- rozpracování
- realizace
- zavedení

Každá z iterací má 5 procesů:

- požadavky
- analýza
- návrh
- implementace
- testování

4.3 Struktura jazyka UML

Jazyk UML (Unified Modelling Language) je nástrojem sloužícím ke grafickému modelování - dokumentaci, vizualizaci a navrhování softwarových systémů. UML podporuje objektový přístup k návrhu, popisu a analýze. Umožňuje modelovat podnikové procesy a systémové funkce, příkazy programovacího jazyka, opětovně použitelné programové komponenty a schémata databází.

V UML zavádíme tzv. metamodel – definuje strukturu, kterou musí všechny UML modely mít. Zajímavostí je, že samotný jazyk UML byl pomocí takového metamodelu navrhnut.

Jazyk UML je tvořen následujícími hlavními částmi:

- stavební bloky – základní prvky modelu, diagramy, vazby (relace)

- společné mechanizmy – obecné způsoby, pomocí nichž je v jazyku UML možno dosáhnout specifických cílů
- architektura – vizualizace architektury navrhovaného systému

Obecně lze říci, že na UML můžeme pohlížet jako na kolekci nebo sadu spolupracujících objektů. To, s jakými objekty či pohledy pracujeme, závisí na požadované úrovni abstrakce.

UML má také čtyři hlavní mechanismy – specifikaci, ozdoby, podskupiny a mechanismy rozšiřitelnosti:

- Specifikace – každý modelovaný prvek by měl mít textovou specifikaci popisující sémantiku tohoto prvku, jeho smysl a pravidla.
- Ozdoby – doplňující informace, které jsou o modelovaném prvku známé. V některých případech postačí vyjádření prvku jednoduchým geometrickým tvarem, avšak v jiných případech je žádoucí doplnit informace o další podrobnosti – ozdoby. Ve většině případů je na počátku modelování informací méně (a tedy i ozdob) a získáváním dalších informací doplňujeme i ozdoby. Použití ozdob závisí i na úrovni abstrakce (např. u některých diagramů vynecháváme nepodstatné podrobnosti (ozdoby), jinde je uvádíme).

    1) podskupiny – určují možný způsob rozdělení jednotlivých prvků. Nejčastěji vytváříme dvě třídy podskupin:

    2) klasifikátory a instance

    3) rozhraní a implementace

- mechanismy rozšiřitelnosti – jazyk UML již obsahuje možnosti rozšiřitelnosti používané při přizpůsobování modelování aktuálním potřebám. Mezi tyto mechanismy patří:

    1) omezení – uvádí se ve složených závorkách {}. Jedná se o pravidla či podmínky, které je nutno vždy splnit.

    2) stereotypy – pomocí stereotypů je možné ze stávajícího prvku vytvořit prvek jiný. Název stereotypu se uvádí do ostrých závorek <<novy_stereotyp>>. Stereotypu je možné přiřadit i symbol, což je

využíváno například v diagramu nasazení pro tvorbu symbolů zařízení (serverů, tiskáren, počítačů, notebooků atd.)

3) označené hodnoty – tento mechanismus dovoluje přidávat k prvkům modelu nové vlastnosti. Uvádí se ve složených závorkách ve stylu název a hodnota, např. {barva=červená, verze=1.03}.

Podívejme se nyní na hlavní části UML podrobněji.

4.3.1 Stavební bloky

Stavební bloky jazyka UML jsou tvořeny třemi typy stavebních bloků:

- **předměty** (things) – samotné prvky modelu
- **vazby, vztahy** (relationships) – vazby, které prvky modelu spojují. Určují, jak spolu dva anebo více prvků modelu významově souvisí.
- **diagramy** (diagrams) – jedná se o pohledy na modely UML. Diagramy zobrazují kolekce prvků modelu a vizuálně zobrazují, co bude systém dělat a jak (jakým způsobem) to bude dělat. Na otázku „co?“ nám odpovídají analytické diagramy a na otázku „jak?“ jsou odpovědí návrhové diagramy.

Předměty

Předměty, jinak nazývané také jako „věci“ nebo „abstrakce“ jsou v UML vyjadřovány podstatnými jmény. I předměty v UML můžeme dále rozdělit na několik následujících kategorií:

- strukturní abstrakce (structural things) – např. třídy, interfejsy, spolupráce, aktivní třída, uzel, komponenta, případ užití, atd.
- chování (behavioural things) – slouží jako slovesa jazyka UML, např. interakce, stav, apod.
- seskupení (grouping things) – balíčky, do nichž jsou seskupovány významově související prvky modelu do soudržných jednotek.
- poznámky (annotational things) – anotace, které je možné k modelu připojit takovým způsobem, aby vynikala. Ekvivalentem z reálného světa je např. podtržení nebo zvýraznění žlutou tužkou, atd.

Důležitými pojmy jazyka UML jsou pojmy klasifikátor a instance, které je nutno striktně rozlišovat! Klasifikátor je v UML označení pro třídu, kdežto v případě instance jde o označení objektu neboli instance třídy.

Relace

Určují nebo zobrazují vazby mezi předměty v modelu. Vztahují se na strukturní abstrakce a seskupování.

Relace se řídí přesnou sémantikou typů relací, kterou je nutné ovládat a dodržovat.

Následující tabulka uvádí stručný přehled relací a jejich grafického znázornění.

Diagramy

Diagramy jsou grafickou pomůckou pro vizualizaci modelu. Jsou pouze prostředkem, jakým se zobrazují jednotlivé části modelu. Zde je nutné upozornit na častou záměnu nebo domnělou identitu diagramu a modelu. Model není diagram a diagram není model! Diagram je pouze pohled na model. Z diagramu je možné odstranit např. některé vazby nebo předměty, které však v modelu mohou stále existovat. Proto, využíváme-li diagramy k vizualizaci modelu, věnujeme pozornost tomu, aby diagram skutečně reflektoval skutečný stav a podobu modelu. Pokud provedeme úpravy v modelu, měli bychom tyto modifikace přenést rovněž do diagramu a opačně.

V UML se používají následující druhy diagramů:

strukturní diagramy:

- diagram tříd
- diagram komponent
- diagram složených struktur
- diagram nasazení
- diagram balíčků
- diagram objektů, též se nazývá diagram instancí
- diagram profilů

diagramy chování:

- diagram aktivit

- diagram užití
- stavový diagram

diagramy interakce:

- sekvenční diagram
- diagram komunikace
- diagram interakcí
- diagram časování

4.3.2 Diagram tříd

Diagram tříd je základním prvkem objektově orientovaného modelování. Je možné jej použít pro obecné koncepční modelování, detailní modelování, pro převod modelů do programového kódu anebo pro modelování dat.

Diagram tříd znázorňuje souvislosti mezi objekty, operace u objektů a datové struktury od konceptuální úrovně až po úroveň implementace. Pokud jde o datové struktury – ty jsou v diagramu tříd zařazeny do tříd, u nichž jsou rovněž zobrazeny jejich vztahy.

Mimo datové struktury obsahuje diagram tříd rovněž třídy systému s jejich atributy, operacemi a metodami.

Jak z názvu diagramu vyplývá, jeho základním prvkem nebo objektem jsou třídy. Grafické znázornění tříd je v diagramu tříd rozděleno do třech částí:

Horní část – obsahuje název třídy (tučným písmem). Podle určitých konvencí, které je na místě dodržovat, začíná název třídy velkým písmenem. V případě víceslovného názvu třídy je vhodné použít tzv. velbloudí styl (CamelCase) – mezi slovy vynecháme mezery a počáteční písmeno každého slova začneme kapitálkou, např.: VíceslovnýNázevTřídy.

Střední část – obsahuje atributy třídy, psané s malým počátečním písmenem. V případě víceslovných názvů atributů používáme rovněž CamelCase, ovšem s malým počátečním písmenem, např.: názevVíceslovnéhoAtributu. Z objektově orientovaných přístupů víme, že atributy definují vlastnosti třídy. Např. pro třídu Uživatel bychom mohli definovat atributy jako jméno, příjmení, telefon, role, apod.

Spodní část – obsahuje operace a metody, které daná třída provádí. Typografická konvence je shodná s atributy – názvy operací a metod se tedy píší s malým počátečním písmenem a stylem CamelCase. Za názvem metod a operací se píší závorky, aby bylo ihned zřejmé, že se jedná o „funkce“ třídy.

Důležitými symboly využívanými u diagramu tříd jsou rovněž znaky, označující viditelnost atributů a metod. Mezi používané symboly patří následující:

\+ označuje viditelnost typu *public*

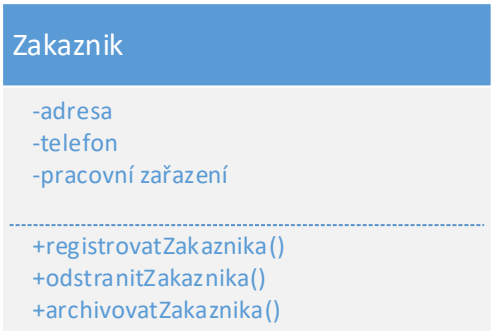
\- označuje viditelnost typu *private*

\# označuje viditelnost typu *protected*

/ označuje viditelnost typu *derived*

~ označuje viditelnost typu *package*

Dle výše uvedeného, pohled na třídu v diagramu tříd je znázorněn např. takto:



Obrázek 4-4 Diagram tříd

Vztahy mezi třídami se prostřednictvím diagramů tříd znázorňují pomocí symbolů pro relace (viz předcházející podkapitola 4.3.1). Např. vztah mezi třídou A a B je možné znázornit následovně:

Obrázek 4-5 Znázornění relace mezi třídami

Číslo u symbolu relace udává tzv. mohutnost (multiplicity). Mohutnost určuje, kolik instancí dané třídy může být svázáno s instancí druhé třídy. K pochopení uvedeného nám pomůže následující tabulka a výše uvedený příklad. Podle tabulky je zřejmé, že instance třídy A může být svázána s jednou nebo více instancemi třídy B.

0	**0..1**	**1**	**0..***	**1..***
Žádná instance (zcela výjimečně)	Žádná nebo právě jedna	Právě jedna instance	Žádná nebo více instancí	Jedna nebo více instancí

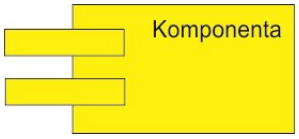
V diagramu tříd je možné znázornit rovněž asociace, agregace, kompozice, dědičnost, realizace, závislost. K bližšímu nastudování této problematiky odkazujeme na doporučenou literaturu a další zdroje.

4.3.3 Diagram komponent

Podle definice komponent v UML se komponentou rozumí část modulárního systému, která zapouzdřuje svůj obsah a jejíž chování či projev lze navenek nahradit. To znamená, že pokud vyjmeme komponentu z funkčního systému a nahradíme ji jinou komponentou, která umožňuje stejné chování a stejné chování požaduje od systému, pak je vše v pořádku. Jako příklad z reálného světa si lze představit např. tranzistor v elektrickém obvodu. Pokud jej vyjmeme a nahradíme jiným typem tranzistoru se shodnými parametry, systém bude nadále funkční. Nemůžeme, ale např. tranzistoru PNP nahradit tranzistorem NPN, byť má podobné pouzdro a rozmístění vývodů.

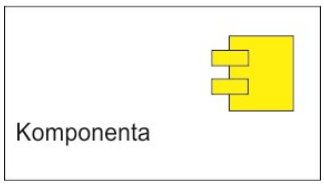
Komponenta (vzhledem k tomu, že se jedná o určitou specializaci třídy) může obsahovat atributy a metody a může se účastnit asociací a generalizací. Může obsahovat vnitřní strukturu a definovat porty.

Pro znázornění komponenty používáme opět standardizovaných grafických prvků. Dle specifikace UML 2.0 je pro notaci komponenty použit symbol obdélníku s ikonou dle starší normy umístěné do pravého horního rohu.



Obrázek 4-6 Notace komponenty dle starší normy

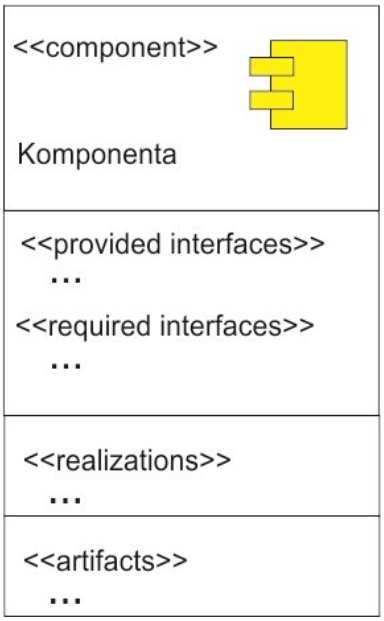
Na obrázku 4-6 je znázorněna ikona komponenty dle starší normy. Uvádíme ji pro úplnost a z toho důvodu, že se s ní stále můžeme setkat u dříve vytvořených modelů.



Obrázek 4-7 Současná notace komponenty

Komponenta může poskytovat nebo vyžadovat interfejs – rozhraní pro komunikaci s okolím.

Podrobná notace komponenty včetně její vnitřní struktury vypadá např. takto:



Obrázek 4-8 Podrobná notace komponenty

Klíčové slovo <<component>> umístěné nad název komponenty je volitelné, avšak velmi často využívané.

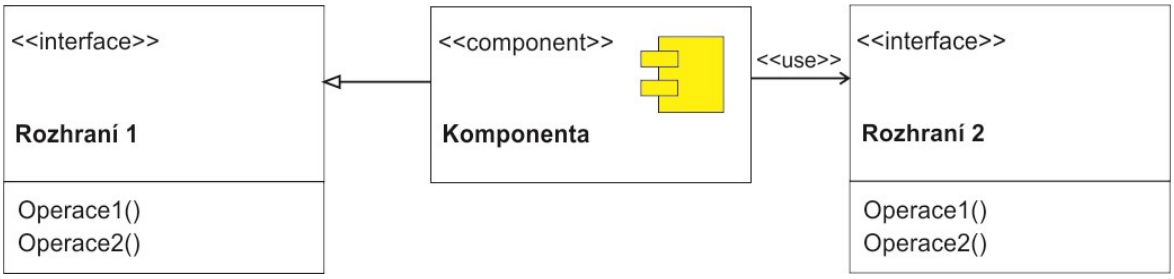
Podobně jako u notace třídy v diagramu tříd, i komponenta může obsahovat grafické znázornění vnitřní struktury. Jednotlivé části struktury se zapisují do oddělených prostor obdélníku znázorňujícího komponentu. Tyto části je možné rozdělit na následující prvky“

- rozhraní – poskytovaná nebo požadovaná. Označují se klíčovými slovy

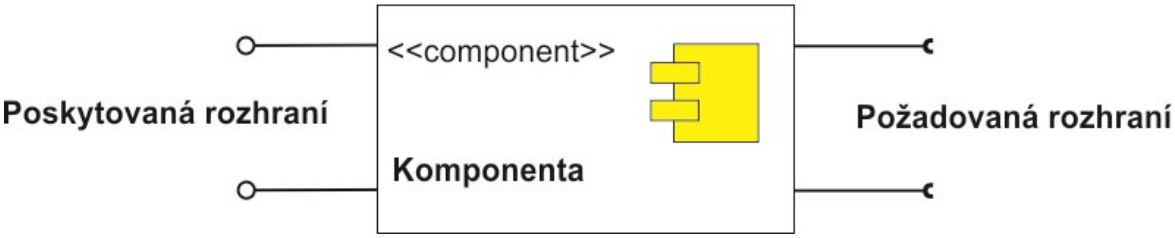
  <<provided interfaces>> a <<required interfaces>>.

- realizace – označuje se notací <<realizations>>
- artefakty – označuje se notací <<artifacts>>

Rozhraní komponenty je možné znázornit dvěma způsoby. Explicitní reprezentací rozhraní (požadovaných a poskytovaných) s uvedením podrobností anebo jednodušším způsobem s použitím symbolů určených pro znázornění rozhraní. Oba uvedené způsoby jsou uvedeny na následujících obrázcích.

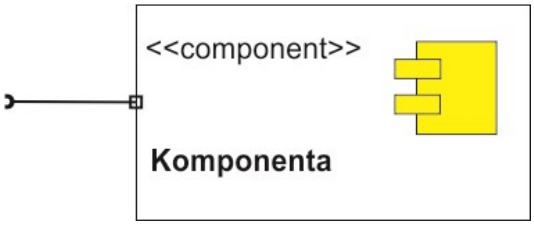


Obrázek 4-9 Rozhraní komponenty s uvedením podrobností

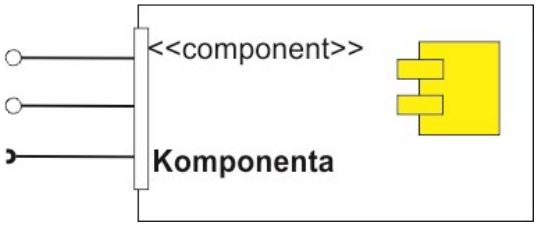


Obrázek 4-10 Rozhraní komponenty s použitím symbolů

V případě, že je nutné komponentu rozdělit na subkomponenty umístěné uvnitř hlavní komponenty. Při tomto pohledu je možné použít znázornění portů a znázornění spojek pro propojení s rozhraními hlavní komponenty. Pro porty a spojky používáme následující grafické znázornění:



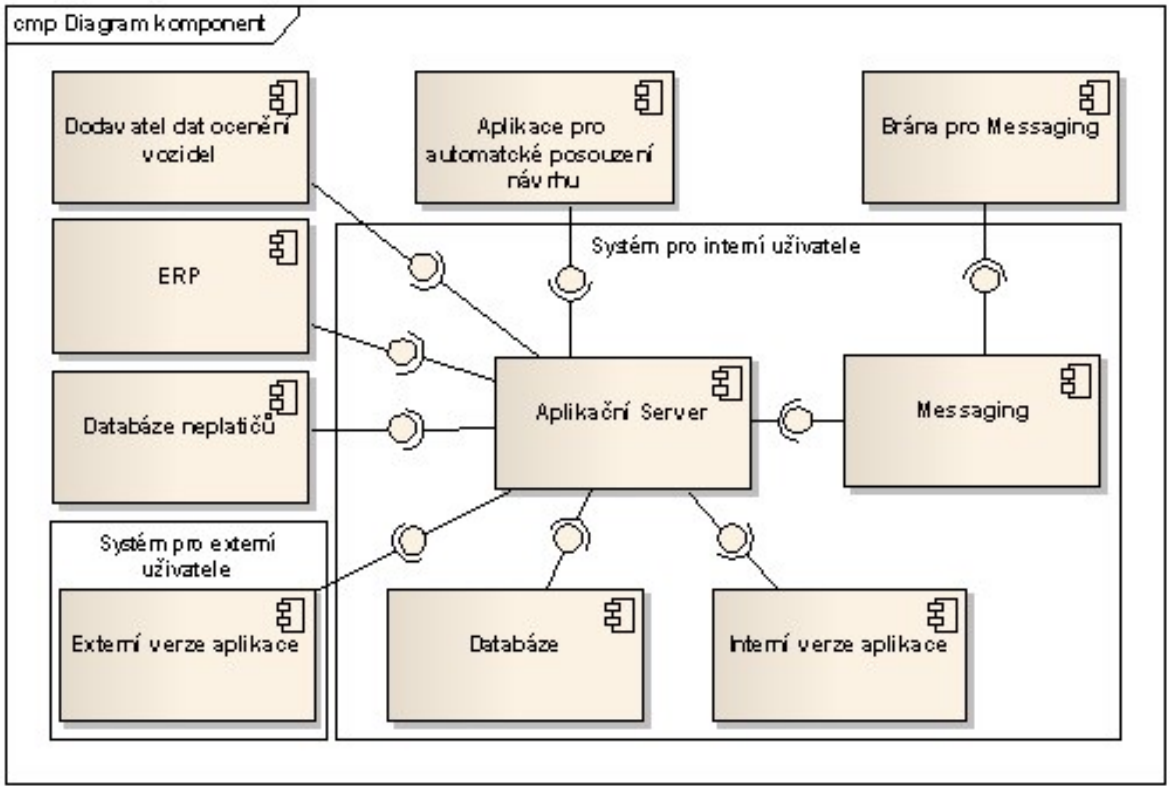
Obrázek 4-11 Komponenta s portem



Obrázek 4-12 Komponenta se složeným (komplexním) portem

Obrázek 4-13 Složená spojka

Příklad diagramu komponent:



Obrázek 4-14 Diagram komponent. Zdroj: http://uml.czweb.org/Diagramy/diagram_komponent.jpg

4.3.4 Diagram složených struktur

Diagramy složených struktur zobrazují vnitřní uspořádání klasifikátorů a znázorňují informace, které se obtížně modelují pomocí jiných typů diagramů. Diagramy složených struktur zavádí až UML 2. V předchozích specifikacích UML se tento diagram neobjevuje.

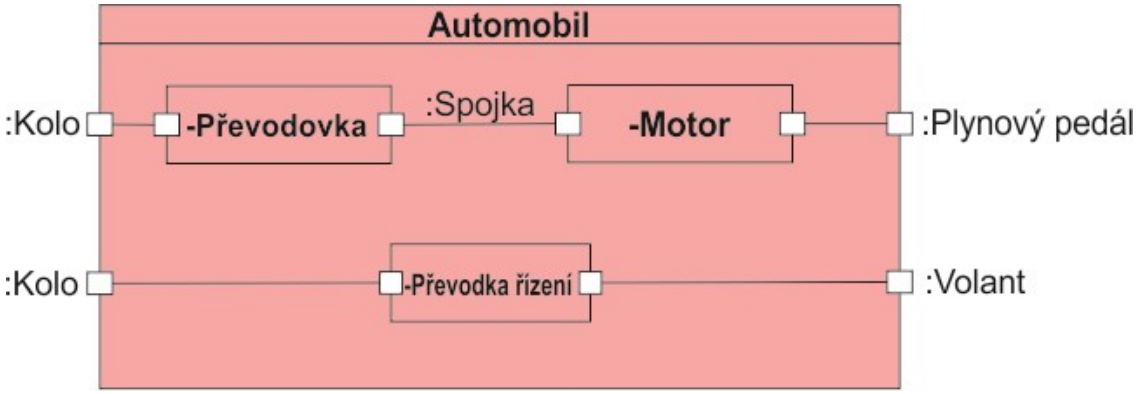
Diagramy složených struktur jsou velice silným prostředkem pro znázornění možností klasifikátoru. Připomeňme si, že pojem klasifikátor v UML znamená označení třídy (nikoli tedy její instance – objektu).

Diagram složených struktur je tedy prostředkem pro znázornění:

- interní struktury klasifikátoru
- interakce klasifikátoru s okolím prostřednictvím portů
- chování při spolupráci

Notace u tohoto typu diagramu je shodná s notací u diagramu tříd.

Na obrázku níže je znázorněn příklad diagramu složených struktur.



Obrázek 4-15 Diagram složených struktur

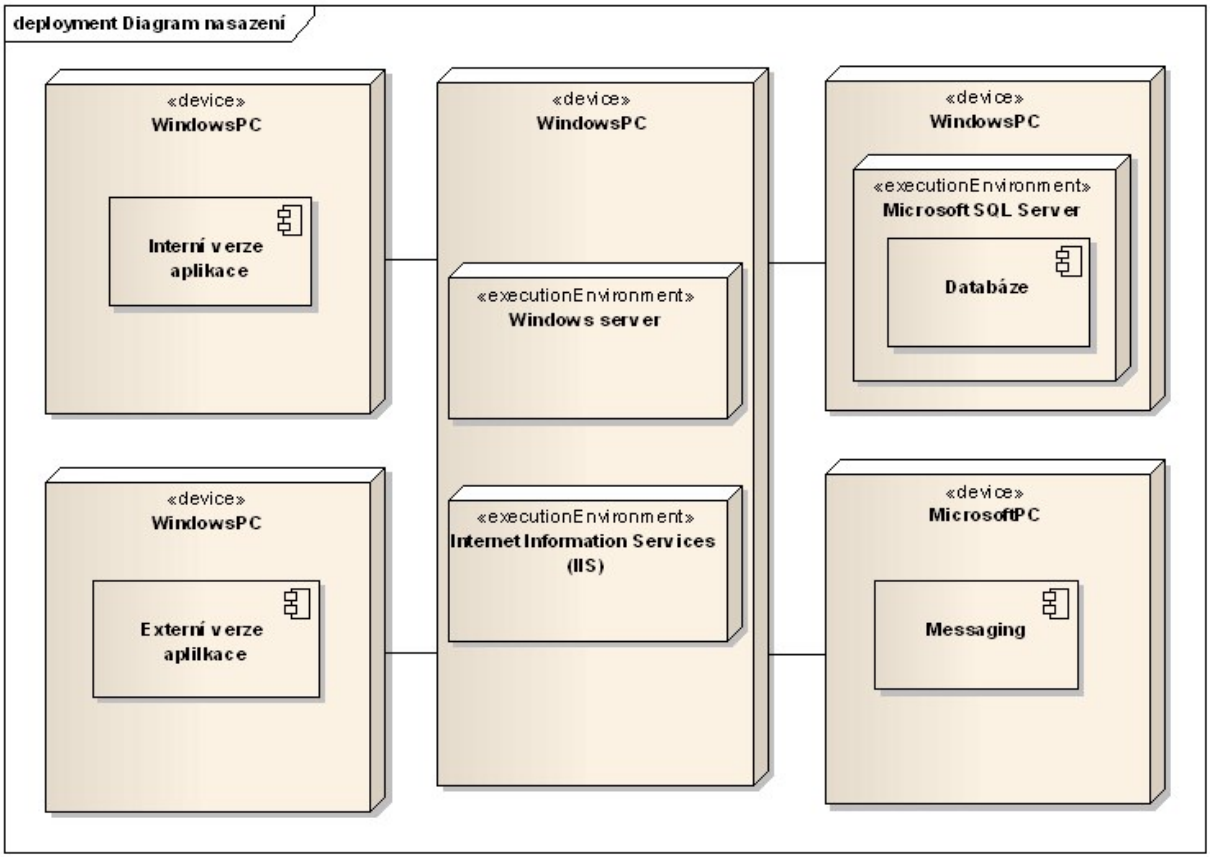
4.3.5 Diagram nasazení

Diagram nasazení znázorňuje rozmístění HW a dalších zdrojů včetně softwarových komponent, objektů a procesů, které je tvoří.

Základním elementem diagramu nasazení jsou uzly (nodes). Uzly jsou vzájemně propojeny komunikačními cestami (communication paths). Uzly mohou v diagramu nasazení zastupovat různé druhy prostředků, např.:

- **fyzická zařízení** – uzel typu <<device>>
- **typ prostředí zpracování softwaru** – např. aplikační server, databázový server, apod. Označuje se jako <<execution environment>>
- **artefakty** – představují fyzické umístění softwaru, většinou soubory, skripty, dokumenty apod. Artefakty často zastupují více komponent (není podmínkou).

Příklad diagramu nasazení:



Obrázek 4-16 Diagram nasazení, Zdroj: http://uml.czweb.org/Diagramy/diagram%20nasazeni.jpg

4.3.6 Diagram balíčků

Jak název napovídá, diagram balíčků je tvořen znázorněním elementů modelů UML zařazených do skupin (typicky např. související třídy, apod.) a případným znázorněním závislostmi mezi těmito skupinami.

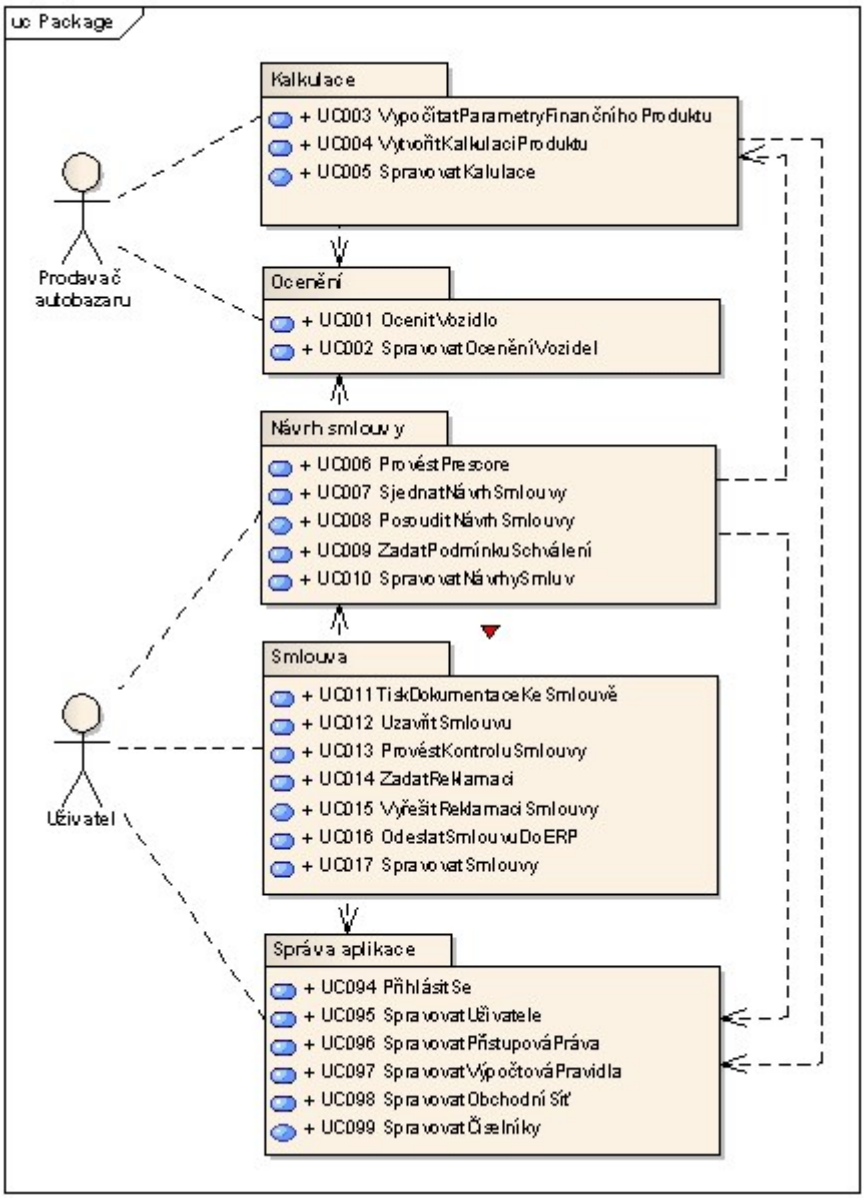
Balíčky mohou být součástmi jiných balíčků a je možné tak vytvořit hierarchickou strukturu se znázorněním závislostí mezi jednotlivými balíčky.

Třídy v jednom balíčku musí mít v rámci tohoto balíčku jednoznačný název.

Při tvorbě diagramu balíčků je vhodné dodržovat následující doporučení:

- názvy balíčků by měly být jednoduché a popisné
- v ostatních diagramech by měly být balíčky použity pouze tehdy, kdy je zpřehlední nebo zjednoduší
- v diagramu balíčků by neměly existovat žádné závislosti ve smyčce
- diagram balíčků by měl vycházet z diagramu komponent znázorňujícího fyzickou strukturu modelovaného programu. Pomocí diagramu balíčku pak znázorníme logickou strukturu.
- do jednoho balíčku by měly být zahrnuty třídy, které jsou na shodné hierarchické úrovni
- balíčky jsou na sobě závislé, pokud jsou na sobě jakkoli závislé dva elementy v daných balíčcích.

Příklad diagramu balíčků použitého v diagramu užití:



Obrázek 4-17 Diagram balíčků. Zdroj: http://uml.czweb.org/Diagramy/diagram_balicku_UC.jpg

4.3.7 Diagram objektů

Poznámka: Namísto označení diagram objektů se lze setkat rovněž s názvem diagram instancí.

Diagram objektů zobrazuje instance tříd a vztahy mezi těmito instancemi. Jde vlastně o diagram velmi podobný diagramu tříd nebo diagramu komunikace (s vyloučením zpráv). Jak lze předpokládat, diagramy objektů s diagramy tříd úzce souvisí. Stejně, jako je objekt instancí dané třídy, je možné diagram objektů považovat za instancí diagramu tříd. Objekty v diagramu objektů jsou instancemi tříd z diagramu tříd a vazby mezi těmito objekty jsou dány vazbami mezi těmito třídami.

Diagramy objektů znázorňují statickou strukturu systému v určitém čase. Využívají se k určování a testování správnosti a přesnosti diagramů tříd, pro znázornění podmínek vzniku událostí a pro pochopení složitých vztahů mezi třídami.

Pokud jde o symboliku, je podobná symbolice diagramu tříd s tím rozdílem, že se do hlavní části obdélníku znázorňujícího objekt uvádí obvykle název objektu následovaný dvojtečkou a názvem třídy, jejích je objekt instancí. Takovéto znázornění je nazýváno **pojmenovaný objekt**. Doplňujícím zobrazením může být o objekt bez názvu – v tom případě se uvádí pouze dvojtečka následovaná názvem třídy, jejíž je daný objekt instancí. Toto znázornění objektu se pak nazývá **nepojmenovaný objekt**.

V případě, že není uvedena třída objektu, dvojtečka se neuvádí.

Obrázek 4-18 Nepojmenovaný objekt

Obrázek 4-19 Pojmenovaný objekt

Obrázek 4-20 Pojmenovaný objekt s cestou objektu

Stejně jako u diagramu tříd, rovněž u diagramu objektů je možné uvést seznam atributů.

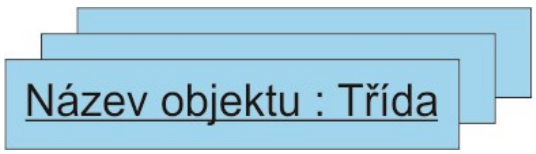
V tomto případě je však nutno uvést i jejich hodnoty.

Obrázek 4-21 Seznam atributů s povinným uvedením hodnot

Vzhledem k tomu, že diagram objektů ukazuje stav v daném časovém okamžiku, může být výhodné odlišit objekt, který je právě aktivní. Toto odlišení se provádí zvýrazněním ohraničení symbolu objektu. Objekt je v tomto případě nazýván **aktivní objekt**. Aktivní objekt řídí tok akcí.

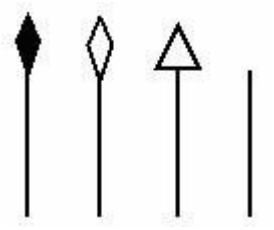
Obrázek 4-22 Aktivní objekt

Dalším potřebným typem zobrazení v diagramu objektů jsou vícenásobné instance (dané třídy). Značíme takto:

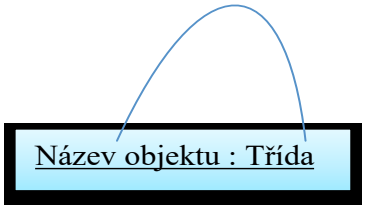


Obrázek 4-23 Vícenásobné instance třídy

Vazby v diagramu objektů jsou nazývány jako **spojení**. Reprezentují instance vztahů (vazeb, relací). U spojení (nad jeho symbolem) se uvádí jeho název. Spojení je dynamické, tzn. že nemusí trvat po celou dobu životního cyklu objektu – v čase se může měnit. Podle konvence UML je název spojení podtržený.

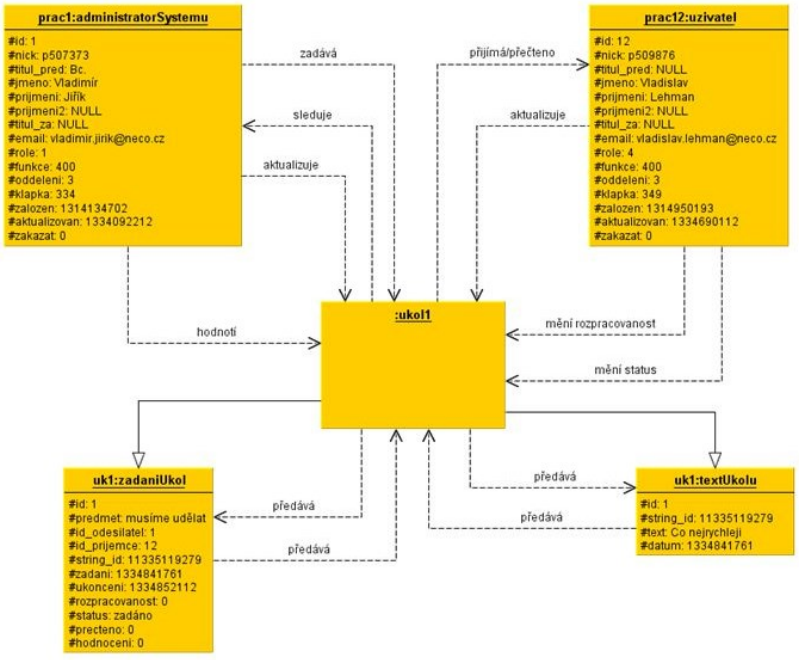


Speciálním případem spojení je odkazování objektu sama na sebe. Tento druh spojení se užívá v případě, že objekt plní vícero rolí.



Obrázek 4-24 Odkazování objektu sama na sebe

V případě dynamické změny (změny role objektu a s tím související změny spojení) je pro tuto situace nezbytné vytvořit nový diagram, zachycující novou, aktuální situaci.



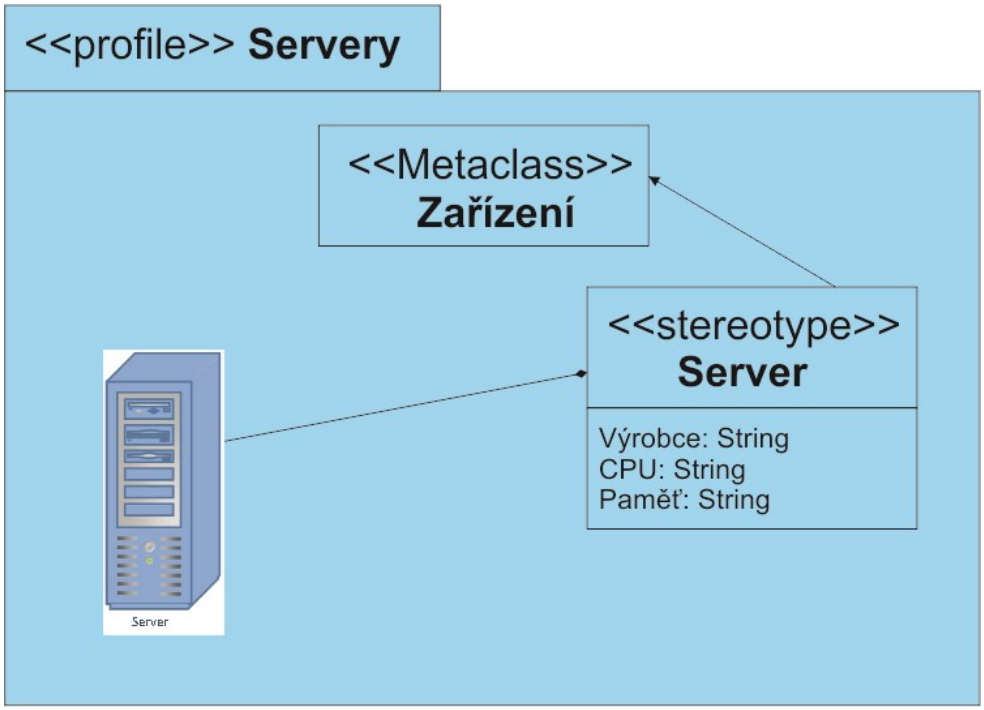
Obrázek 4-25 Příklad diagramu objektů. Zdroj: https://www.google.cz/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&ved=0ahUKEwjkwuGvj63YAhUM66QKHTxNBTMQjRwIBw&url=http%3A%2F%2Fslideplayer.cz%2Fslide%2F2514668%2F&psig=AOvVaw2uycYnc7xVGKHDsgeYyUVp&ust=1514564706458701

4.3.8 Diagram profilu

Diagram profilu slouží ke grafickému znázornění rozšíření pomocí stereotypů. Pracuje na úrovni metamodelu.

V UML 1 se tento typ diagramu nevyužíval, zavádí jej až UML 2.

Příklad diagramu profilů:

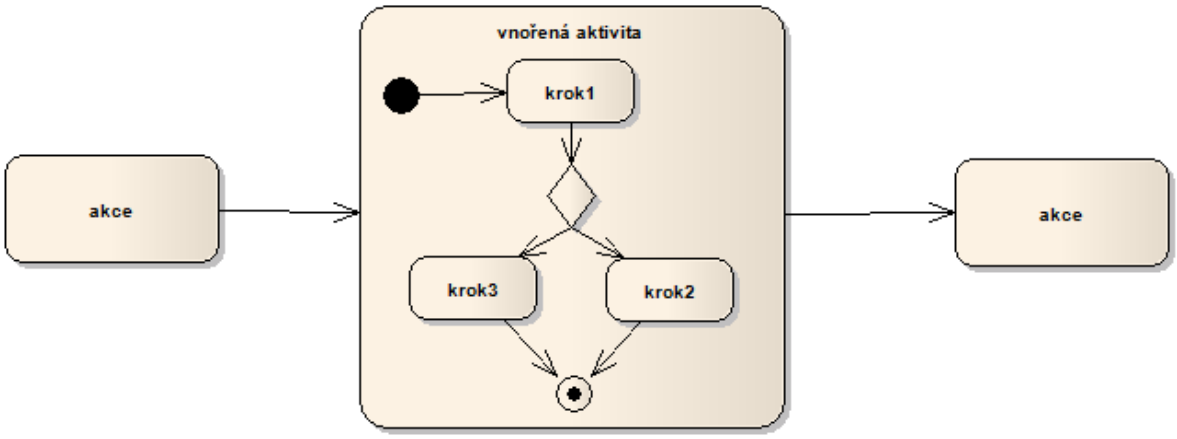


Obrázek 4-26 Diagram profilů

4.3.9 Diagram aktivit

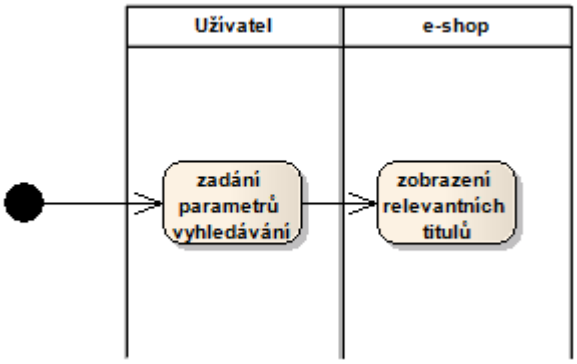
Diagram aktivit patří mezi diagramy chování. Jeho účelem je zejména modelování procesů a procedurální logiky. Procesy jsou v diagramu aktivit znázorňovány jako posloupnost kroků, jež můžeme v diagramu znázornit jako akce či vnořené aktivity, přičemž

- akce jsou kroky, které již dále nedělíme (tzv. atomické kroky). Je to činnost, která se aktivně vykonává uvnitř aktivity. Může se jednat i o vnořenou aktivitu.



Obrázek 4-27 Vnořená aktivita. Zdroj: https://cs.wikipedia.org/wiki/Diagram_aktivit#/media/File:AKCE_aktvita_axample.png

Akce může být prováděna buďto člověkem s přidělenou rolí anebo systémem. Role a systémy jsou v diagramu aktivit znázorněny jako pole rozdělená čarami. Uvádí se i název aktivity. Každé takové pole obsahuje akce prováděné člověkem nebo systémem.



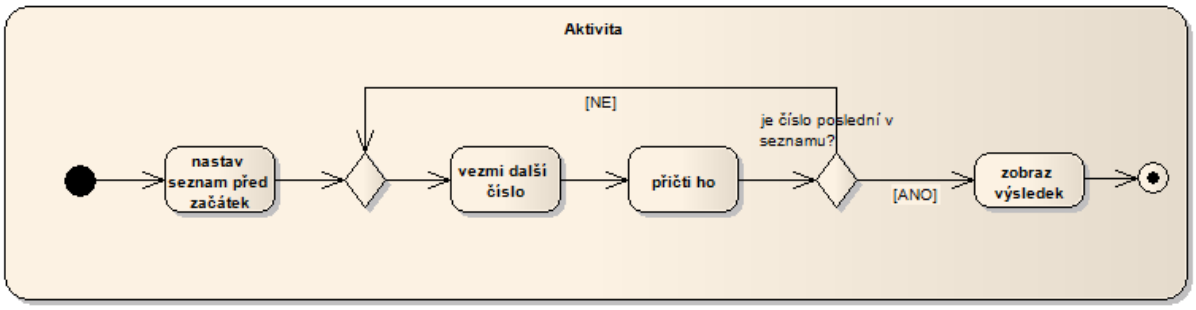
Obrázek 4-28 Znázornění oblastí. Zdroj: https://cs.wikipedia.org/wiki/Diagram_aktivit#/media/File:Swinline.png

- vnořené aktivity reprezentují volání dalších procesů (tzn. aktivit). Tyto další aktivity je možné znázornit pomocí dalšího diagramu aktivit.

Pořadí jednotlivých kroků v tomto diagramu je určeno řídícím tokem.

Na vstupu aktivity je možné předávat aktivitě data nebo objekty ve formě parametrů. Aktivita pak může objekty zase předávat na svém výstupu. Spuštění aktivity je inicializováno po naplnění všech parametrů a přivedení všech řídících toků.

Modelovány jsou pouze aktivní akce.



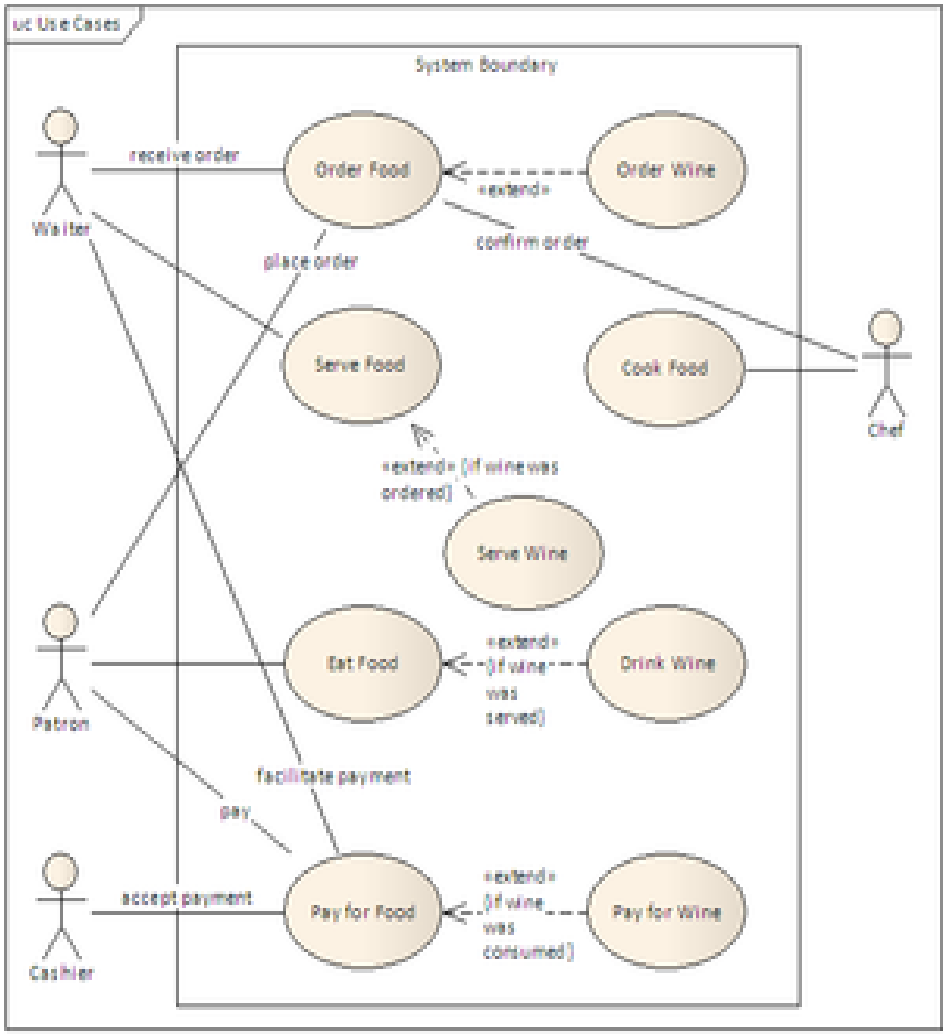
Obrázek 4-29 Modelování aktivních akcí. Zdroj: https://cs.wikipedia.org/wiki/Diagram_aktivit#/media/File:Aktivita_example.png

4.3.10 Diagram užití

Patří mezi další diagramy znázorňující chování modelu. Diagram užití znázorňuje pohled na modelovaný systém zvenčí. Umožňuje tak pohled na hranice systému. Diagram užití zobrazuje posloupnosti transakcí mezi uživatelem s přidělenou rolí (případně jiným systémem) a systémem, které spolu nějakým způsobem souvisí. Diagram případů užití využívá následující prvky:

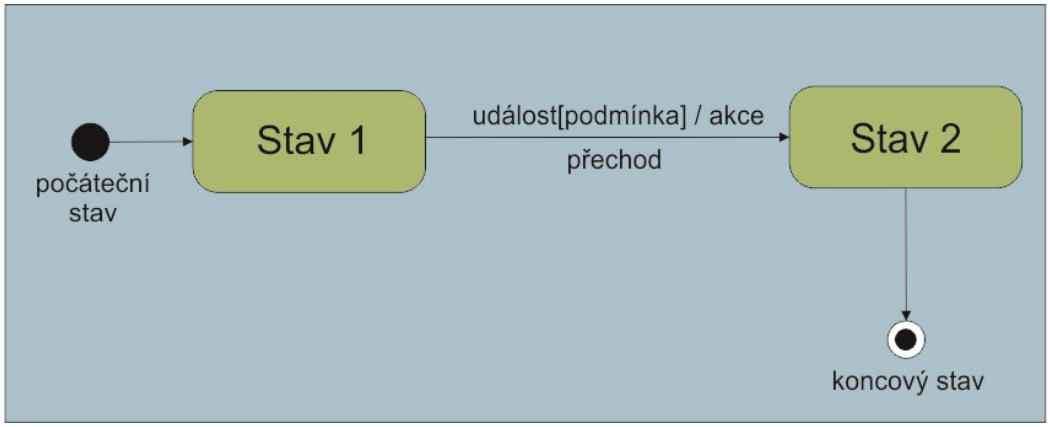
- případ užití – jedná se o sekvenci akcí mající nějaký vztah s aktéry. Případ užití se značí jako ovál a může zobrazovat i následující vazby:
    - Include – případ užití zahrnuje nebo obsahuje jiný případ užití, Typickým příkladem může být např. nabídka Soubor -> Otevřít, apod.
    - Extend – případ užití slouží k rozšíření jiného případu užití. Typickým příkladem je např. Uložit -> Uložit jako, apod.
    - Generalization – případ užití je určitým případem jiného případu užití.
- Účastník (aktér) – popisuje vnější objekty, které vstupují do vztahu s procesy. Může obsahovat následující relace:
    - Generalization

  Aktér je značen symbolem postavy (figury).



Obrázek 4-30 Diagram užití. Zdroj: https://en.wikipedia.org/wiki/Use_case_diagram#/media/File:Use_case_restaurant_model.svg

4.3.11 Stavový diagram



Obrázek 4-31 Stavový diagram

Stavový diagram je další z řady diagramů chování. Jak název napovídá, jedná se o znázornění stavů vývoje systému s konečným počtem stavů. Jinými slovy se dá říci, že se může jednat o konečný stavový automat nebo podobné systémy, které umožňují znázornění stavů objektu a přechody mezi těmito stavy (přechodová funkce).

Během průchodu přes určitý stav mohou být vykonávány různé činnosti (aktivity). Podobně jako tomu je u stavových automatů, u stavového diagramu jsou definovány tři základní stavební prvky – přechod, stav a událost.

Stavový diagram se vytváří pro prvky, u kterých má toto znázornění (pomocí stavového automatu) nějaký smysl, tzn.:

- prvek může reagovat na vnější události.
- existenci prvku je možné vyjádřit pomocí množin stavů, událostí a přechodů mezi jeho stavy.
- aktuální stav prvku je důsledkem předchozího stavu, neboli prvek se chová způsobem, který je důsledkem předešlého chování.

Podívejme se nyní na základní stavební prvky stavového automatu podrobněji:

stav – určuje trvání neměnných podmínek v daném systému. Stav vyjadřuje momentální chování prvku (objektu), jeho činnost nebo čekání na událost. Je určen následujícími parametry:

- hodnotami atributů objektu
- vazbami, které má objekt s ostatními objekty
- momentálně vykonávanou činností (aktivitou)

Každému stavu je možné přiřadit jakýkoli počet aktivit a akcí, přičemž pod pojmem **aktivita** (činnost) se rozumí déletrvající proces, který je možno přerušit a pod pojmem **akce** rychle probíhající nepřerušitelný proces.

Všechny akce (každá z nich) jsou přidruženy k vnitřnímu přechodu, který je důsledkem událostí. Vnitřní přechod je významný pro skutečnosti v rámci daného stavu, není však určen k přechodu do jiného stavu.

Stav je v diagramu znázorněn obdélníkem s kulatými rohy a názvem nacházejícím se v horní části obdélníka.

přechod – je přímým propojením zdrojového a cílového stavu (v uvedeném směru). Jedná se o reakci objektu na určitou událost (její vznik) anebo na dokončení aktivní činnosti.

Ke každému přechodu je možné (nikoli však nezbytné) přidat popisek ve tvaru událost[podmínka] /akce. Kterákoli z částí popisku může být vypuštěna.

Přechod je tedy definován následujícími vlastnostmi:

- událost – spuštění přechodu. Událost může být interní anebo externí. V případě události dokončení aktivity se jedná o tzv. automatický přechod.
- podmínka – jedná se o logický výraz (booleovský), který podmiňuje přechod.
- akce – činnost vyvolaná při započetí přechodu

událost – podle specifikace UML je událost „něco významného, co nastane v určitém čase a prostoru a co nemá trvání“. Událostí může být celá řada, těmi nejdůležitějšími jsou:

- událost volání – jedná se o požadavek na vyvolání konkrétní činnosti instance kontextové třídy. Syntaxe události volání by tedy měla být shodná se syntaxí dané činnosti. Zaznamenání tohoto volání objektem pak vyvolává danou činnost a vyvolává událost. V rámci události volání je možné určit posloupnost a pořadí akcí. V tomto případě se jednotlivé akce oddělují středníkem.
- událost změny – řadí se do kategorie signálů a jedná se o změnu splnění podmínky z False na True.
- časová událost – patří rovněž do kategorie signálů a dochází k ní po uplynutí určité časové periody nebo po splnění časové podmínky. Příkladem může být např. opakování jednou za minutu, jednou za den, pětkrát do měsíce, atd.

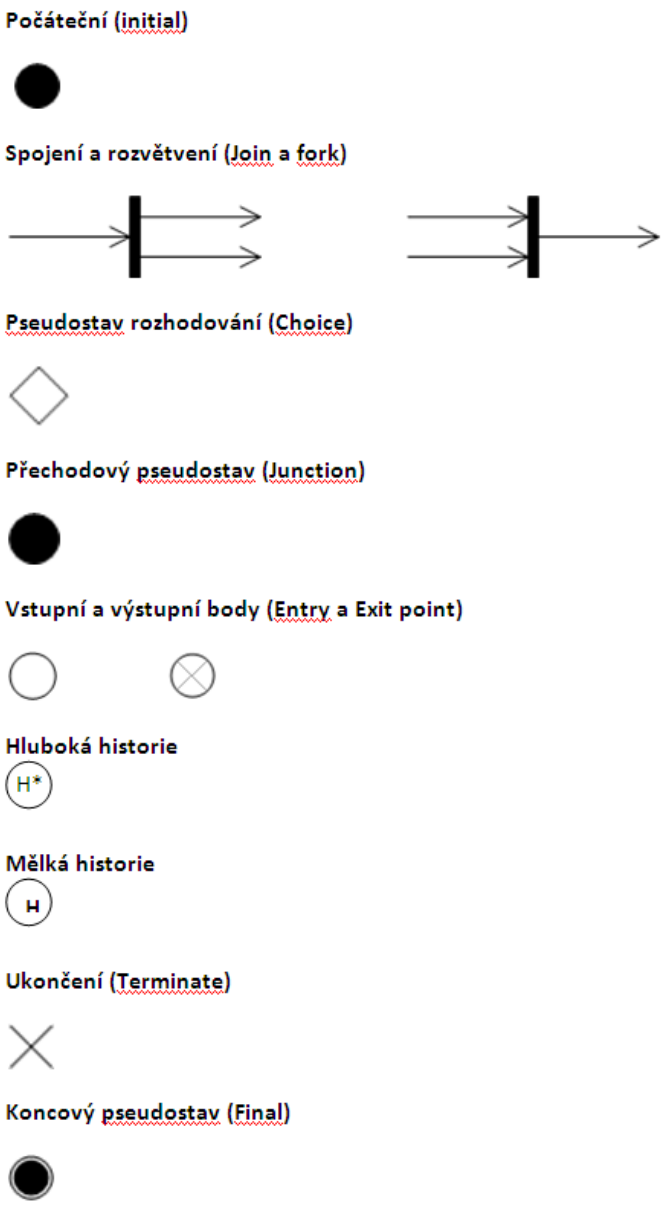
V souvislosti se stavy je nutné se zmínit rovněž o tzv. **pseudostavech**. Pseudostavy nejsou stavy v pravém slova smyslu. Prvek (objekt) v pseudostavech nesetrvává, jen jimi prochází. Ve stavovém diagramu se můžeme setkat s několika typy pseudostavů:

- **počáteční stav** – nejedná se o stav, ale o pseudostav. Je výchozím krokem, ze kterého stavový automat přechází do tzv. defaultního stavu. Pro přechod z počátečního stavu není definována žádná událost, s výjimkou události <<create>>, která definována býti může. U přechodu z počátečního stavu k defaultnímu je možno definovat chování.

- **rozvětvení** – přechody vycházejí z jednoho stavu k několika dalším stavům. U těchto přechodů se nedefinují žádné podmínky ani události. Z pseudostavu rozvětvení vedou minimálně dva přechody.
- **spojení** – je opakem rozvětvení. Rovněž se zde nedefinují události ani podmínky. Do pseudostavu spojení vstupují minimálně dva přechody.
- **rozhodování** – jedná se o větvení přechodů na základě podmínek definovaných u výstupních přechodů. Stavový automat vybere ten přechod, jehož podmínka se vyhodnotí jako splněná. Je tedy nezbytné dbát na determinismus.
- **přechod** (přechodový pseudostav) – využívá se k vytvoření složených přechodů (viz dále) mezi stavy. Je obdobou psedostavu rozvětvení a spojení s tím rozdílem, že u výstupních resp. vstupních přechodů jsou definovány podmínky. U těchto přechodů se definují pouze podmínky, nikoli události.
- **vstupní/výstupní body** – zastupují vstupy a výstupy uskutečňující přechody ze složených stavů nebo do nich.
- **hluboká historie** – je pseudostavem zastupujícím poslední nastavení kompozitního stavu po jeho opuštění, včetně vnořených složených stavů. Z tohoto pseudostavu může být vytvořen pouze jediný přechod k některému ze stavů tvořících složený stav, ve kterém se pseudostav hluboké historie nachází. Jde o výchozí přechod, neboť je takto definován stav, ve kterém se systém ocitne tehdy, pokud v hluboké historii není uložena žádná konfigurace stavů (jde o případ, kdy složený stav nebyl ještě aktivní, anebo přešel do koncového stavu).
- **mělká historie** – zastupuje poslední aktivní podstav složeného stavu. Přechod, který vstupuje do prvku mělké historie reprezentuje přechod do posledního aktivního podstavu složeného stavu. Mělká historie může být součástí pouze jednoho prvku složeného stavu. Přechod, jež vystupuje z mělké historie znázorňuje přechod do výchozího stavu v případě, že složený přechod dosud nebyl aktivován a mělká historie je tedy prázdná. Název „mělká“ symbolizuje pouze povrchní paměť konfigurace, neboť si pamatuje jen poslední aktivní stav (poslední aktivní konfiguraci) v rámci podstavu.

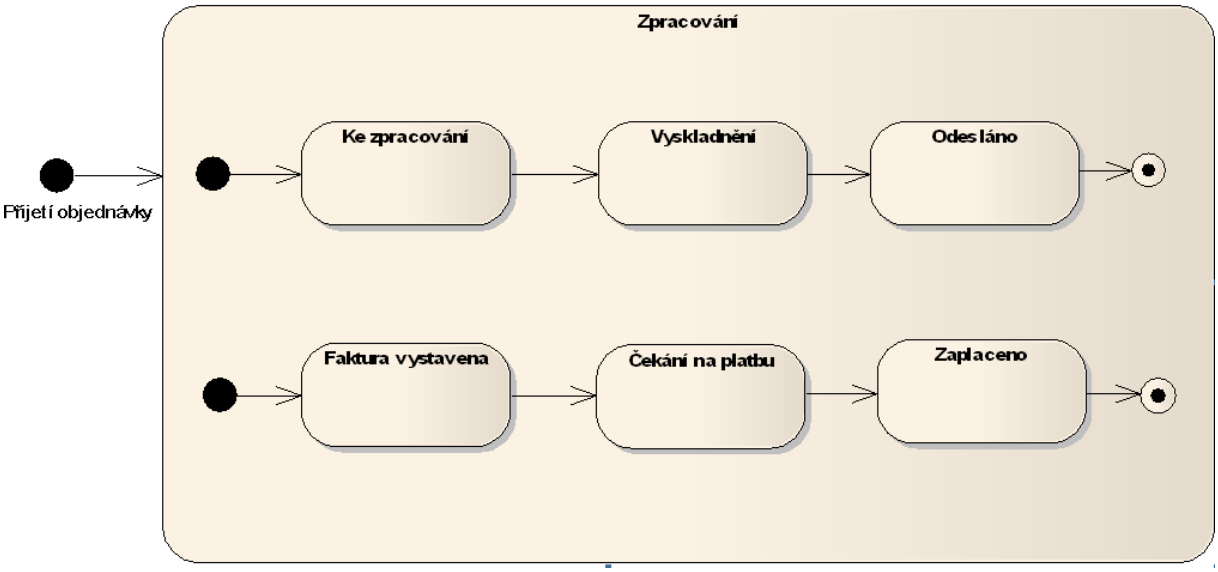
- **ukončení** – po dosažení tohoto pseudostavu jsou veškeré činnosti ukončeny. Výjimkou je činnost vykonaná u přechodu do tohoto pseudostavu.
- **koncový** (pseudostav) – ukončení činnosti celého stavového automatu nebo složeného stavu v případě jeho přechodu do koncového stavu. Z tohoto koncového pseudostavu nejsou již definovány žádné další přechody. Tento stav rovněž nemá definovány žádné vnitřní aktivity.

Pro znázornění pseudostavů UML využívá následující symboly:



Obrázek 4-32 Symboly pseudostavů stavového diagramu. Zdroj: https://upload.wikimedia.org/wikipedia/commons/a/a5/Pseudostavy.PNG?1514543239109

Termínem **složený stav** označujeme stavy, které obsahují jeden nebo více vnořených stavových automatů.



Obrázek 4-33 Složené stavy. Zdroj: https://upload.wikimedia.org/wikipedia/commons/4/4e/Ortogonalni_pseudostavy.PNG?1514534149436

4.3.12 Sekvenční diagram

Účelem sekvenčního diagramu je:

- popsat **spolupráci** mezi objekty,
- popsat **komunikaci objektů v čase**,
- identifikovat **události (zprávy)** vyměňované mezi objekty.

Vstupy pro tvorbu sekvenčního diagramu jsou:

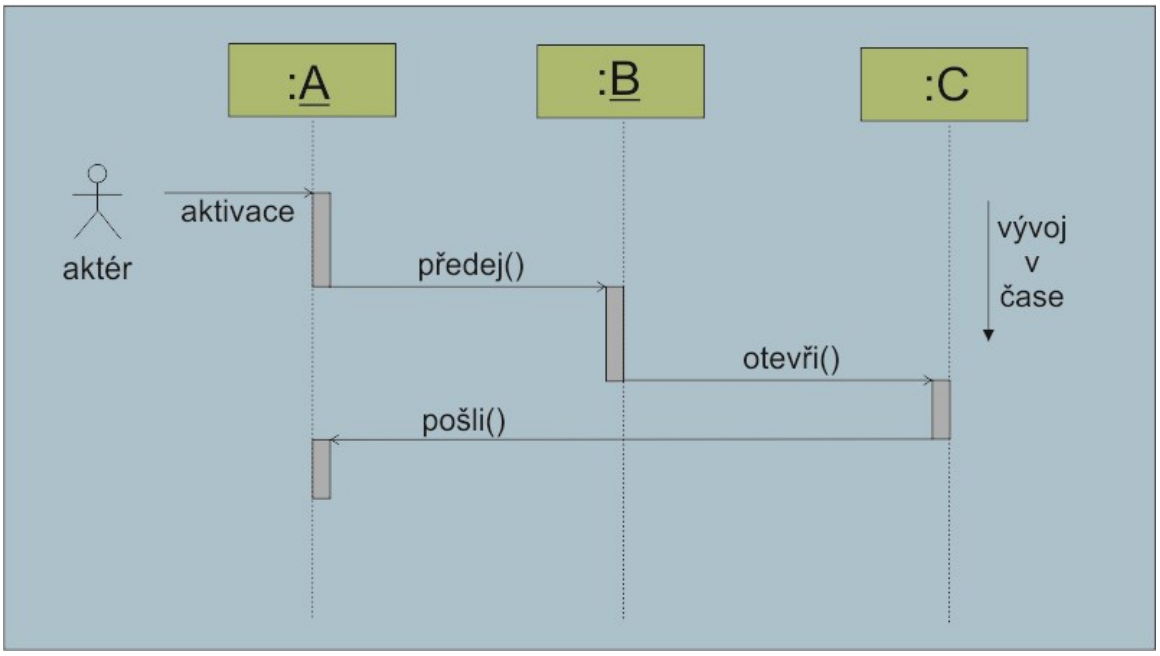
- slovní scénáře diagramu případů užití,
- diagram tříd.

Poznámky k tvorbě sekvenčního diagramu:

- slouží k popisu interakcí tříd během realizace případu užití,
- k popisu vybíráme jen klíčové případy užití,
- další přidáváme podle potřeby – iterativní přístup během tvorby.

- Sekvenční diagram je dvourozměrným modelem vyvíjeného SW:
- na horizontální ose znázorňujeme jednotlivé objekty,
- vertikální osa je osou časovou (tzv. životočáry).

Příklad sekvenčního diagramu:



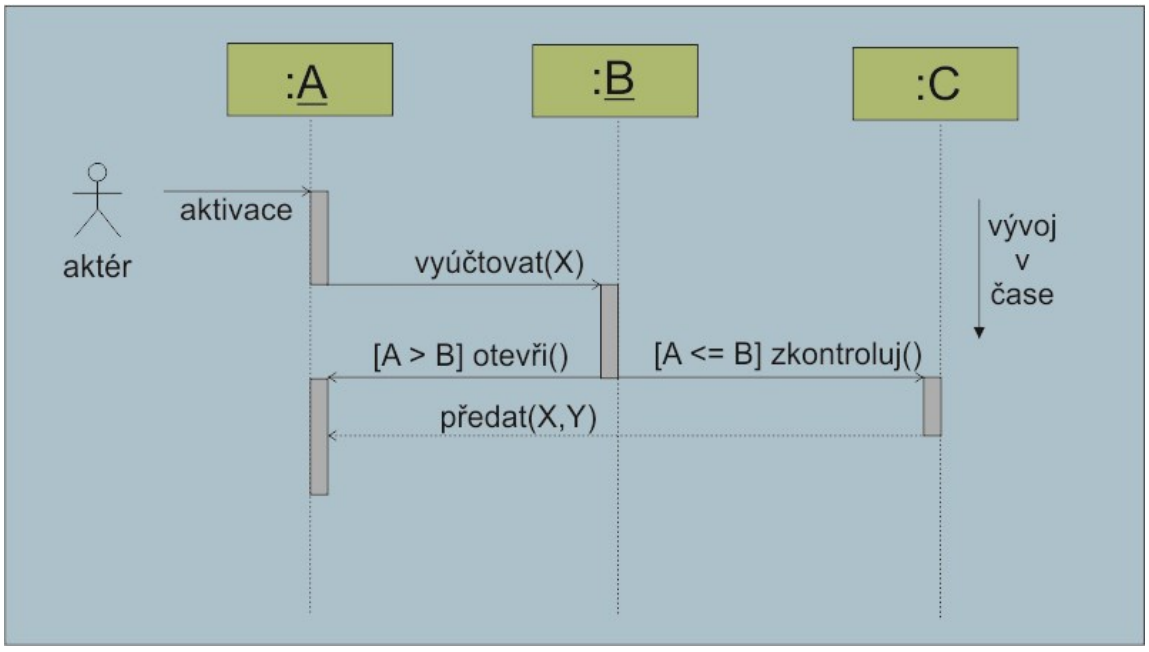
Obrázek 4-34 Sekvenční diagram

Zprávy jsou v sekvenčním diagramu znázorňovány následujícími symboly:

Základní typy zpráv:

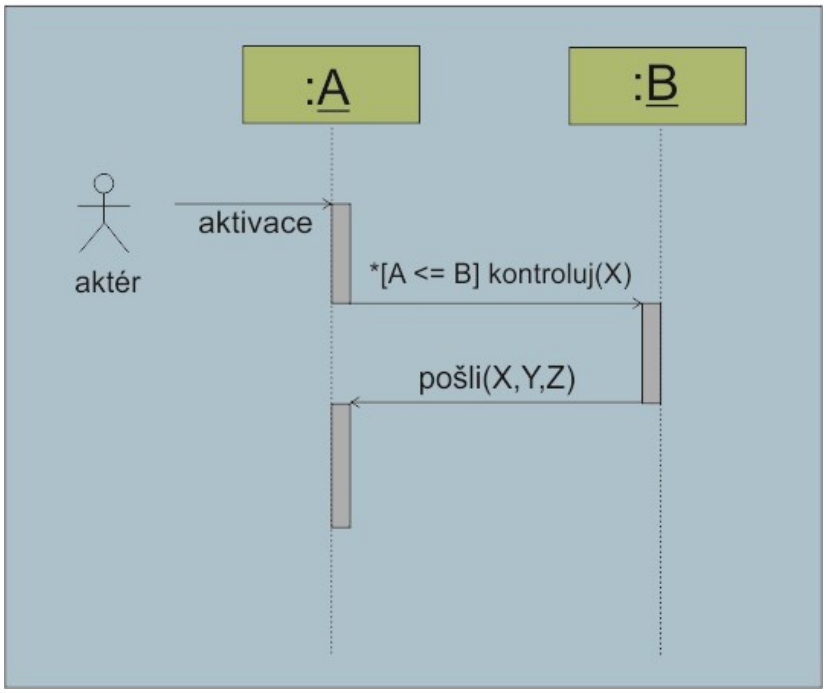
- zpráva synchronní,
- zpráva asynchronní,
- návrat zprávy,
- poznámka – typ zprávy rozlišujeme až později - ve fázi návrhu.

Zprávy je také možno větvit:



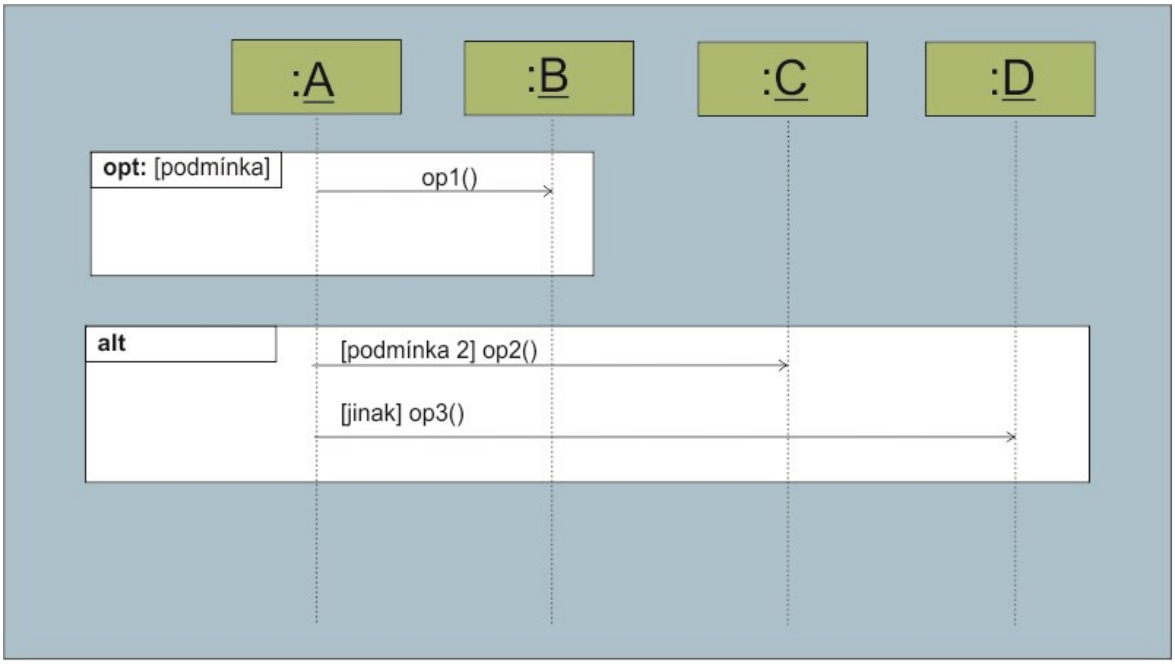
Obrázek 4-35 Sekvenční diagram - příklad větvení

Zprávy je možné také předávat opakovaně:

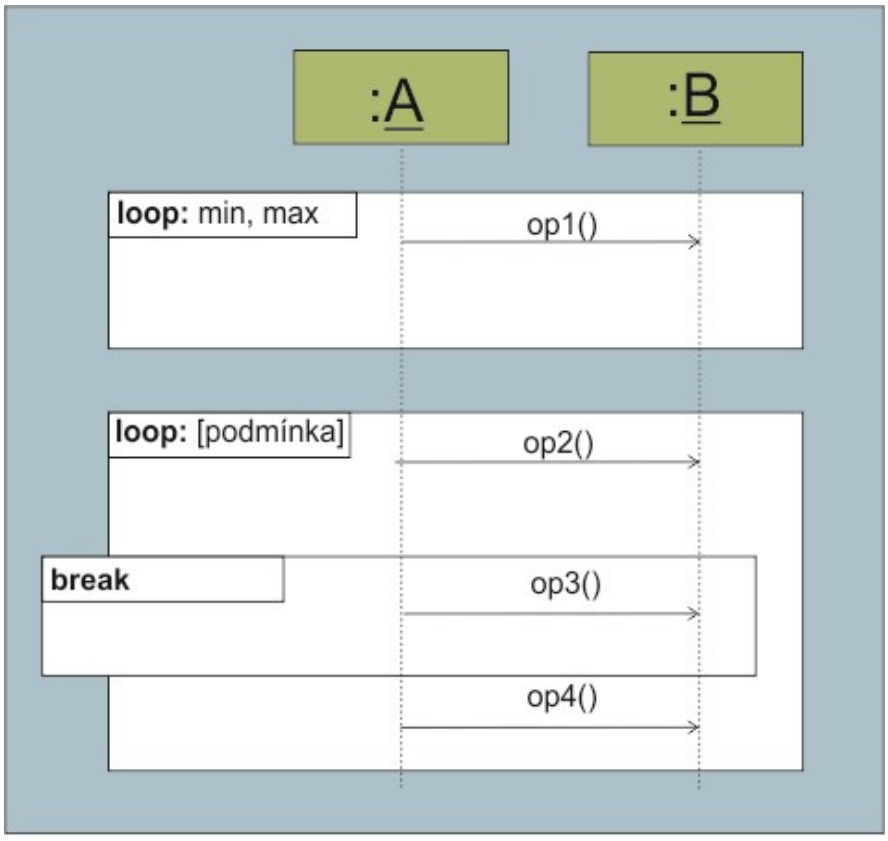


Obrázek 4-36 Sekvenční diagram - příklad cyklického opakování

V sekvenčním diagramu můžeme využívat i tzv. vnořené bloky. Používáme je zejména pro paralelismus, větvení a cykly:



Obrázek 4-37 Použití operátorů větvení a vnořených bloků

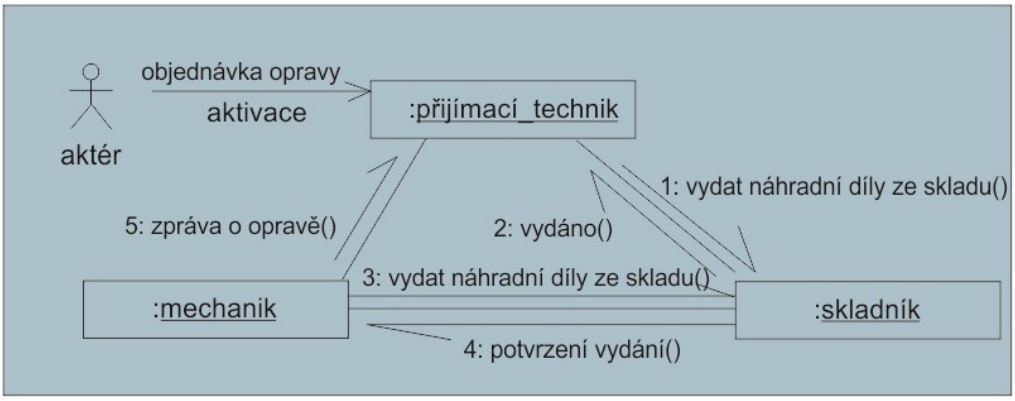


Obrázek 4-38 Použití operátorů pro smyčky

Rozdílem mezi stavovým a sekvenčním diagramem je, že stavový diagram znázorňuje stavy a jejich změny pro každý prvek (objekt) zvlášť, kdežto sekvenční diagram znázorňuje kooperaci vícero objektů v čase.

4.3.13 Diagram komunikace

Z názvu diagramu je zřejmé, že znázorňuje určitou formu komunikace, konkrétně toky zpráv, interakce a vzájemné vztahy mezi instancemi tříd. Ve verzích předcházející UML 2 byl tento diagram nazýván diagramem spolupráce. Diagram nezobrazuje pouze objekty, které spolu komunikují, ale objasňuje rovněž, jakým způsobem tato komunikace probíhá.



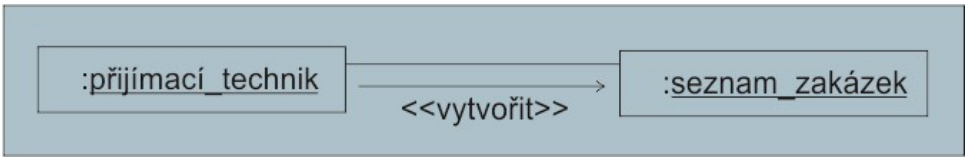
Obrázek 4-39 Ukázka diagramu komunikace mezi mechanikem, skladníkem a přijímacím technikem autoservisu

Znázornění změny stavu objektu:

Obrázek 4-40 Znázornění změny stavu objektu

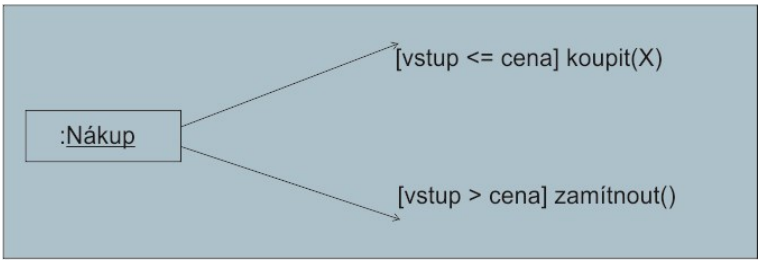
Další využívaná zobrazení:

- vnořování zpráv,
- vytvoření objektu,
- zrušení objektu.

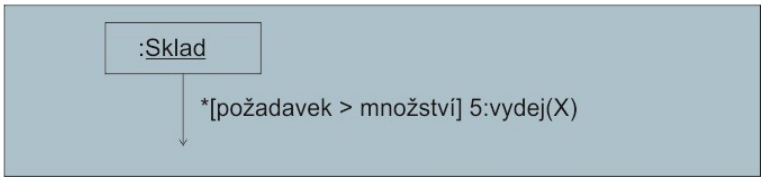


Obrázek 4-41 Tvorba objektu seznam_zakázek

Větvení zpráv a cyklus (opakované zasílání zprávy):



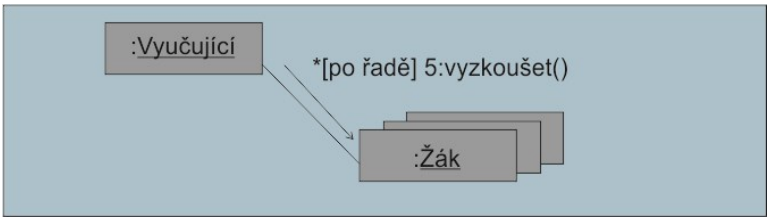
Obrázek 4-42 Ukázka větvení zpráv



Obrázek 4-43 Ukázka opakovaného zasílání zpráv (cyklus)

Zaslání zprávy více objektům postupně (sekvenčně):

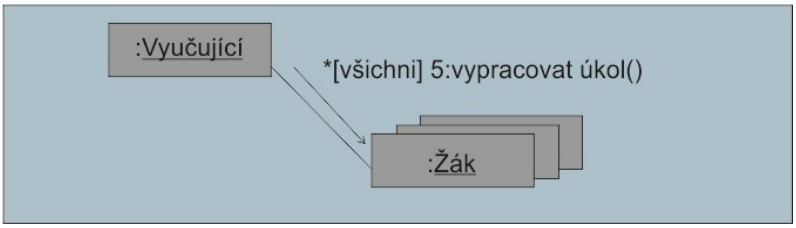
- 10: * vyzkoušet() .... defaultně znamená sekvenční zpracování zpráv.



Obrázek 4-44 Ukázka sekvenčního zpracování zpráv

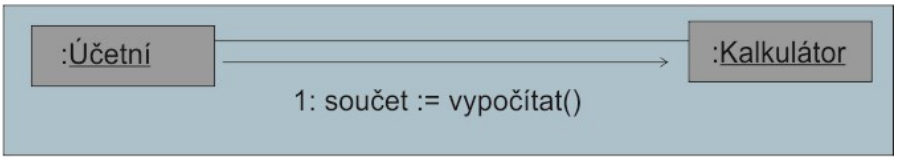
Zaslání zprávy více objektům najednou (paralelně):

- jiný operátor paralelismu zpráv – 10: *// odevzdat úkol().



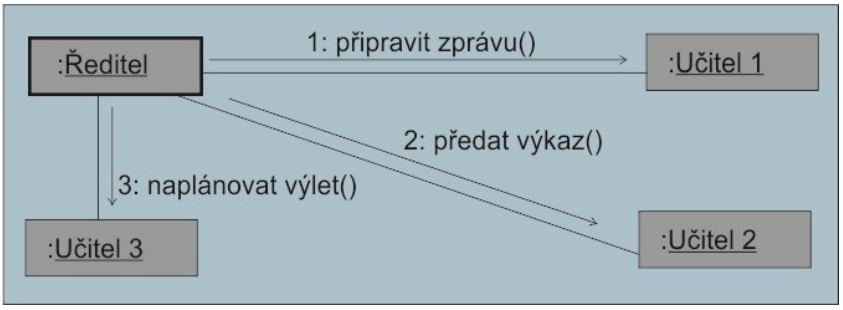
Obrázek 4-45 Zasílání zpráv více objektům zároveň

Vrácení návratové hodnoty:



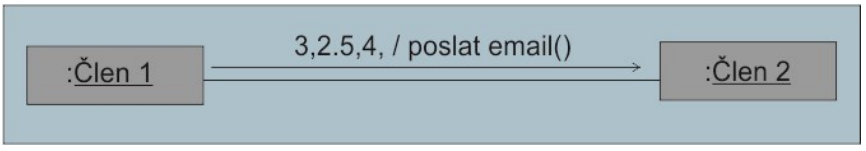
Obrázek 4-46 Vrácení hodnoty "součet"

Aktivní objekt:



Obrázek 4-47 Ukázka aktivního objektu "Učitel"

Synchronizace zpráv:



Obrázek 4-48 Znázornění synchronizace zpráv

Popis zprávy zahrnuje:

- pořadové číslo zprávy,
- název zprávy,

- případně i parametry v závorkách ()
- Vztah diagramu komunikace a sekvenčního diagramu:
- oba modely - jsou významově ekvivalentní
    - obsahují stejné informace, jeden je možné převést na druhý,
- sekvenční diagram - je organizován v závislosti na čase
    - zdůrazňuje, co se děje v čase,
- diagram spolupráce - popisuje kontext a uspořádání spolupracujících objektů
    - zdůrazňuje, co se děje v prostoru.

Každý z obou diagramů zdůrazňuje svůj úhel pohledu na vyvíjený software.

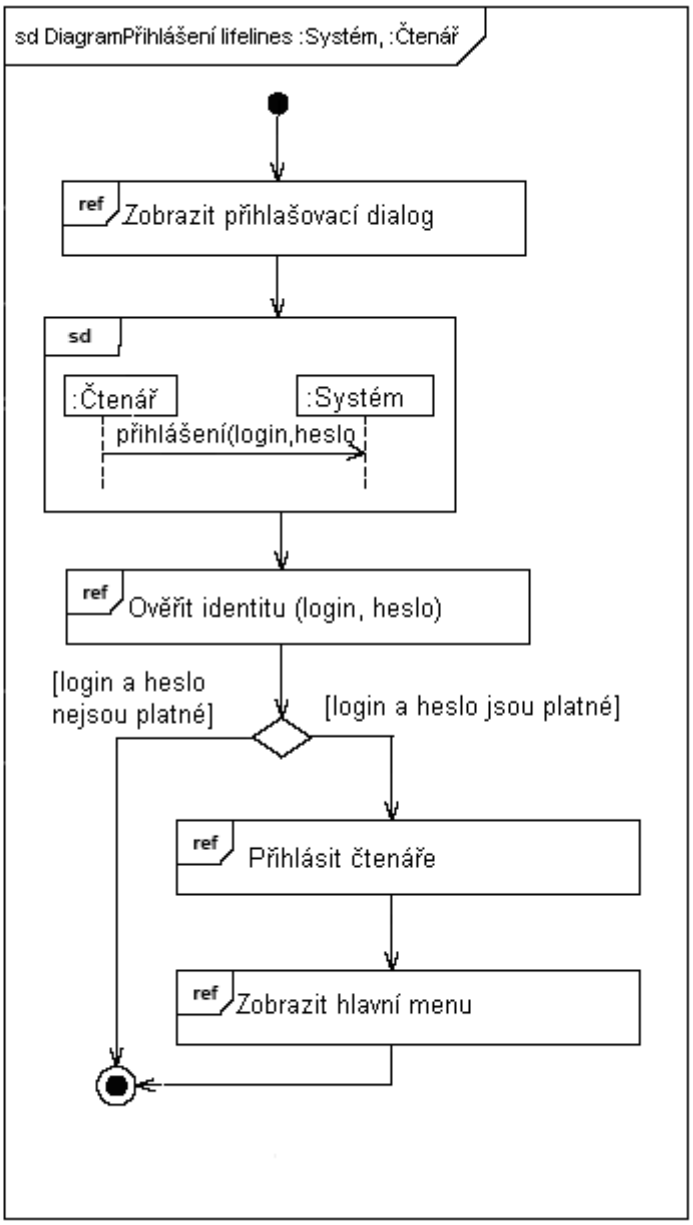
4.3.14 Diagram interakcí

Diagram (přehledu) interakcí je zvláštním případem diagramu aktivit – místo činností zde vystupují interakce. Je kombinací diagramu aktivit a diagramu sekvencí resp. diagramu spolupráce. Zobrazuje řízení toku interakcí během procesu znázorněného diagramem aktivit (např. v průběhu případu užití).

Diagram interakcí používáme k modelování obecné cesty (návazností) mezi popisy interakcí, resp. návazností mezi případy užití:

Use Case --> Diagram aktivit --> Diagram interakcí (sekvence, spolupráce) --> Diagram přehledu interakcí

Pro grafické znázornění využíváme podobného značení jako v diagramech aktivit - větvení, souběžnost, cykly:

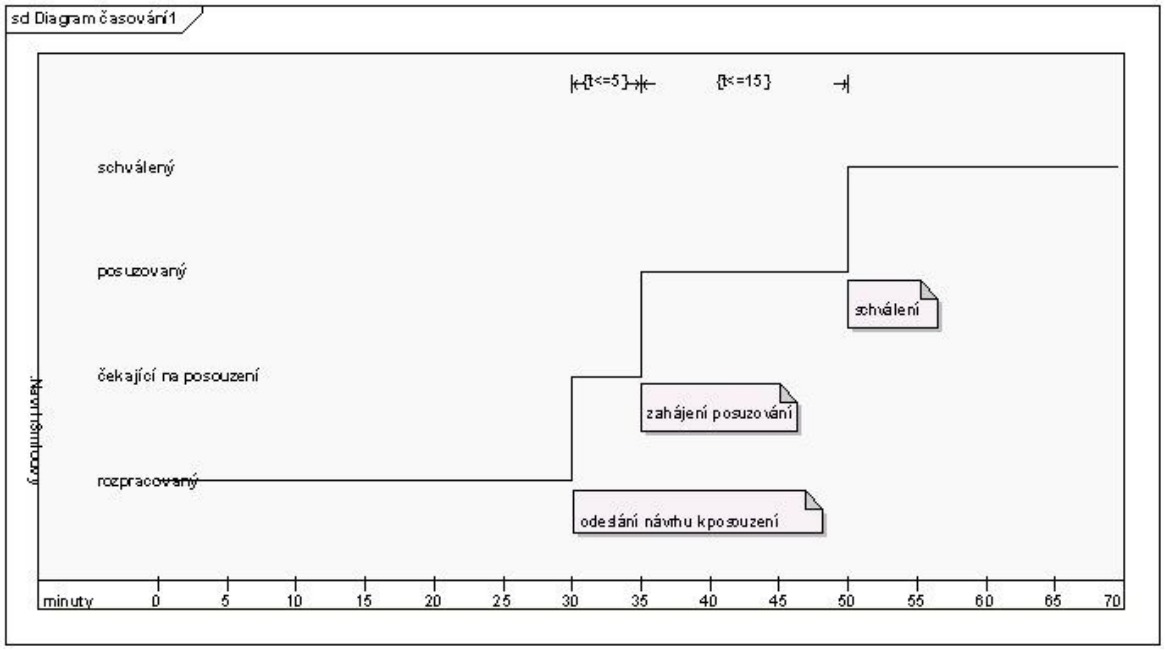


Obrázek 4-49 Diagram interakcí "knihovna". Zdroj: http://orca.xf.cz/ooms/012/012m2_soubory/image001.gif

4.3.15 Diagram časování

Diagram časování znázorňuje průchod objektu jednotlivými stavy v čase a interakce objektu s událostmi a s jinými objekty v čase. Pro přechody mezi stavy je určující **časový údaj** (význam časových oken), tzn., že každá událost musí proběhnout v přesně stanoveném časovém okně. Časový diagram se využívá pro systémy pracující v reálném čase.

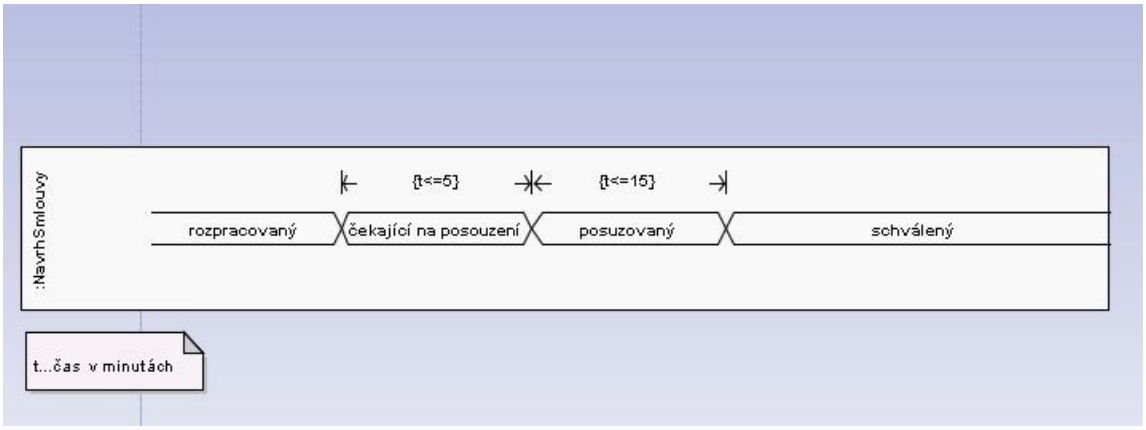
Notace v jazyku UML:



Obrázek 4-50 Diagram časování. Zdroj: http://uml.czweb.org/Diagramy/diagram_casovani1.jpg

Poznámka k obrázku:

- po rozpracování je návrh smlouvy odeslán k posouzení,
- poté je zahájeno posuzování návrhu smlouvy,
- po fázi posouzení je návrh smlouvy schválen.



Obrázek 4-51 Diagram časování - kompaktní forma

4.4 Vztah UP a UML

Jazyk UML není vázán na žádnou konkrétní metodiku anebo životní cyklus. Může být použit v podstatě se všemi existujícími metodikami. Metodika Unified Process využívá jazyk UML jako vlastní syntaxi pro vizuální modelování. Z tohoto pohledu je možné metodiku Unified Process považovat za upřednostňovanou metodiku při používání jazyka UML, protože je pro ni tento jazyk nejlépe přizpůsoben. Jazyk UML však může poskytovat (a také poskytuje) podporu vizuálního modelování i pro jiné metodiky a metody. Záměrem jazyka UML a metodiky UP byla od jejich vzniku podpora nejlepších známých postupů využívaných v softwarovém inženýrství, vycházejících z ověřených zkušeností. K tomuto účelu byly v jazyku UML a metodice UP sjednoceny všechny dřívější pokusy o tvorbu jazyků pro vizuální modelování a proces vývoje softwaru.

Modelovací jazyk UML je především souhrnem grafických notací sloužících ke znázornění návrhových a analytických modelů. Jazyk UML umožňuje modelovat jednoduché i složité aplikace pomocí stejné formální syntaxe a proto je možné výsledky práce sdílet s ostatními návrháři v týmu. Vybrané modely slouží samozřejmě i objednateli aplikace, což umožňuje kvalitnější ujasnění požadavků na navrhovaný systém. Žádný z diagramů nezachycuje vytvářený systém jako celek, ale soustřeďuje se vždy na některou jeho část, nebo lépe řečeno – na jeden pohled na daný systém. Různé pohledy pak tvoří přehledný a komplexní celek pro analýzu, programování, tvorbu dokumentace a ujasňování požadavků. Ve světě již existují různé metodické postupy vycházející z technik modelování v UML, které tyto postupy dále rozšiřují o vlastní doporučené postupy, diagramy a techniky. Nejznámější z nich je metodika RUP společnosti Rational, o které bude řeč v následující kapitole.

Shrnutí pojmů 4.1.

V této obsáhlé kapitole jsme se seznámili se strukturou metodiky UP a uvedli si její hlavní vlastnosti. Naučili jsme se, že metodika UP **musí být přizpůsobena cílům a podmínkám daného vývoje**, tzn. používaným normám, šablonám, nástrojům, atd.

Objasnili jsme pojem unifikovaný proces (UP):

- řízení požadavky a případy užití.
- řízení rizikem.
- základ na architektuře.
- iterativní a přírůstkový proces vývoje SW produktu

a zjistili jsme, že je rozdělen do jednotlivých iterací, z nichž každá prochází pěti základními pracovními procesy (činnostmi):

- stanovení požadavků
- analýza
- návrh
- implementace
- testování

Každá činnost v UP je dále rozdělena do pěti fází:

- zahájení
- rozpracování
- realizace
- zavedení

V další části kapitoly jsme probrali jazyk UML (Unified Modelling Language). Jak již z názvu vyplývá, jedná se o nástroj k modelování procesů. UML pracuje i s tzv. metamodelem, který definuje strukturu všechny modelů UML.

Víme, že UML je tvořen třemi hlavními prvky:

- **stavební bloky** – základní prvky modelu, diagramy, vazby (relace)
- **společné mechanizmy** – obecné způsoby, pomocí nichž je v jazyku UML možno dosáhnout specifických cílů
- **architektura** – vizualizace architektury navrhovaného systému

Modelování v UML (verze 2.0) provádíme prostřednictvím následujících diagramů, které můžeme rozdělit do tří hlavních kategorií – strukturní diagramy, diagramy chování a diagramy interakce:

strukturní diagramy:

- diagram tříd
- diagram komponent
- diagram složených struktur
- diagram nasazení
- diagram balíčků
- diagram objektů, též se nazývá diagram instancí
- diagram profilů

diagramy chování:

- diagram aktivit
- diagram užití
- stavový diagram

diagramy interakce:

- sekvenční diagram
- diagram komunikace
- diagram interakcí
- diagram časování

Otázky 4.1.

1. Charakterizujte metodiku UP – na jakých čtyřech hlavních principech je postavena?
2. Vysvětlete pojmy iterace a přírůstek.
3. Kolika iteracemi může být tvořena každá fáze UP?
4. Co je to milník?
5. K čemu se využívá UML?
6. Čím je UML tvořen?
7. Do jakých třech hlavních kategorií lze rozdělit diagramy v UML?

8. Vyjmenujte všech 14 diagramů a uveďte stručně jejich vlastnosti a k jakému účelu slouží.

5 METODIKA RUP A EUP: RUP (RATIONAL UNIFIED PROCESS) CHARAKTERISTIKA, ZPŮSOB DISTRIBUCE, NOTACE, ZÁKLADNÍ ELEMENTY, POSLOUPNOST AKCÍ. EUP SROVNÁNÍ A SPOLEČNÉ APLIKACE S RUP.

V této kapitole se seznámíme s metodikami, které rozšiřují a obohacují metodiku UP. Jedná se o metodiky RUP a EUP.

Čas ke studiu: 2 hodiny

Cíl: Po prostudování této kapitoly budete umět

- Objasnit, z čeho vychází metodika RUP.
- Charakterizovat metodiku RUP.
- Uvést odlišnosti metodik RUP a UP.
- Objasnit hlavní principy metodiky RUP.
- Charakterizovat metodiku EUP.
- Uvést způsoby distribuce RUP a EUP.

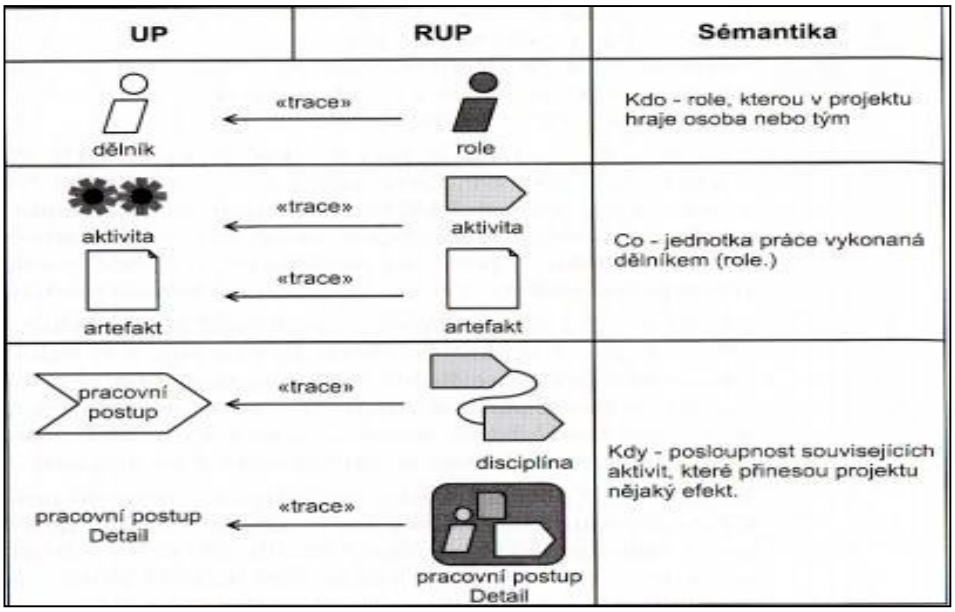
Výklad

5.1 RUP (Rational Unified Process)

Metodika RUP je komerční verzí UP. RUP obsahuje veškeré normy obsažené v UP. Je dodávána s bohatým uživatelským prostředím. RUP rozšiřuje UP mnoha významnými způsoby. Ačkoli obě metodiky mají mnoho společného, rozdíly jsou spíše v jednotlivých detailech a v úplnosti, než v sémantice.

5.1.1 Charakteristika RUP

RUP, v porovnání s UP, zavádí některé syntaktické odlišnosti, uvedené na následujícím obrázku:



Obrázek 5-1 Odlišnosti syntaxe RUP a UP. Zdroj: ARLOW, Jim a Ila NEUSTADT. UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky. 2., aktualiz. a dopl. vyd. Brno: Computer Press, 2007. ISBN 978-80-251-1503-9.

Metoda RUP využívá během vývoje následující ověřené postupy:

- iterativní vývoj (postupné zjemňování, přidávání vlastností)
- správa uživatelských požadavků (doplňování, třídění, hodnocení)
- použití architektury komponent (využití stávajících vzorů, komponent částečných řešení)
- vizuální modelování (vytváření modelů reality)
- sledování kvality

Podobně jako UP, RUP má 4 etapy vývoje:

- zahájení
- rozpracování
- realizace
- zavedení

5.1.2 Způsob distribuce RUP

Vzhledem ke skutečnosti, že metodika RUP je komerčním produktem, pro její využití je nezbytné řádně dodržovat distribuční a licenční podmínky vlastníka práv k této metodice, kterým je v současné době (2017) společnost IBM.

Distribuce je prováděna na základě zakoupení licence k používání této metodiky od distributorů této společnosti. Ve většině případů je platnost licence roční a po uplynutí její platnosti je nutné ji (zaplacením licenčního poplatku) obnovit.

Bližší informace týkající se distribuce RUP je možné získat online, například na stránkách firmy cnet - https://www.cnet.com/products/ibm-rational-unified-process-software-subscription-and-support-renewal-1-year-e013bllsb/specs/ nebo přímo u výrobce - https://www.ibm.com.

Firma IBM nabízí licenci k využívání RUP rovněž jako součást jiného balíčku s názvem Rational Method Composer.

Metodika RUP je distribuována jako sada webových stránek (HTML), které tvoří znalostní bázi.

5.1.3 Notace RUP

Díky faktu, že metodika RUP je postavena na základě UP, využívá tedy shodné diagramy a postupy a její notace je v podstatě shodná s notací UP. Drobné syntaktické odchylky mezi metodikami RUP a UP jsou uvedeny v předchozí kapitole 5.1.1 Charakteristika RUP.

5.1.4 Základní elementy RUP

Metodika RUP se opírá o čtyři hlavní prvky, na kterých je celý vývoj software v této metodice založen. Jedná se o následující pojmy:

- pracovníci (workers)
- činnosti (activities)
- meziprodukty (artifacts)
- pracovní procesy (workflows)

Stručně nyní objasníme význam těchto pojmů metodiky RUP.

Pracovníci – jedná se o elementy odpovídající na otázku **KDO**?. Pracovník určuje odpovědnost skupiny nebo jednotlivce a jejich chování. Chování pracovníka se definuje pomocí činností – aktivit. Každý pracovník má tedy vazbu s množinou aktivit. Odpovědnost je zpravidla určena vztahy s meziprodukty – pracovník je generuje, upravuje, kontroluje… Pracovník není většinou konkrétní osoba, pod tímto pojmem se spíše skrývá určitá role, která může být konkrétním osobám přidělována. Pracovník může mít více rolí a role je možné přidělovat pracovníkům dle konkrétních požadavků.

Činnosti – jsou odpovědí na otázku **JAK**? Jedná se v podstatě o jednotku práce vykonávané jednotlivcem nebo skupinou. Výstupem činnosti by měl být výsledek, který je v rámci projektu smysluplný. Činnost má jednoznačně určený záměr, cíl (čeho danou činností dosáhneme?) Tímto záměrem nebo cílem je ve valné většině případů úprava nebo tvorba meziproduktu – třídy, plánu, modelu, atd. Obvykle se činnost vztahuje k jednomu pracovníku a dotýká se jen nevelkého počtu meziproduktů. Do činnosti mohou meziprodukty vstupovat a upravené mohou být zase jejím výstupem.

Jako příklad výše uvedeného může posloužit následující:

Projektový manažer vytváří plán. Pracovník bude tedy projektový manažer a jeho činností bude vytvoření plánu. Meziproduktem se stane vytvořený plán.

Aktivity je možné rozdělit do dalších kroků:

- úvahy (Thinking Steps)
- provádění (Performing Steps)
- přezkoumání (Reviewing Steps)

U každé činnosti je nutné stanovit jasný sled kroků vedoucích k jejímu úspěšnému dokončení. Ke každému z kroků činnosti je v RUP vytvořena velmi detailní dokumentace. Využít přitom lze množství návodů, tipů, pomůcek, atd., které metodika RUP poskytuje.

Meziprodukty – dávají odpověď na otázku **CO**? Jde o určitou informaci nebo její část, která je v rámci daného procesu generována, upravována či používána. Meziprodukty jsou reálnými výsledky projektů. Využívány jsou jako vstupy do činností prováděných pracovníky nebo jako jejich výstupy. Určený pracovník má rovněž i odpovědnost za správnost vygenerovaného nebo upraveného meziproduktu.

Meziprodukty mohou mít více podob. Patří mezi ně:

- model (např. model případů užití, model návrhu, atd.)
- element modelu (např. třída, případ užití, podsystém)
- dokument (např. část specifikace, plán)
- zdrojový kód
- spustitelná aplikace

Meziprodukty je možné generovat pomocí mnoha různých nástrojů, kupříkladu Rational Rose pro tvorbu modelů, Microsoft Project pro tvorbu projektů, atd.

Díky opravdu velkému počtu meziproduktů definovaných v RUP je žádoucí jejich rozdělení do několika skupin:

- meziprodukty týkající se požadavků (Requirements Artifact Set)
- meziprodukty týkající se analýzy a designu (Analysis & Design Artifact Set)
- meziprodukty týkající se implementace (Implementation Artifact Set)
- meziprodukty týkající se testování (Test Artifact Set)
- meziprodukty týkající se šíření (Deployment Artifact Set)
- meziprodukty týkající se konfiguračního a změnového řízení (Configuration & Change Management Artifact Set)
- meziprodukty týkající se projektového řízení (Project Management Artifact Set)
- meziprodukty týkající se vývojového prostředí (Environment Artifact Set)
- meziprodukty týkající se obchodního modelování (Business Modelling Artifact Set)

I přes značný počet meziproduktů definovaných v RUP je jejich využití nebo nevyužití dáno konkrétním potřebám projektu. Stejně jako je tomu u pracovníků a činností, zpravidla se využívá jen určitá podmnožina množiny všech meziproduktů (pracovníků, činností…).

Pracovní procesy – poskytují odpověď na otázku **KDY**? Proces není tvořen pouze seznamem činností, pracovníků a meziproduktů. Nezbytné je ještě přidání série aktivit a interakcí mezi pracovníky. Z uvedeného vyplývá, že pod pojmem pracovní proces se rozumí sled činností, které směřují ke splnění vytýčeného cíle. Z kapitoly ***Chyba! Nenalezen zdroj odkazů.****2 Struktura UP* víme, že takovouto posloupnost aktivit můžeme modelovat pomocí Diagramu interakcí a Sekvenčního diagramu. Bohužel, ani pomocí těchto sofistikovaných nástrojů není možné většinou obsáhnout úplně všechny závislosti a vazby mezi činnostmi. Intepretaci diagramů nelze tedy provádět čistě mechanicky, je žádoucí se nad různými závislostmi dobře zamyslet.

Každý pracovní proces je tvořen řadou aktivit. Obvykle těchto činností bývá nanejvýš deset. Metodika RUP poskytuje devět definovaných hlavních pracovních procesů. Tyto procesy korespondují se skupinami meziproduktů uvedenými v předchozím odstavci.

Vzhledem ke skutečnosti, že každý z těchto devíti pracovních procesů zasahuje poměrně velkou oblast, jsou metodikou RUP dále rozděleny do skupiny, které říkáme **podrobnosti pracovních procesů**. Tyto skupiny sdružují procesy, které spolu úzce souvisí.

5.1.5 Posloupnost akcí RUP

Metodika RUP se řadí mezi metodiky, které jsou řízeny **případy užití** (use-case driven approach). To znamená, že za základní prvek je považován případ užití, který je definován jako sekvence akcí, které provádí systém nebo které jsou prováděny uvnitř systému. Posloupnost akcí v RUP je tedy v daném projektu řízena případem užití.

5.2 EUP (Enterpsise Unified Process)

Metodika EUP je další komerční verzí vycházející z RUP. Podobně, jako RUP rozšiřuje a obohacuje UP, stejně tak EUP rozšiřuje a obohacuje procesní Framework RUP. Toto rozšíření se podrobuje normě ISO12207. Metodika EUP má ve své snaze pokrytí celého životního cyklu vývoje software.

5.2.1 Charakteristika EUP

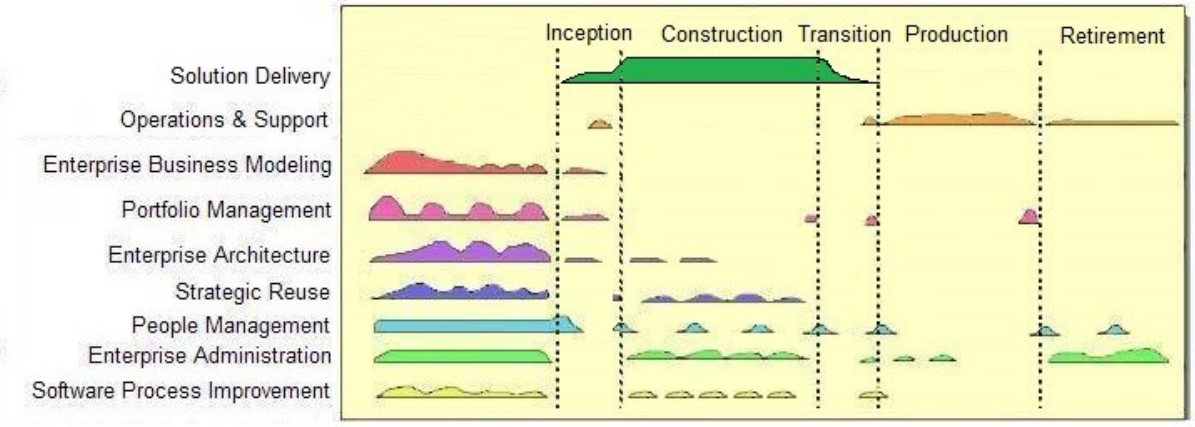
Základ EUP rozšiřuje RUP o dvě fáze:

- fáze produkční (production)
- fáze stažení z provozu (retirement)

Součástí metodiky EUP jsou rovněž nově zavedené součásti:

- provoz a podpora (Operations and Support)
- modelování podnikových procesů (Enterprise Business Modelling)
- správa portfolia (Portfolio Management)
- podniková architektura (Enterprise Architecture)
- strategie znovupoužitelných komponent, postupů a šablon (Strategic Reuse)
- zlepšování softwarového procesu (Software Process Improvement)

Výhodou EUP je napojení na podnikové procesy a neustálé zlepšování.



Obrázek 5-2: Metodika EUP. Zdroj: http://enterpriseunifiedprocess.com/

5.2.2 Způsob distribuce EUP

EUP je intelektuálním vlastnictvím společnosti Ambysoft Inc. Pro informace o získání licence k tomuto produktu je nutné firmu kontaktovat a individuálně dohodnout podmínky jejího udělení. V současné době nemá tato společnost dostupné veřejné informace o způsobu udělování práv k využívání metody EUP a jejich ceně – je nezbytné společnost kontakt, což je možné učinit na webových stránkách http://enterpriseunifiedprocess.com/licensing.html.

5.2.3 Notace EUP

Vzhledem ke skutečnosti, že metodika EUP je postavena na základě RUP, využívá tedy shodné diagramy a postupy a její notace je v podstatě shodná s notací RUP a UP.

5.2.4 Základní elementy EUP

Jak již bylo zmíněno, metodika EUP je postavena na základech RUP, kterou rozšiřuje a proto využívá stejné základní elementy jako RUP. Odkazujeme proto na kapitolu 5.1.4 Základní elementy RUP.

5.2.5 Posloupnost akcí EUP

Metodika EUP se, stejně jako RUP, řadí mezi metodiky, které jsou řízeny **případy užití** (use-case driven approach). To znamená, že za základní prvek je považován případ užití, který je definován jako sekvence akcí, které provádí systém nebo které jsou prováděny uvnitř systému. Posloupnost akcí v EUP je tedy v daném projektu řízena případem užití.

Shrnutí pojmů 5.1.

V této kapitole jsme se zabývali komerční verzí metodiky UP – metodikou RUP. Dozvěděli jsme se, že metodika RUP je dodávána s bohatým uživatelským rozhraním a od metodiky UP se odlišuje několika syntaktickými vlastnostmi. RUP využívá následující ověřené postupy:

- iterativní vývoj (postupné zjemňování, přidávání vlastností)
- správa uživatelských požadavků (doplňování, třídění, hodnocení)
- použití architektury komponent (využití stávajících vzorů, komponent částečných řešení)
- vizuální modelování (vytváření modelů reality)
- sledování kvality

- a má 4 etapy vývoje:

- zahájení
- rozpracování
- realizace
- zavedení

Důležité pro metodiku RPU jsou rovněž čtyři základní prvky, na kterých je vystavěna:

- pracovníci (workers)
- činnosti (activities)
- meziprodukty (artifacts)
- pracovní procesy (workflows)

Distribuce RUP je prováděna na základě zakoupení licence k používání této metodiky od distributorů společnosti IBM.

V další části kapitoly jsme se věnovali metodice EUP, která vychází z metodiky RUP (obohacuje ji o další fáze):

- fáze produkční (production)
- fáze stažení z provozu (retirement)

Nově rovněž zavádí další součásti:

- provoz a podpora (Operations and Support)
- modelování podnikových procesů (Enterprise Business Modelling)
- správa portfolia (Portfolio Management)
- podniková architektura (Enterprise Architecture)
- strategie znovupoužitelných komponent, postupů a šablon (Strategic Reuse)
- zlepšování softwarového procesu (Software Process Improvement)

Notace RUP se shoduje s notací RUP a vychází se stejných principů, pokud jde o posloupnost akcí a základní prvky.

Distribuce EUP je zajišťována společností Ambysoft Inc.

Otázky 5.1.

1. Jaké jsou hlavní postupy metodiky RUP?

2. Jak byste charakterizovali notaci RUP?
3. Z čeho RUP vychází (z jaké metodiky)?
4. Jakým přístupem se řídí metodiky RUP?
5. Popište posloupnost činností metodiky RUP.
6. Uveďte hlavní rozdíly RUP a EUP.
7. Jaká je notace EUP?
8. Je možné RUP resp. EUP volně používat? Pokud ne, uveďte podmínky, za kterých by to bylo možné.
9. Kdo je distributorem RUP a EUP?

6 AGILNÍ PŘÍSTUP K TVORBĚ SW: VÝHODY AGILNÍCH METODIK (RYCHLOST, WEBOVÉ TECHNOLOGIE, ITERATIVITA, INKREMENTACE). MANIFEST AGILNÍHO VÝVOJE SW (THE AGILE MANIFESTO).

V této kapitole se seznámíme s dalším typem metodik, které nazýváme agilní metodiky. Uvedeme si, v čem se agilní metodiky liší od rigorózních a jakými třemi základními principy se liší. Naučíme se rovněž, v jakých případech je využití agilních metodik vhodné a uvedeme důvody proč.

Čas ke studiu: 2 hodiny

Cíl: Po prostudování této kapitoly budete umět

- Vysvětlit pojem *agilní metodika*.
- Objasnit rozdíl mezi agilní a rigorózní metodikou.
- Uvést hlavní pilíře agilních metodik.
- Vysvětlit pojem *manifest agilního vývoje*.
- Charakterizovat složení týmu agilního vývoje.

Výklad

6.1 Agilní metodiky

Agilní metodiky se snaží **oprostit od náročnějších postupů vývoje v zájmu rychlosti vývoje**. Jejich vznik reaguje na potřeby vývoje určitých typů IS (menší systémy, www aplikace apod.), pro které mohou být doporučení stávajících metodik zbytečně složitá a neúměrná vyvíjenému systému.

Agilní metodiky vznikly tedy zejména proto, že klasické přístupy jsou někdy administrativně náročné a nepružné, tj. neúměrně vzhledem k rozsahu a typu vyvíjeného IS. Rovněž zadání mnohdy není zcela jasné, často se mění – v těchto případech je agilní přístup téměř nezbytný. Před jejich vznikem se používaly jen těžké, rigorózní metodiky. Ty mají své odpůrce jednak díky vodopádovému modelu a jednak díky tomu, že vedoucí projektu omezuje

vývojáře v práci svým blízkým dohledem. Rigorózní metodiky také těžkopádně reagují na jakékoli změny. Díky uvedeným skutečnostem začaly vznikat odlehčené metodiky, které se (podle názoru jejich autorů) vracejí k praktikám využívaných na samotném počátku vývoje software.

Agilními metodikami se odlehčené metodiky nazývají od doby vydání Manifestu agilního vývoje SW.

Agilní metodiky se řídí třemi základními principy:

- **přírůstkový (iterativní) vývoj s velmi krátkými iteracemi** – nejprve se tvoří nejdůležitější funkce SW, po odzkoušení zákazníkem se přidávají další.
- **důraz na komunikaci mezi zákazníkem a vývojářem** – zástupci zákazníka by měli být členy vývojového týmu a podílet se na návrhu systému. Díky krátkým iteracím sdělují vývojářům své zkušenosti.
- **přísné automatizované testování** – pro daný SW je vytvořena komplexní sada testů. Testy předem sestavené a prověřené pro každou změnu SW. Poté je změněný SW testován.

V současné době (2017) existuje řada metodik, využívajících agilní přístup k vývoji SW. Mezi nejznámější z nich patří:

- ASD (Adaptive Software Development) - Adaptivní vývoj SW,
- FDD (Feature-Driven Development),
- XP (Extreme Programming) - Extrémní programování,
- Lean Development,
- SCRUM,
- Crystal metodiky.

O vybraných metodikách s adoptovaným agilním přístupem bude řeč v následujících kapitolách.

6.2 Manifest agilního vývoje SW

Manifest agilního přístupu byl vytvořen autory: Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland a Dave Thomas.

Manifest zahrnuje **priority** a **principy** agilního vývoje softwaru.

Priority agilního programování

- Lidé a jejich spolupráce před procesy a nástroji
- Fungující software před obsáhlou dokumentací
- Spolupráce se zákazníkem před sjednáváním smluv
- Reakce na změnu před dodržováním plánu

Principy agilního programování

V manifestu je obsaženo následujících dvanáct principů:

1. Nejvyšší prioritou je uspokojení zákazníka prostřednictvím rychlého a průběžného dodávání kvalitního software.
2. Změnové požadavky jsou vítány, dokonce i v průběhu vývoje. Agilní procesy je zpracují tak, aby zákazníkovi přinášely konkurenční výhody.
3. Dodávejte fungující software často, v intervalech týdnů až měsíců. Upřednostňujte kratší intervaly dodání.
4. Lidé z businessu a vývojáři musí spolupracovat každý den během celého projektu.
5. Pro práci na projektu vybírejte motivované jedince. Dejte jim prostředí a podporu, kterou potřebují, a důvěřujte jim, že práci dokončí.
6. Nejúčinnější metoda sdílení informací vývojářskému týmu (i uvnitř tohoto týmu) je osobní setkání.
7. Fungující software je hlavním měřítkem postupu vývoje.
8. Agilní procesy podporují udržitelný vývoj. Sponzoři, vývojáři i uživatelé by měli být schopní dodržovat stálý výkon, dokud je třeba.
9. Průběžná pozornost věnovaná technické dokonalosti a dobrému návrhu posiluje agilní přístup.
10. Základem je jednoduchost – umění co nejvíce práce vůbec nedělat.
11. Nejlepší architektury, požadavky a návrhy vznikají v týmech, které se samy organizují.
12. Tým v pravidelných intervalech vyhodnocuje svou práci a upravuje své postupy tak, aby byl co nejefektivnější.

6.3 Omezení rizik při agilním přístupu

Agilní metodiky **omezují následující rizika**:

- rizika, která souvisejí s nepřesným zadáním (které často představuje problém) a s komplexností vytvářeného systému
- rizika, která souvisejí s nestálostí členů vývojového týmu
- rizika spojená s neexistencí dostatečné dokumentace
- rizika plynoucí z neplnění termínů a lhůt a z překračování plánovaných rozpočtů

Obrázek 6-1 Iterace agilního vývoje

6.4 Složení týmu agilního vývoje

Při agilním vývoji je důležité, aby byl tým složen z vývojářů schopných pružné komunikace a spolupráce spíše než ze specialistů, kteří většinou pracují samostatně. Základním konceptem při agilním vývoji je komunikace.

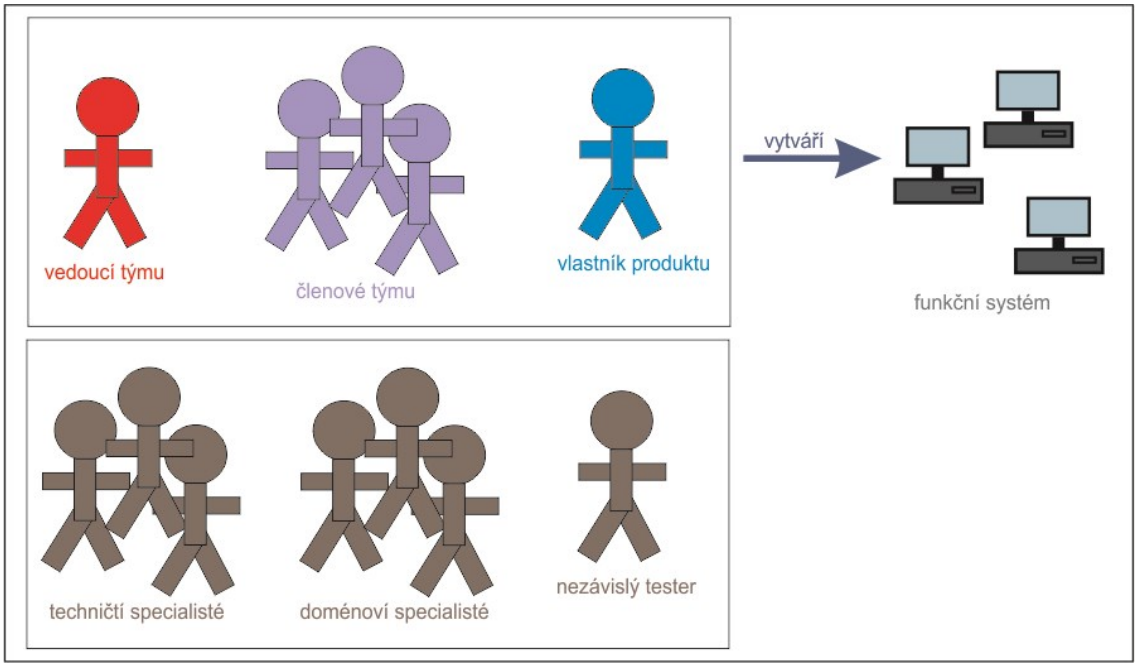
Při agilním vývoji v menším týmu se ve složení týmu obvykle využívají role. Pozor, role nejsou pracovní pozice – každý člen týmu může zastávat jednu nebo více rolí a tyto role je možné v průběhu času měnit. Každá role může mít přiřazen nulový nebo vyšší počet osob, ve kterémkoli bodě v průběhu projektu. Obecně se za role při agilním vývoji považují následující:

- **vedoucí týmu** (vedení projektu, týmový kouč) – tato role odpovídá za podporu týmu, získávání prostředků pro tým a ochranu týmu před problémy. Tato role zahrnuje osobní dovednosti s vedením týmu, ale nikoli technické dovednosti jako je plánování a činnosti, které je výhodnější přenechat ostatním členům týmu.

- **člen týmu** – tato role, která je občas také nazývána programátor či vývojář, je odpovědná za tvorbu a dodávku systému. To zahrnuje činnosti jako modelování, programování, testování a uvolnění hotového produktu, atd.
- **vlastník produktu** – tato role zastupuje investory. Tato role nebo osoba (většinou je v týmu pouze jedna) odpovídá za vytvoření seznamu položek prioritních prací, za včasná rozhodnutí a včasná poskytování informací.
- **investor** – je kdokoli, kdo je přímým uživatelem, nepřímým uživatelem, vedoucím uživatelů, senior manažer, člen provozního týmu, tzv. „zlatý majitel“ poskytující prostředky členům týmu podpory, auditorům, manažerům programu/portfolia, vývojářům pracujícím na jiných systémech, kteří integrují aktuálně vyvíjený projekt, atd.

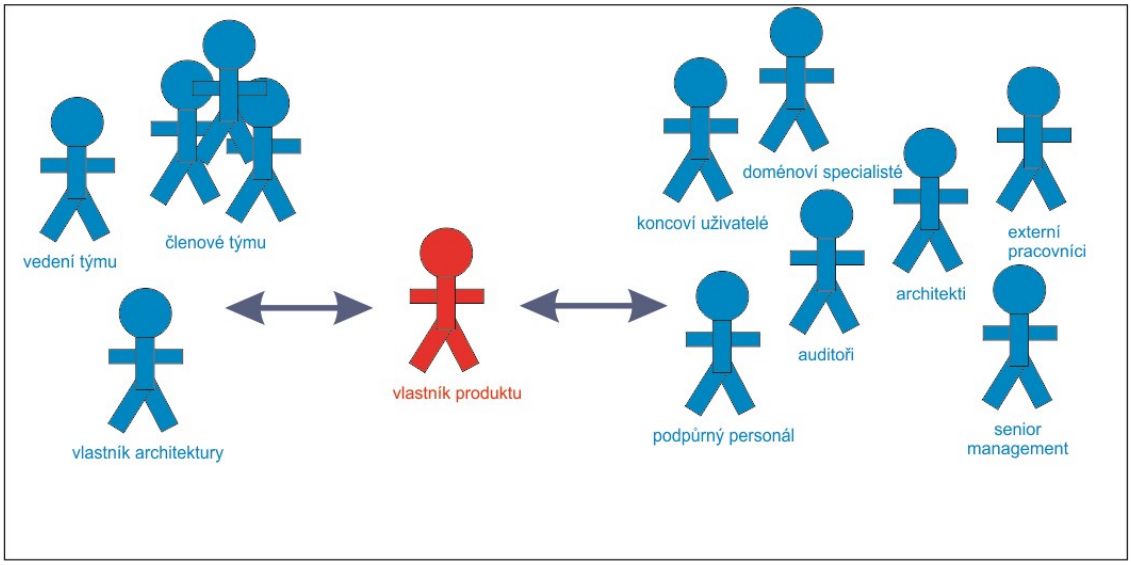
Mezi role týmu lze počítat rovněž ty nepřímé, které jsou však rovněž pro agilní vývoj důležité, ačkoli osoby, které je přebírají, většinou nejsou přímými členy týmu. Jedná se o následující role:

- **techničtí specialisté** (experti) – občas členové týmu potřebují pomoc technických odborníků, jako např. databázových specialistů apod. pro činnosti jako je vytvoření a testování databází, vytvoření sestavovacích skriptů apod. Techničtí experti jsou využívání dle potřeby, nepravidelně, aby pomohli týmu překonat složitý problém a přenést jejich znalosti na jednoho nebo více vývojářů v týmu.
- **doménoví specialisté** – jak je patrné z následujícího obrázku, vlastník projektu zastupuje širokou skupinu investorů, nikoli pouze jednoho koncového uživatele a v praxi není rozumné očekávat, že by se jednalo o odborníky v každém směru ve vaší doméně. Vlastník projektu občas využívá služeb doménových specialistů a zajišťuje jejich občasnou spolupráci s týmem. Jedná se například o daňové specialisty, kteří objasní podrobnosti požadavků anebo sponzora, který objasní vize týkající se projektu, atd.
- **nezávislý tester** – efektivní agilní týmy mají obvykle nezávislého testera, který současně s vývojem pracuje na ověřování práce týmu během celého životního cyklu. Tato role je většinou využívána u velmi komplexních projektů.



Obrázek 6-2 Tým agilního vývoje

Vlastník produktu obvykle reprezentuje více investorů.



Obrázek 6-3 Reprezentace více investorů vlastníkem produktu

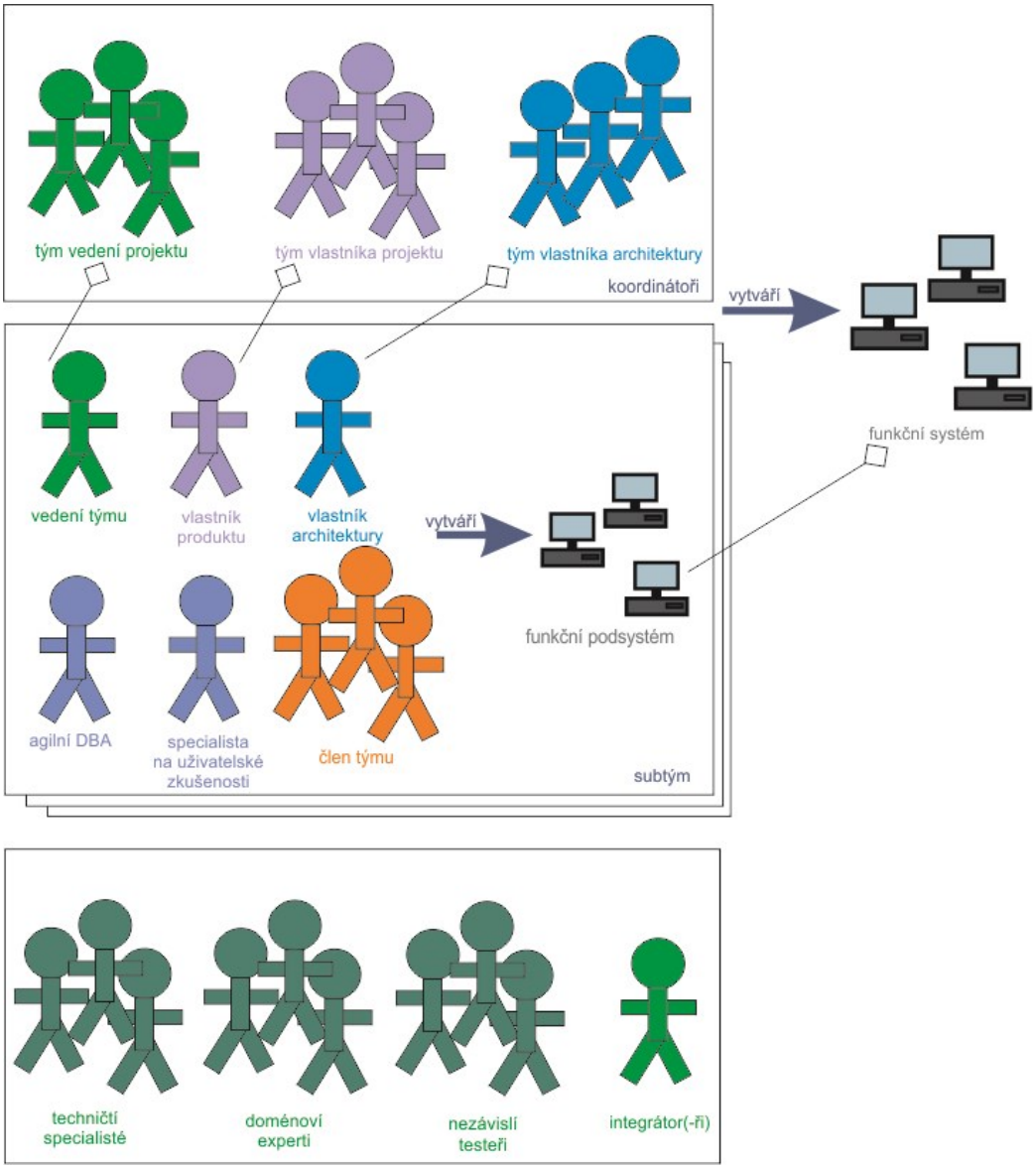
6.4.1 Doplnění týmových rolí u projektů většího rozsahu

V případě větších týmů se role menšího týmu doplňují o následující dvě:

- **vlastník architektury** – tato role je odpovědná za podporu rozhodnutí týkající se architektury u sub-týmu, který je odpovědný za celkový architektonický směr projektu. Vlastník architektury vede svůj sub-team během počátečních představ o jeho

podsystémech a bude zapojen do plánování výchozí architektury pro systém jako celek. Vlastníci architektury se od tradičních architektů liší v tom, že nejsou samostatně odpovědní za určení architektonického směru, ale spíše pomáhají při jeho vytváření (určování) a vývoji.

- **integrátor** – sub-týmy jsou obvykle odpovědné za jeden nebo více podsystémů. Čím větší je celý tým obecně, tím větší a komplikovanější je celý vytvářený systém. V těchto situacích může celý tým potřebovat jednoho člověka nebo více lidí v roli integrátora, zodpovědného za vytvoření (složení) celého systému z jednotlivých dílčích podsystémů. Tito lidé často úzce spolupracují s nezávislým testovacím týmem (pokud takový existuje), který bude v průběhu projektu požadovat provedení testování integrace systému.



Obrázek 6-4 Týmové role u projektů většího rozsahu

6.5 Koordinace činností

Jak obrázek napovídá, u velkých agilních projektů je potřeba koordinovat několik kritických činností:

činnosti spojené s řízením projektu – při managementu projektu není dostatečné se u řešení technických aspektů soustřeďovat pouze na samo-organizaci. To může fungovat v jednotlivých sub-týmech, ale z pohledu celého projektu nebo programu se stávají technické aspekty řízení projektu kritickými. Týká se to především řízení závislostí, sledování zdrojů, řízení smluv a řízení prodeje. Tým projektového řízení na obrázku je

složen z vedení různých sub-týmů. Jejich cílem je koordinovat aspekty řízení celého týmu. Tým by měl mít každodenní krátké koordinační schůzky.

technické/architektonické problémy – tým vlastnictví architektury je složen z vlastníků architektury sub-týmů a je odpovědný za návrh předpokládané architektury v počátcích projektu. Odpovídá za určení počátečního technického směru a poskytuje základ pro organizaci sub-týmů. V prvním týdnu od zahájení projektu (v některých případech, u složitějších projektů se jedná i o několik týdnů) je jejich úkolem návrh podsystémů a jejich rozhraní, zredukování vazeb mezi podsystémy a tím zredukování míry koordinace mezi sub-týmy. Jakmile jsou rozhraní správně definována, mohou se jednotlivé sub-týmy zaměřit na implementaci vnitřních částí těchto podsystémů. Během doby trvání projektu se tento tým účastní pravidelných schůzek za účelem sdílení myšlenek a řešení technických problémů, konkrétně těch, které souvisejí s rozhraními podsystémů. Na začátku projektu jsou obvyklé každodenní schůzky, ale po stabilizaci architektury je běžné pořádat tyto schůzky jedno či dvakrát týdně.

problémy související s požadavky a vlastnictvím projektu – tým vlastnictví projektu je tvořen vlastníky produktu všech sub-týmů a je odpovědný za koordinaci požadavků mezi sub-týmy. Tento tým by měl dojednávat požadavky s větší částí investorů, které zastupují a vhodně je rozdělovat mezi sub-týmy. Tým by měl rovněž projednávat nevyhnutelné neshody mezi sub-týmy, týkající se otázek kdo má co dělat a co daný požadavek skutečně znamená. Spravuje rovněž závislosti požadavků mezi sub-týmy a usiluje o minimalizaci překrývajících se činností mezi sub-týmy.

Integrace systému – integrace systému je důležitá pro jakoukoli velikost týmu, ale je často opravdu kritická v případě velkých týmů (které obvykle řeší složitější problémy). Složitost větších projektů obvykle vyžaduje zařazení integrátora (či několika integrátorů) systému do týmu. Integrace systému je prováděna v průběhu celého agilního životního cyklu projektu, nejen na konci projektu v průběhu fáze testování. Na začátku vývoje je pro všechny sub-týmy důležité vytvořit jakousi kostru své části systému (podsystému) podle definovaných a dohodnutých rozhraní. Cílem je vytvoření takové kostry celého systému, aby bylo zřejmé, že sub-týmy pracují na stejné technické vizi celého sytému. U velkých projektů je rovněž důležité, aby nezávislý tým určený pro testování často (po dokončení některého z podsystémů) prováděl integrační testy celého systému – což by bylo velmi obtížné pro jednotlivé sub-týmy v jejich vlastním prostředí.

Shrnutí pojmů 6.1.

Vzhledem k časové náročnosti a administrativní složitosti vývoje software podle těžkých rigorózních metodik je na místě, v případě menších projektů, použít jiný přístup. Tímto přístupem jsou tzv. agilní metodiky. Agilní metodiky se řídí manifestem, obsahujícím principy a priority agilního programování.

Alfou a omegou agilního přístupu jsou krátké přírůstkové iterace, participace zákazníka při vývoji a časté konzultace vývojového týmu se zákazníkem. U agilního přístupu je možné pružně reagovat na změny a upřesňování požadavků na systém. Důležitou součástí agilních metodik je rovněž testování hotových částí v průběhu celého projektu, většinou na konci každé iterace.

Otázky 6.1.

1. Vyjmenujte výhody agilního přístupu v porovnání s vývojem podle rigorózních metodik.
2. Co definuje manifest agilního vývoje SW?
3. Jaká je role investora projektu?
4. Vyjmenujte složení běžného týmu agilního vývoje
5. Jsou specialisté součástí projektového týmu? Pokud ano, tak za jakých okolností?
6. Jaká rizika agilní přístup eliminuje nebo potlačuje?
7. Proč je u agilního přístupu důležité testování?
8. Kdo provádí testování u menších projektů a kdo u větších?
9. Jaká je úloha integrátora?

7 METODIKY ADS, DSDM, FDD, XP: ADS (ADAPTIVE SOFTWARE DEVELOPMENT), DSDM (DYNAMIC SYSTEMS DEVELOPMENT METHOD), FDD (FEATURE-DRIVENDEVELOPMENT), CHARAKTERISTIKY, VÝHODY, PRINCIPY VÝVOJE, SROVNÁNÍ. EXTREME PROGRAMMING (XP), CHARAKTERISTIKA A VÝHODY XP.

V této kapitole se seznámíme s několika typy agilních metodik. Vysvětlíme si jejich hlavní principy, u některých uvedeme role a další souvislosti, které se těchto konkrétních metodik týkají.

Čas ke studiu: 2 hodiny

Cíl: Po prostudování této kapitoly budete umět

- Objasnit hlavní myšlenky metodik ASD, FDD, DSDM, XP a LD.
- Uvést výhody metodiky XP.
- Uvést hlavní postupy metodiky XP.
- Uvést hlavní postupy metodiky DSDM.

Výklad

7.1 ASD (Adaptive Software Development) - Adaptivní vývoj SW

Metodika ASD nahrazuje klasický postup vývoje IS „Plánování-Návrh-Realizace“ dynamickým cyklem „Spekulace-Spolupráce-Učení“. Cyklus předpokládá neustálé učení poháněné změnami. Odchylky od plánu nejsou chápány jako chyby, ale jako příležitost k učení.

Fáze spekulace zahrnuje určení termínu ukončení projektu, určení optimálního počtu iterací, termínů ukončení každé iterace, určení obsahu jednotlivých iterací, přiřazení SW komponent a technologií jednotlivým iteracím.

Ve fázi spolupráce je prováděn vývoj SW komponent. Důraz se klade (jak již plyne z principů agilních metodik) na komunikaci a interakci členů vývojového týmu.

Fáze učení slouží k hodnocení a zlepšování procesu vývoje na konci každé iterace, tj. k ohodnocení kvality vyvíjeného SW, práce týmu a stavu projektu.

Jedná se o nejdynamičtější metodiku ze souboru agilních metodik, u níž však zůstává zachován procesní přístup. Změna a přizpůsobování se změnám je hnacím motorem této metodiky. Na rozdíl od většiny ostatních agilních metodik se tato metodika nesoustřeďuje na předávání funkčního produktu na konci každé iterace (a to i přesto, že iterativní přístup agilních metodik zůstává zachován). Finálním cílem každé iterace je předání takových podkladů zákazníkovi, které umožní jeho ověření, zda je postup vývoje správný. V tomto rámci je možné předávat prototypy a návrhy rozhraní, případně beta-verze produktu. Finální, otestovaný a odladěný systém je zákazníkovi předán až po ukončení celého vývoje.

Z výše uvedeného vyplývá, že metodika ASD je vhodná v případech, kdy není z jakýchkoli důvodů možno projekt rozdělovat a dodávat po menších částech. Týká se to např. řídících a kontrolních systémů, bezpečnostních systémů, atd.).

Důležitým faktorem metodiky ASD je, že metodika neobsahuje definici konkrétních postupů – detailní obsahy procesu jsou naplňovány z jiných metodik, ať už rigorózních či agilních.

Metodika ASD se s úspěchem využívá u menších i rozsáhlých projektů.

7.2 FDD (Feature-Driven Development)

Proces vývoje v této metodě je založen na krátkých (asi dvoutýdenních) iteracích řízených vlastnostmi produktu (feature-driven). Vlastností (feature) se zde rozumí dílčí výsledek, užitečný z pohledu zákazníka, měřitelný a realizovatelný v rámci dvoutýdenní iterace. Vlastnosti (features) jsou v metodice FDD přesně definovány. FDD tedy vede vývojáře k vytváření fungujících přírůstků každé dva týdny. FDD doporučuje definovat tzv. „lehké“ procesy, každý proces je stručně popsán ve struktuře:

- vstupní kritéria pro proces
- úkoly (název, účastníci, povinnosti, popis úkolu)
- nástroje verifikace
- výstupní podmínky procesu – hmatatelné výstupy

Ve všech procesech se zaznamenávají alternativní řešení. Posloupnost procesů je následující:

- vypracování celkového modelu vyvíjeného IS
- vytvoření detailního seznamu vlastností IS s prioritami jejich řešení
- plánování vývoje pro každou vlastnost, každé skupině (třídě) vlastnosti je přiřazen vlastník, datum ukončení
- návrh vlastností, tj. kontaktování vývojářů realizujících danou vlastnost SW a zpracování sekvenčního diagramu řešení dané vlastnosti
- realizace vlastnosti

Mezi agilními metodikami je FDD chápána jako konvenční a konzervativní, protože se více než ostatní agilní metodiky blíží konvenčním přístupům. Důraz je kladen na modelování a definici pojmů.

Na rozdíl od jiných agilních metodik FDD určuje přesný termín dokončení vývoje – mezi její činnosti tedy nepatří dynamické a flexibilní plány vývoje (které jsou např. součástí metodiky SCRUM).

FDD definuje tzv. vlastnictví tříd, čímž konkrétnímu vývojáři (programátorovi) přiděluje odpovědnost za danou třídu.

Díky dynamickému skládání týmů metodika FDD umožňuje využití při velkých projektech, obecně větších než je tomu u ostatních agilních metodik.

7.3 XP (Extreme Programming) - Extrémní programování

je metodika vhodná pro malé až středně velké vývojové týmy, které se musí vyrovnat se zadáním, jež se rychle mění nebo není jasné. Metodika XP se opírá o následující principy:

- využívá včasnou, konkrétní a nepřetržitou zpětnou vazbu vyplývající z krátkých iteračních cyklů vývoje IS
- přírůstkový přístup k plánování počítá s tím, že plán se může v průběhu projektu dále vyvíjet a měnit
- využívání automatizovaných testů, na jejichž sestavení se podíleli programátoři i zákazníci
- důraz je kladen na verbální komunikaci vývojového týmu se zákazníkem
- návrh systému je evolučním procesem probíhajícím neustále po celou dobu existence systému
- úzká spolupráce programátorů (programování v párech)

Metodika XP je dobře propracovanou metodikou. Je velmi striktní, pokud jde o dodržování pravidel. Při vývoji software by měla být implementována celá, což je v častých případech velmi složité, či téměř nemožné. I přesto je metodika velmi rozšířená, díky čemuž existuje mnoho knižních i webových zdrojů a diskuzních fór, dokonce i v češtině.

Metodika XP získala svůj název díky až extrémnímu využívání činností osvědčených i v jiných metodách.

Metodika Extreme Programming je postavena na dvanácti postupech, které by měly být přísně dodržovány:

- **Plánovací hra** – tvorba plánu, na které se podílí všichni členové týmu.
- **Malé verze** – co nejčastější uvolňování nových verzí.
- **Metafora** – je náhradou za termín „architektura“. Vývoj je popisován příběhem o tom, jak má konečný systém fungovat.
- **Jednoduchý návrh** – dodržování maximální jednoduchosti systému.
- **Testování** – u metodiky XP se testování provádí velmi často. Testy jsou tvořeny ještě před vlastní implementací funkcí a pro pokračování je nutné, aby testování proběhlo s pozitivním výsledkem.
- **Refraktorizace** – při tomto postupu je zdrojový kód maximálně optimalizován.
- **Párové programování** – veškeré programování je prováděno dvěma programátory na jednom počítači.
- **Společné vlastnictví** – tato metodika se od většiny ostatních liší tím, že změny zdrojového kódu mohou provádět všichni programátoři. U mnoha jiných (objektových) metodik mají třídy své vlastníky, kteří za ně odpovídají.
- **Nepřetržitá integrace** – celý systém se v průběhu dne vícekrát sestavuje a testuje.
- **40hodinový pracovní týden** – zamezuje pracovnímu vyčerpání a přepínání programátorů. Metodika propaguje filozofii, že odpočatý a spokojený programátor podává kvalitnější pracovní výkony a jeho práce je efektivnější.
- **Zákazník na pracovišti** – podobně jako je tomu u jiných agilních metodik, i zde je zákazník součástí týmu. Vzhledem k extrémnímu přístupu metodiky XP se přísně dbá na jeho faktickou přítomnost.
- **Standardy pro psaní zdrojového kódu** – vzhledem k tomu, že se u metodiky XP nevytváří dokumentace, je nezbytné, aby byl zdrojový kód psán podle daných konvencí

a dobře okomentovaný. Dodržování tohoto postupu usnadňuje programátorům rychlou orientaci v i kódu, jehož nejsou autory.

7.4 DSDM - Dynamic Systems Development Method

Vývojem této metodiky se společně zabývalo šestnáct různých organizací. Kladem této metodiky je i dodávka frameworku – vývojového prostředí. Metodika se vyznačuje vysokou propracovaností; její vývoj usilovně pokračuje již přes dvacet let. Během této doby byly do metodiky neustále přidávány zkušenosti z vývoje a moderní tendence. Součástí metodiky DSDM je i série dobře propracovaných technik, včetně doporučení kdy a za jakých podmínek je používat a k jakému účelu jsou určeny.

Stejně jako u ostatních agilních metodik i u DSDM probíhá vývoj na základě iterací, a to nejen z hlediska hlavních fází, ale i uvnitř těchto fází. U hlavních fází platí, že je možné se v nich vracet, což znamená, že vývoj a ostatní činnosti jsou vždy mířeny do (v daném okamžiku) klíčových míst.

Jednou z „negativ“ této metodiky je, že pro její legální používání je nezbytné složit šestnáctičlenné organizaci členský poplatek. Na druhou stranu je výhodou již zmíněný podpůrný framework, který je již zahrnut do ceny členského poplatku. Členský poplatek činí částku od pár desítek GBP po tisíce GBP. Částka je rozlišovaná druhem členství, např. akademické, firemní, vládní, atd.

Metodika DSDM se soustřeďuje na následujících osm principů (vycházejících z manifestu agilního vývoje):

- zaměření na potřeby podniku
- včasná dodávka
- spolupráce
- nikdy neohrožovat kvalitu
- stavět přírůstkově na pevných základech
- vyvíjet iterativně
- komunikovat neustále a jasně
- demonstrovat (předvádět) ovládání

Jak bylo výše uvedeno, metodika DSDM obsahuje sérii propracovaných technik. Jádro těchto technik představují následující:

- **timeboxing** – je jednou z projektových technik DSDM určenou k podpoře hlavních cílů DSDM. Technika se soustřeďuje na včasnou realizaci vývoje, v rámci daného rozpočtu a v požadované kvalitě. Hlavní myšlenkou timeboxingu je rozdělení projektu do částí, z niivchž každá má stanovený rozpočet a dodací lhůtu. U každé části jsou rovněž stanoveny požadavky, které mají vyšší prioritu – v souladu s principem MoSCoW (viz dále). Vzhledem ke skutečnosti, že čas a rozpočet jsou pevně stanoveny, proměnnou hodnotou jsou pouze požadavky.
- **MoSCoW** – reprezentuje způsob (technika) upřednostnění položek. Jedná se o akronym tvořený následujícími slovy (angličtina):
    - MUST have - MUSÍ mít
    - SHOULD have – MĚL BY mít
    - COULD have – MOHL BY mít
    - WON¨T have this time – nyní NESMÍ mít.
- **prototypování** – tato technika odkazuje na tvorbu prototypů vyvíjeného systému v prvních fázích projektu. Umožňuje včasné odhalení nedostatků a chyb v systému.
- **testování** – jedním z důležitých aspektů vývoje pomocí metodiky DSDM je tvorba a dodání softwaru v dobré kvalitě. Aby mohlo být tohoto cíle dosaženo, metodika DSDM preferuje testování v průběhu každé iterace. Vzhledem k tomu, že DSDM je metodikou nezávislou na konkrétních nástrojích a technikách, je na rozhodnutí projektového týmu, jaké nástroje a techniky testování zvolí.
- **workshop** – jedná se o techniku metodiky DSDM využívanou pro přizvání různých investorů k diskuzi o požadavcích, funkčnosti a k vzájemné dohodě. Na workshopu se investoři schází a diskutují o projektu.
- **modelování** – tato základní technika se u DSDM, podobně jako v jiných metodikách, využívá k vizualizaci specifických aspektů projektu nebo podnikových oblastí prostřednictvím diagramů. Modelování poskytuje projektovému týmu dokonalejší porozumění jednotlivým částem systému.
- **správa konfigurace** – dobrá implementace techniky správy konfigurace je z hlediska dynamické povahy metodiky DSDM důležitá. Vzhledem ke skutečnosti, že je v průběhu vývoje projektu zpracováváno více úloh současně a produkty jsou dodávány velmi často, musí být tyto produkty (nebo jejich části) velmi často přísně kontrolovány.

V prostředí metodiky DSDM jsou rovněž zavedeny role, kterých je celkem patnáct. Podobně, jako je tomu v případě jiných metodik, i zde mají role přiřazeny své odpovědnosti. Před započetím projektu je důležité členům týmu tyto role přiřadit (resp. přiřadit členy týmu do určitých rolí). V DSDM se jedná o následující role (uvedeny jsou ty nejvýznamnější z patnácti celkových):

- **výkonný ředitel** – důležitá role vycházející z organizace zákazníka, která má ve svých kompetencích přidělování zdrojů a kapitálu.
- **vizionář** – odpovědností této role je inicializace projektu po zajištění základních požadavků. Vizionář má nejpřesnější informace o cílech projektu a zákazníka. Dalším úkolem osoby v této roli je dohled nad správností směru vývoje projektu.
- **zástupce uživatelů** – do projektu přináší názory a znalosti komunity uživatelů a zajišťuje, aby vývojáři měli od uživatelů dostatečnou zpětnou vazbu.
- **rádce** – může být jakýkoli uživatel (zástupce uživatelů), který přináší do projektu důležitá stanoviska a zajišťuje komunikaci požadavků a názorů uživatelů členům vývojového týmu. Má dokonalou znalost projektu (včetně každodenních změn) a s projektem seznamuje ostatní uživatele.
- **projektový manažer** – role obecného vedení projektu. Zvláštností u metodiky DSDM je, že tato role může být obsazena jak členem vývojového týmu, tak i osobou ze strany organizace, pro kterou je projekt vytvářen.
- **technický koordinátor** – odpovědný za sestavení architektury systému a řízení technické kvality projektu.
- **vedoucí týmu** – vede tým a stará se o jeho efektivní práci (týmu jako celku).
- **vývojář řešení** – vytváří prototypy a dodávaný programový kód podle požadavků projektu, které interpretuje.
- **tester řešení** – testuje správnost (v technickém rámci) prováděním testování, odhaluje chyby, řeší je a opětovně testuje. Tester komentuje a vytváří část dokumentace.
- **zapisovatel** – odpovědný za získávání a zaznamenávání požadavků, dohod a rozhodnutí přijatých na každém workshopu.
- **moderátor** – stará se o průběh workshopu, pomáhá s přípravami a komunikací
- **role specialistů** – různé speciální role, jako manažer jakosti, systémový integrátor, doménový expert, atd.

Podle velikosti projektu se na vývoji podílí jeden tým nebo více týmů. Pokud jde o řízení projektu, nedělá se rozdíl mezi menším a větším projektem – větší projekty je možné rozdělit na několik nezávislých částí, které jsou řešeny jednotlivými týmy samostatně. Týmy jsou tvořeny dvěma až šesti osobami. Minimum dvou osob vychází z předpokladu, že členy týmu jsou minimálně vývojář a uživatel (jedná se o agilní metodiku). Maximální počet šesti členů týmu je dán na základě dlouhodobých zkušeností s používáním metodiky DSDM.

7.5 Lean Development

Další agilní metodikou, kterou je na místě zmínit (jedná se rovněž o velmi populární metodiku) je metodika Lean Development, zkráceně LD. Metodika se zaměřuje na vytváření změnám přizpůsobivého SW. Tato metodologie ztělesňuje představu dynamické stability, o které lze smýšlet v podobném duchu jako o Scrumu, který představuje kontrolovaný chaos. 12 principů Lean Developmentu:

1. Uspokojit zákazníka je nejvyšší priorita.
2. Vždy poskytovat nejvyšší kvalitu za peníze.
3. Úspěch závisí na aktivní účasti zákazníka.
4. Každý LD projekt je týmová práce.
5. Všechno se může změnit.
6. Oblastní, ne bodová řešení.
7. Nevytvářet, ale kompletovat.
8. 80 procentní řešení dnes než 100 procentní řešení zítra.
9. Minimalismus je podstatný.
10. Potřeby určují technologii.
11. Růst produktu znamená růst jeho vlastností ne velikosti.

Nikdy se nesnažte překročit omezení LD.

Shrnutí pojmů 7.1.

V této kapitole jsme si objasnili základní vlastnosti metodik ASD, FDD, XP, DSDM a Lean Development. Uvedli jsme si vhodnost použití jednotlivých metodik. U metodiky XP jsme popsali i dvanáct hlavních postupů, které je nezbytné přísně dodržovat:

- Plánovací hra
- Malé verze – co nejčastější uvolňování nových verzí.
- Metafora
- Jednoduchý návrh – dodržování maximální jednoduchosti systému.
- Testování
- Refraktorizace
- Párové programování
- Společné vlastnictví
- Nepřetržitá integrace
- 40hodinový pracovní týden
- Zákazník na pracovišti
- Standardy pro psaní zdrojového kódu

U metodiky DSDM jsme uvedli osm základních principů:

- zaměření na potřeby podniku
- včasná dodávka
- spolupráce
- nikdy neohrožovat kvalitu
- stavět přírůstkově na pevných základech
- vyvíjet iterativně
- komunikovat neustále a jasně
- demonstrovat (předvádět) ovládání

Objasnili jsme rovněž sérii propracovaných technik metodiky DSDM (timeboxing, MoSCoW, prototypování, testování, atd.) a uvedli si základní role v této metodice.

Otázky 7.1.

1. Objasněte hlavní principy metodiky ASD.
2. Jak rozumíte termínu „feature-driven“?
3. Proč je metodika XP nazývána jako extrémní?
4. Objasněte pojem „párové programování“ v metodice XP.
5. Vyjmenujte osm hlavních principů metodiky DSDM.

6. Kdo může být v metodice DSDM v roli rádce?
7. Za co odpovídá osoba v roli rádce v metodice DSDM?
8. Definujte hlavní myšlenky metodiky Lean Development

8 METODIKY ADS, DSDM, FDD, XP: ADS (ADAPTIVE SOFTWARE DEVELOPMENT), DSDM (DYNAMIC SYSTEMS DEVELOPMENT METHOD), FDD (FEATURE-DRIVENDEVELOPMENT), CHARAKTERISTIKY, VÝHODY, PRINCIPY VÝVOJE, SROVNÁNÍ. EXTREME PROGRAMMING (XP), CHARAKTERISTIKA A VÝHODY XP.

V této kapitole se dozvíme podrobnosti o metodikách Scrum a rodině metodik Crystal. Uvedeme hlavní role a postup práce v metodice Scrum a způsob využití metodik Crystal.

Čas ke studiu: 2 hodiny

Cíl: Po prostudování této kapitoly budete umět

- Objasnit hlavní myšlenky metodiky Scrum.
- Uvést role metodiky Scrum a odpovědnosti těchto rolí.
- Uvést hlavní principy rodiny metodik Crystal.
- Vyjmenovat jednotlivé metodiky rodiny Crystal.

Výklad

8.1 SCRUM

Scrum je iterativní inkrementální framework pro řízení komplexních prací (jako je vývoj nového produktu). Je navrhnutý pro týmy o třech až devíti pracovnících, které si rozdělí svou práci do činností, které mohou být dokončeny během cyklů s pevnou dobou trvání (tzv. sprinty). Týmy sledují vývoj a provádějí přizpůsobení plánů na denních 15minutových schůzkách, které se vyznačují tím, že probíhají ve stoje (tzv. stand-up meeting). Funkční části software jsou dodávány na konci každého sprintu.

Klíčovým principem metodiky Scrum je dvojí poznání:

- zákazníci mění své pohledy na to, co potřebují nebo chtějí

- vzniknou nepředvídatelné situace, pro které není vhodný předpokládaný nebo plánovaný přístup.

Autoři metodiky Scrum (Hirotaka Takeuchi a Ikujiro Nonaka) představili v roce 1986 tuto metodiku jako „tvorbu organizačních znalostí, která je vhodná především pro přínos inovace nepřetržitě, přírůstkově a spirálově“. Autoři popsali nový přístup k vývoji komerčního produktu, který by měl zvýšit rychlost a flexibilitu. Tento přístup byl vytvořen na základě případových studií z automobilového průmyslu, průmyslu zabývajícího se fotokopiemi a tiskařského průmyslu. Svůj přístup autoři nazývali holistickým nebo ragbyovým přístupem, ve kterém je celý vývoj prováděn jedním multifunkčním týmem v několika překrývajících se fázích; přístup, kde se tým „pokouší dostat dopředu jako celá jednotka, která si předává míč dozadu i dopředu“ (v překladu slovo „scrum“ znamená „mlýn“, používaný termín z ragby).

8.1.1 Role v metodice Scrum

Ve frameworku Scrum existují tři hlavní role. Všechny tyto role dohromady tvoří tým Scrum („tým mlýnu“). Názvy rolí (a ostatní termíny spojené s frameworkem Scrum se podle jeho konvence píší kapitálkami na počátku každého podstatného jména. Můžeme se rovněž setkat s tzv. „velbloudím stylem“, viz výše). Jedná se o následující:

- Vlastník Produktu (Produkt Owner)
- Vývojový Tým (Development team)
- Scrum master (v doslovném překladu „Mistr Mlýnu“ nebo Mlynář…v češtině však tento překlad v souvislosti s metodikami nevyznívá příliš dobře, budeme se proto v dále textu držet původního anglického označení Scrum master).

Vlastník produktu – reprezentuje investory a hlas zákazníka. Odpovídá za to, aby tým do obchodu přinášel hodnotu. Vlastník produktu definuje produkt v termínech vlastních zákazníkovi (typicky se jedná o uživatelské příběhy – vyjádření požadavků a vlastností či funkcí systému v přirozeném jazyce), přidává je do tzv. backlogu (seřazený seznam požadavků. Tvoří jej vlastnosti, opravy chyb, jiné než funkční požadavky, atd.) a přiřazuje jim prioritu, na základě důležitosti a závislostí. V týmu Scrum v roli vlastníka produktu měla být jedna osoba, která by neměla být kombinována s rolí mistra. Role vlastníka produktu je ekvivalentem role zástupce zákazníka známé z metodiky XP.

Vlastník produktu má v popisu svých úkolů následující komunikační záležitosti:

- předvádí řešení klíčovým investorům, kteří nebyli přítomni na hodnocení sprintu.

- definuje a oznamuje uvolnění funkčních verzí
- informuje o stavu týmu
- organizuje revize při dosažení milníků
- zaškoluje investory v procesu vývoje
- dojednává priority, rozsahy, časové plány, finanční fondy
- zajišťuje, že je backlog produktu dostupný, transparentní a jasný

Klíčovým povahovým rysem vlastníka produktu je empatie. Musí umět porozumět různým osobám v rolích investorů, které mají různé cíle, zkušenosti, pracovní role… Vlastník projektu se musí být schopen dívat z těchto různých úhlů pohledu. Musí být schopen rovněž „filtrovat“ informace, které předává, podle aktuálního stavu projektu, osoby se kterou komunikuje, atd. Příliš více informací, než je nutné, může v některých případech vést např. ke ztrátě investora.

Schopnost vlastníka produktu komunikovat efektivně je rozšířena také jeho znalostmi v oblastech umožňujících identifikovat potřeby investora, vyjednat priority mezi zájmy investora a spolupracovat s vývojáři, aby byla zajištěna efektivní implementace požadavků.

Vývojový tým – úkolem vývojového týmu je dodávat tzv. PSI – Potentially Shippable Products (potenciálně použitelný přírůstků) produktu v závěru každého sprintu. Jedná se vlastně o cíl sprintu. Team je tvořen třemi až devíti členy, kteří plní aktuální úkoly – design, vývoj, analýza, testování, dokumentace, technická dokumentace, atd. Vývojové týmy mají více funkcí, mají veškeré dovednosti, aby mohly vytvořit produktový přírůstek. Vývojový tým je sebeorganizační, avšak určité formy projektového managementu se mohou využívat i zde.

Scrum master – nejedná se o klasického vedoucího nebo manažera týmu, ale spíše o prostředníka mezi týmem a možnými negativními okolnostmi či vlivy. Stará se o dodržování dohodnutých procesů a plánované dodržování procesu scrum.

Mimo jiné, odpovědnost Scrum mastera zahrnuje zejména:

- pomoc vlastníkovi produktu udržovat backlog produktu způsobem, umožňujícím správné pochopení požadovaným úkolům tak, aby tým mohl při vývoji neustále postupovat vpřed
- pomoc týmu určovat definici hotových částí produktu, se vstupem klíčových investorů

- koučing týmu v rámci principů metodiky Scrum s cílem dodat vlastnosti produktu ve vysoké kvalitě
- podporu samoorganizace v rámci týmu
- pomoc týmu vyhnout se nebo odstranit překážky bránící v postupu
- usnadnění týmových schůzek pro zajištění pravidelného postupu
- školení klíčových investorů v záležitostech týkajících se produktu – podle zásad Scrum.

Rozdíl mezi projektovým manažerem a Scrum master je ten, že projektový manažer může mít odpovědnost také za lidské zdroje.

8.1.2 Průběh práce v metodice Scrum

Sprint – je v metodice Scrum název iterace. Je základní jednotkou vývoje. Podléhá časovému omezení, které je předem pro každý sprint pevně stanoveno; obvykle se jedná o jeden týden až jeden měsíc. Nejobvyklejší jsou dva týdny.

Každý sprint začíná plánovací schůzkou s cílem určit backlog sprintu, práci v daném sprintu a předběžný plán sprintu. Každý sprint končí revizí a retrospektivou sprintu, ve kterých probíhá hodnocení pokroku, které bude předloženo investorům a návrh zlepšení pro následující sprinty. V případě vývoje SW se na konci sprintu předpokládá funkční, integrovaný, otestovaný a zdokumentovaný produkt, který je možno dodat.

Plánování sprintu – na začátku sprintu team uspořádá plánovací schůzku určenou k:

- vzájemnému prodiskutování a odsouhlasení rozsahu prací pro daný sprint
- výběru položek backlogu produktu, které budou dokončeny v jednom sprintu
- přípravě backlogu sprintu obsahujícího práci, kterou je nutno vykonat pro dokončení vybraných položek backlogu produktu
- doporučené trvání je čtyři hodiny pro dvoutýdenní sprint
    - během první poloviny celý tým scrum (vývojový tým, scrum master, vlastník produktu) vybere položky produktového backlogu, u kterých předpokládají, že budou dokončeny během daného sprintu

    - během druhé poloviny určí tým podrobné úkoly, které je nezbytné splnit pro dokončení těchto položek backlogu produktu – je vytvořen schválený backlog sprintu.
        - během plnění vytýčených úkolů mohou být některé položky backlogu rozděleny nebo vráceny zpět do produktového backlogu, pokud tým nepředpokládá jejich dokončení v průběhu daného (jednoho) sprintu
- jakmile tým připravil backlog sprintu, může předběžně určit (většinou na základě hlasování), které úkoly budou během daného sprintu dodány

Denní scrum – každý den během sprintu tým uspořádá denní scrum, tj. denní rychlý meeting, probíhající většinou ve stoje, dle následujících pravidel:

- všichni členové vývojového týmu přijdou připraveni. Denní scrum:
    - začíná přesně v určeném čase a to i v případě nepřítomnosti některých členů týmu
    - každý den by měl probíhat ve stejném čase a na stejném místě
    - je časově omezen na patnáct minut
- vítán je kdokoli, avšak přispívat mohou pouze členové vývojového týmu
- během denního scrumu obvykle každý člen týmu odpovídá na tři otázky:
    1. Jaký úkol, který může přispět týmu ke splnění cíle sprintu, jsem včera dokončil?
    2. Jaký úkol, který může přispět týmu ke splnění cíle sprintu, plánuji dokončit dnes?
    3. Vím o nějakých překážkách, které by mohly mně osobně anebo týmu zabránit ve splnění cílů sprintu?

Jakákoli překážka (riziko, problém, opožděná závislost, atd.) identifikovaná na denním scrumu by měla být zaznamenána scrum masterem a vyvěšena (zobrazena) na týmové scrum tabuli nebo na tabuli sdílených rizik. Zároveň by měla být určena osoba pracující na vyřešení (odstranění) překážky (mimo denní scrum). Během denního scrumu by neměly probíhat žádné diskuze o podrobnostech.

Revize a retrospektiva sprintu – na konci sprintu tým uspořádá dvě schůzky – revizi sprintu a retrospektivu sprintu.

Na **revizi** sprintu tým:

- reviduje dokončenou práci a plánovanou práci, která nebyla dokončena
- prezentuje dokončenou práci investorům
- tým a investoři spolupracují na dalším postupu (co se bude dělat dále)

Revize sprintu se řídí následujícími pravidly:

- nekompletní nebo nedokončená práce nemůže být prezentována (demonstrována)
- doporučená doba trvání je dvě hodiny v případě dvoutýdenního sprintu

Při **retrospektivě** sprintu tým:

- uvažuje o minulém sprintu
- určuje, probírá činnosti pro neustálé vylepšování procesu, na kterých se dohodne

Retrospektiva sprintu se řídí následujícími pravidly:

- jsou položeny dvě zásadní otázky:
  1. Co šlo během sprintu dobře?
  2. Co by mohlo být během příštího sprintu vylepšeno?
- doporučená doba trvání schůzky je hodina a půl u dvoutýdenního sprintu
- na schůzce je nápomocen scrum master

Rozšíření činností – následující činnosti se obecně provádějí, ačkoli nejsou považovány za hlavní části metodiky Scrum:

Upřesňování backlogu – je trvalým procesem revidování položek produktového backlogu společně s kontrolou, zda jsou tyto položky vhodně připraveny a zda je jim přiřazena správná priorita, a to takovým způsobem, aby byly pro týmy během vstupu do sprintu jasné a proveditelné (prostřednictvím činnosti plánování sprintu). Položky produktového backlogu mohou být rozděleny do několika menších, musí být ujasněna akceptační kritéria. Rovněž musí být identifikovány závislosti, výzkumné a přípravné práce.

I když se nejedná o jednu ze základních praktik metodiky Scrum, upřesňování backlogu bylo přidáno do pravidel (příručky) metodiky Scrum a bylo přijato jako způsob řízení kvality položek produktového backlogu vstupujících do sprintu.

Zrušení sprintu – vlastník produktu může v případě potřeby sprint zrušit. Může tak učinit na základě informací od týmu, scrum mastera nebo managementu. Management může požadovat zrušení sprintu například v případě, že vnější okolnosti negují hodnotu cíle sprintu.

Je-li sprint abnormálně ukončen, pak dalším krokem je provedení nového plánování sprintu, kde je zrevidován důvod ukončení sprintu.

8.2 Crystal metodiky

Myšlenka Crystal metodik vychází z toho, že jedna metodika nemůže vyhovovat všem projektům a vývojovým týmům. Proto Alistair Cockburn definoval techniku vytváření metodiky „just in time“. Vychází přitom z trojrozměrné matice (crystal) charakteristik projektu. Dimenze matice tvoří:

- počet lidí zúčastněných na projektu
- míra důležitosti vyvíjeného systému pro zákazníka
- priority projektu

Podle úrovně těchto základních faktů je vybrána metodika, která se dále přizpůsobí a vyladí podle potřeby konkrétního projektu.

Metodiky Crystal jsou považovány za tzv. „lehké metodiky“. U Crystal metodik rozlišujeme reprezentace technik, nástrojů, norem a rolí.

Metodiky Crystal se zaměřují na:

- osoby
- interakci
- komunitu
- zkušenosti
- talenty
- komunikace

Cockburn tvrdí, že na proces, i když je důležitý, bychom se měli zaměřovat až ve druhé fázi. V první fázi bychom se měli zaměřit především na výše uvedené body. Myšlenkou stojící za metodikami Crystal je, že týmy zapojené v procesu vývoje software budou mít různé schopnosti, zkušenosti a budou různě talentované, a proto prvek procesu není hlavním faktorem. Protože týmy mohou řešit podobné úkoly rozdílnými způsoby, je rodina metodik Crystal v tomto ohledu velmi tolerantní, což z ní činí jednu z nejsnadnějších agilních metodik.

V průzkumu, který Cockburn provedl (1999) definuje chování lidí v týmech:

- „Lidé jsou komunikující bytosti, nejlépe komunikující tváří v tvář, osobně, s otázkami a odpověďmi v reálném čase.“

- „Lidé mají problém jednat během času pořád důsledně.“
- „Lidé jsou velmi proměnliví, mění se ze dne na den, z místa na místo.“
- „Lidé obecně chtějí být dobrými občany, umí převzít iniciativu a udělat vše možné pro úspěšné zvládnutí projektu.“

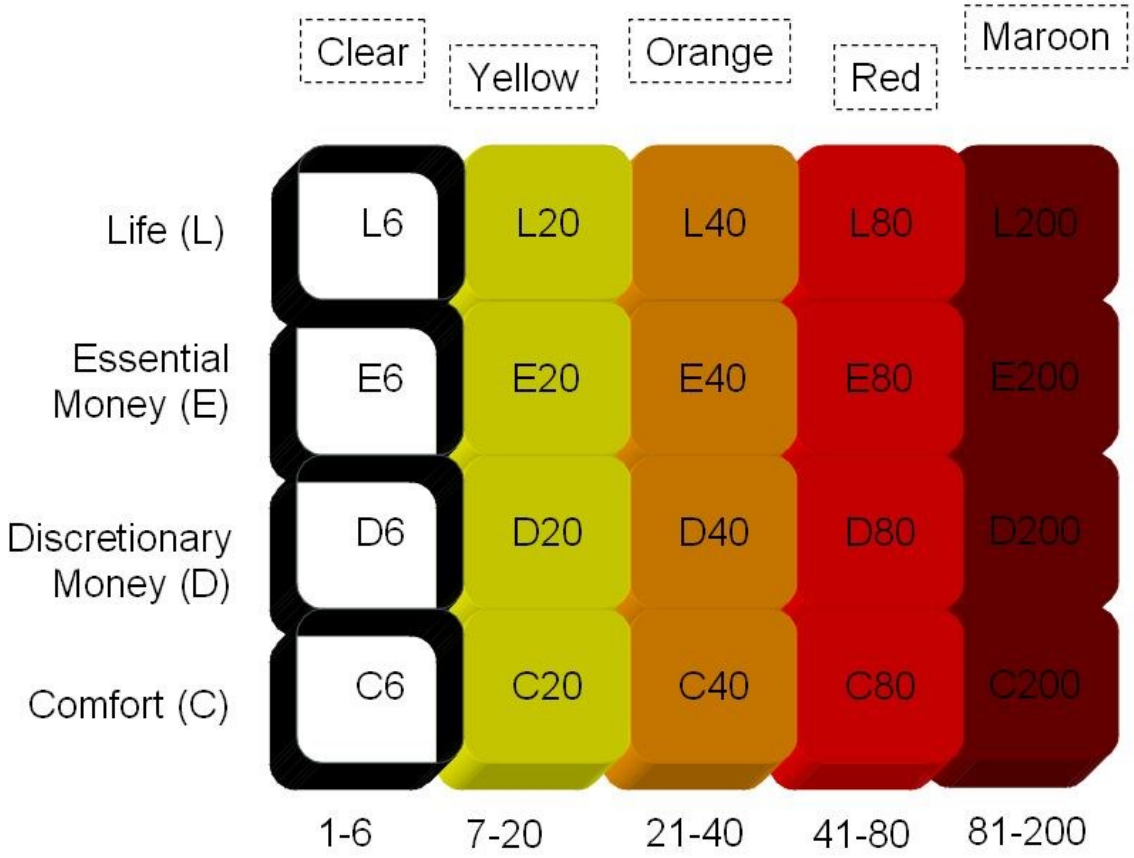
Výše uvedené body jsou důvodem, proč jsou metodiky Crystal tak flexibilní a proč se vyhýbají striktním a rigidním procesům, které jsou často využívány v jiných metodikách.

Cuckburn vyvinul různé metody v rodině metodik Crystal, které jsou vhodné pro týmy různých velikostí, které vyžadují rozdílné strategie k řešení rozličných problémů.

Rodina metodik Crystal využívá k označení „váhy“ metodik různé barvy. V případě menších projektů je možné použít metodiky Crystal Clear, Crystal Yellow nebo Crystal Orange. V případě kritických projektů, při kterých může být v ohrožení lidský život, by měly být použity metodiky Crystal Diamond nebo Sapphire.

Několik příkladů rozdělení rodiny Crystal podle barev:

1. Crystal Clear
2. Crystal Yellow
3. Crystal Orange
4. Crystal Orange Web
5. Crystal Red
6. Crystal Maroon
7. Crystal Diamond
8. Crystal Sapphire



Obrázek 8-1 Projekty rodiny metodiky Crystal. Zdroj: A Practical Guide to Seven Agile Methodologies Part 2, http://www.devx.com/architect/Article/32836/0/page/2

Podle obrázku je patrné, že větší projekty mají tmavší barvu.

Mezi všemi metodikami rodiny Crystal je sedm sdílených, obecných a převládajících vlastností. Cockburn zjistil, že čím více těchto vlastností je v projektu, tím větší je pravděpodobnost úspěchu. Vlastnostmi, o kterých je řeč, jsou:

1. Časté dodávky
2. Reflektivní vylepšování
3. Blízká komunikace nebo komunikace „na pozadí“ (osmotic communication)
4. Osobní bezpečnost
5. Zaměření
6. Snadný přístup k expertům
7. Technické prostředí s automatizovanými testy, řízením konfigurace a častou integrací

V následujícím textu se podíváme na uvedené body podrobněji.

Časté dodávky - znamenají časté uvolňování iterací softwaru. Myšlenka vychází z podstaty agilních metodik. Designéři a vývojáři rozhodují, které části softwaru, jeho vlastností a funkcí uvolní po každé iteraci. V případě metodik Crystal se jedná o iterace v trvání jednoho týdne až jednoho čtvrtletí.

Reflektivní vylepšování – obnáší přerušení práce na vývoji a snahu nalézt lepší řešení procesů. Reflektivní vylepšování jsou usnadněna iteracemi – iterace poskytují zpětnou vazbu o tom, zda je daný proces funkční či nikoli. U metodik Crystal je doporučeno pořádání tzv. „reflektivních workshopů" jednou za několik týdnů. Tyto workshopy pomáhají s nalezením procesů, které nepracují dobře a jsou týmu nápomocny při jejich modifikaci tak, aby mohla být vyvinuta strategie, která pro daný tým dobře funguje.

Blízká nebo osmotická komunikace – je využívána v metodice Crystal Clear a dalších „menších" metodikách rodiny Crystal. Osmotická komunikace znamená, že tým pracuje v jedné místnosti a jeho členové tak mají možnost komunikovat a řešit problémy spontánně, během celého dne, na rozdíl od diskuzí, které se dějí pouze na schůzkách. Tento přístup je velice výhodný, neboť umožňuje rychlé předávání informací mezi členy týmu, rychlé zodpovězení otázek, informovanost členů týmu o tom, co se právě děje a možnost rychle se zbavit mylných představ. Poslouchání komunikace mezi ostatními členy týmu má vývojář povědomí o tom, co ostatní dělají, může takto získávat zkušenosti a rozvíjet nové myšlenky.

Osobní bezpečnost - schopnost členů týmu důvěřovat jeden druhému a svobodně vyjadřovat své myšlenky a problémy, kdykoli se vyskytnou. Pokud se člověk necítí v bezpečí např. při hovoru před skupinou lidí, nebo byl-li v minulosti za svůj názor, myšlenku, nápad ve skupině lidí např. zesměšněn, jen stěží se o jejich prezentaci pokusí příště, což je v týmové práci kontraproduktivní.

Zaměření – v metodikách Crystal se zaměření týká dvou věcí – za prvé je to zaměření se na jednotlivou úlohu v projektu dostatečně dlouho na to, aby bylo dosaženo požadovaného pokroku, a za druhé jde o směr, kterým se projekt ubírá. U prvního z uvedených zaměření jde o pokrok v souvislosti s problémy, které by jej mohly ovlivnit, jak např. meetingy, dlouhé otázky, telefonní hovor, atd. Tyto činnosti narušují proces vývoje a může nějakou dobu trvat, než je v procesu pokračováno. Tato zdržení zpožďují dokončení aktuální úlohy. Crystal definuje dvě pravidla pro záležitosti, které mohou přerušit zaměření se na daný úkol. Jedním jsou dvouhodinové časové úseky, ve kterých by vývojář neměl práci nijak přerušovat. Druhým pravidlem je přiřazení vývojáře k projektu alespoň dva dny předtím, než je přiřazen k jinému projektu.

U druhého významu zaměření jsou diskutovány takové problémy, jako např. definice cílů. Definice by měly být jasné a vývojáři by měli přesně vědět, co cíle projektu zahrnují.

Snadný přístup k expertům - tento bod obnáší práci vývojářů se specialistou tak, aby tento specialista mohl odpovídat na otázky, navrhovat řešení problémů, atd. Expert by měl být skutečný uživatel a nikoli pouze tester jako člen vývojového týmu.

Technické prostředí s automatizovanými testy, managementem konfigurace a časté integrace – hlavní myšlenkou by měly být neustálá integrace a testování tak, aby mohly být odhaleny chyby, poškození, atd. v případě provedení změny. Protože se uvedené provádí pravidelně, problémy by neměly narůstat, protože mohou být odstraněny/vyřešeny dříve během projektu.

Shrnutí pojmů 8.1.

V této kapitole jsme se zabývali metodikami SCRUM a rodinou metodik Crystal. Uvedli jsme hlavní role frameworku Scrum:

- Vlastník Produktu (Produkt Owner)
- Vývojový Tým (Development team)
- Scrum master

včetně popisu těchto rolí. U metodiky Scrum jsme rovněž popsali průběh práce a pojmy

- Sprint
- Plánování sprintu
- Denní scrum
- Revize a retrospektiva sprintu

a rozšiřující činnosti:

- *Upřesňování backlogu*
- *Zrušení sprintu*

U rodiny metodik Crystal jsme si vysvětlili, z čeho její tvůrce Cockburn vycházel a na co se metodiky Crystal zaměřují. Jde o

- osoby

- interakci
- komunitu
- zkušenosti
- talenty
- komunikace

Zmíněny byly i některé z metodik patřících do rodiny Crystal:

1. Crystal Clear
2. Crystal Yellow
3. Crystal Orange
4. Crystal Orange Web
5. Crystal Red
6. Crystal Maroon
7. Crystal Diamond
8. Crystal Sapphire

V této souvislosti jsme si uvedli i systém jejich barevného značení a uvedli sedm jejich základních společných vlastností:

1. Časté dodávky
2. Reflektivní vylepšování
3. Blízká komunikace nebo komunikace „na pozadí“ (osmotic communication)
4. Osobní bezpečnost
5. Zaměření
6. Snadný přístup k expertům
7. Technické prostředí s automatizovanými testy, řízením konfigurace a častou integrací

Otázky 8.1.

1. Vysvětlete principy metodiky Scrum.
2. Jaké znáte hlavní role metodiky Scrum?
3. Objasněte pojem „sprint“ v metodice Scrum.
4. K čemu slouží tzv. „denní scrum“?
5. Co je hlavním přínosem metodik rodiny Crystal?

6. Objasněte, k čemu slouží „osmotická komunikace“.
7. Čím osmotická komunikace přispívá při vývoji software?

9 SW NÁSTROJE: CASE NÁSTROJE A JEJICH ROZDĚLENÍ (PRE, UPPER, MIDDLE, LOWER, POST). IDE NÁSTROJE. CASE IDE NÁSTROJE, PŘEHLED VYBRANÝCH NÁSTROJŮ (CASE STUDIO, ORACLE DESIGNER).

V této kapitole se seznámíme s pojmem CASE nástroje a jejich rozdělením podle různých kritérií. Uvedeme si způsoby jejich využití a konkrétní příklady vybraných CASE nástrojů.

Čas ke studiu: 2 hodiny

Cíl: Po prostudování této kapitoly budete umět

- Objasnit co jsou to CASE nástroje a k čemu slouží.
- Orientovat se v různých typech CASE nástrojů.
- Uvést příklady CASE nástrojů.

Výklad

9.1 Obecná charakteristika CASE nástrojů

CASE nástroj:

- CASE - Computer Aided System (Software) Engineering
- účel - automatizace některých fází vývoje IS
- množina integrovaných softwarových nástrojů

CASE nástroje pomáhají při tvorbě grafických modelů softwarových systémů. Jedná se o software sloužící k podpoře vývoje dalšího softwaru a vylepšení procesů jeho vývoje. CASE nástroje často obsahují možnosti pro ladění kódu, vytvoření uživatelského rozhraní, nástroje pro hromadnou úpravu kódů, atd.

Podpora CASE nástrojů spadá do mnoha činností a oblastí, mezi které se řadí např. podpora při stanovení požadavků, podpora při analýzách, podpora při návrhu a podpora při programování.

CASE nástroje umí generovat zdrojový kód, vytvářet datové modely, spravovat konfigurace, provádět refraktoring apod.

Pro člověka je často pochopitelnější systém zobrazený např. diagramem (tzn. obrázkem) než složité slovní popisy tohoto systému.

CASE nástroje rovněž disponují schopností automatického generování zdrojového kódu z modelů, což je významným usnadněním práce vývojářů. Některé CASE nástroje umožnují rovněž opak – generování diagramů/modelů ze zdrojového kódu (tzv. reverse engineering). V obou případech poskytují rovněž možnosti synchronizace mezi modelem a zdrojovým kódem.

Často jsou CASE nástroje využívány při tvorbě dokumentace – z modelu, ze zdrojového kódu.

Integrované nástroje:

- jsou schopné vzájemně si předávat výsledky
- jednotná databáze (repository) CASE nástroje

Výhody CASE:

- podpora tvorby katalogu požadavků na systém (specifikace systému)
- podpora tvorby analytických a návrhových modelů
- údržba projektové dokumentace k vyvíjenému IS
- automatické generování programového kódu
- automatické generování struktury databáze v daném databázovém prostředí
- podpora testování systému

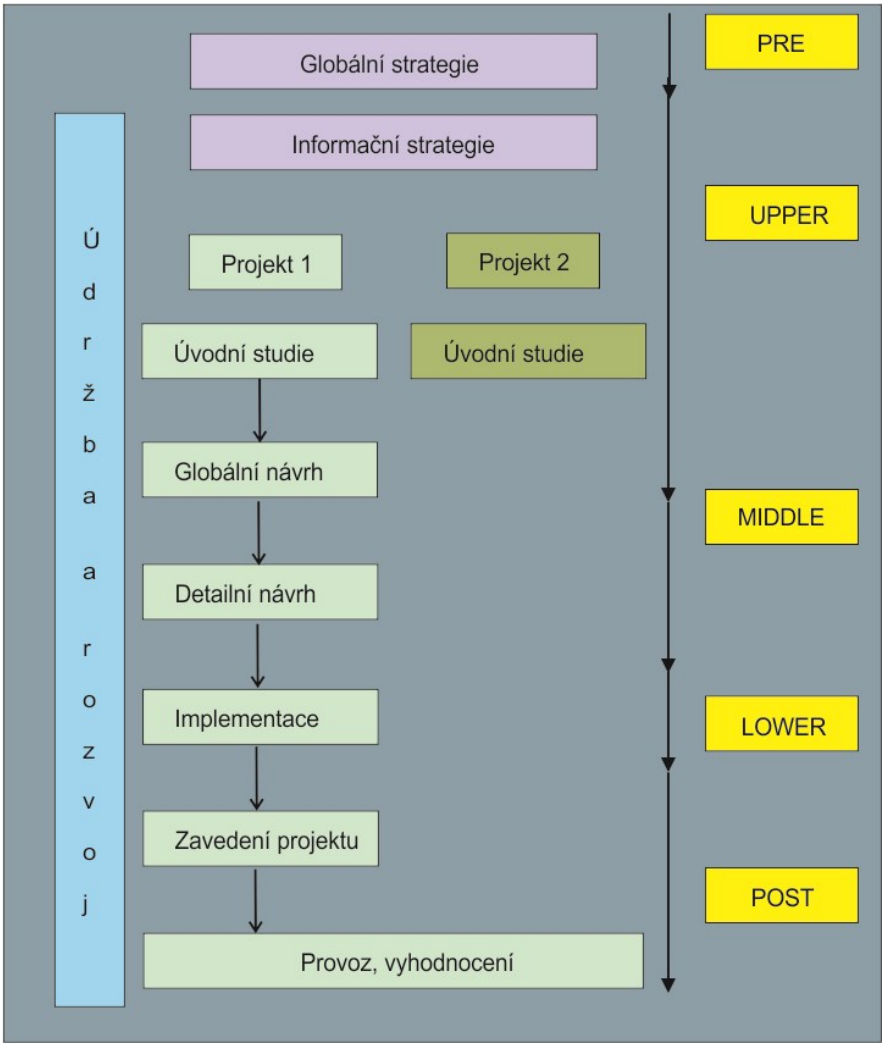
9.2 Dělení CASE nástrojů

CASE nástroje se dělí podle různých kritérií. Nejčastějším dělením je:

- dělení podle životního cyklu projektu (PRE CASE, UPPER CASE, MIDDLE CASE, LOWER CASE, POST CASE)
- dělení podle interaktivity
- dělení podle fáze projektu
- dělení podle délky využívání (jsou využívány během celého životního cyklu SW?)
- dělení podle stupně integrace

9.2.1 Dělení podle životního cyklu projektu

- **pre** CASE - jsou využívány při činnostech předcházejícím vývoji SW (globální strategie)
- **upper** CASE - jsou využívané v etapě analýzy systému
- **middle** CASE - jsou používány v etapě návrhu (designu) systému
- **lower** CASE - jsou používány v etapě implementace systému
- **post** CASE - jsou využívány k podpoře fáze uvedení SW do provozu, jeho údržbě a další organizační činnosti



Obrázek 9-1 Dělení CASE nástrojů podle životního cyklu projektu

9.2.2 Dělení podle interaktivity

Toto dělení koresponduje se svým názvem. CASE nástroje mohou, ale nemusí být interaktivní. Mezi interaktivní patří například nástroje, které podporují návrhové metody a mezi neinteraktivní se řadí například vývojové nástroje a překladače.

9.2.3 Dělení podle fáze projektu

V tomto dělení se jedná o tzv. front-endové či back-endové nástroje. Do jaké skupiny CASE nástroj patří, závisí na fázi vývoje software, ve které se využívají.

- **front-end** CASE – využívají se v prvotních fázích projektu, jedná se například o nástroje pro podporu návrhu.
- **back-end** CASE – využívají se v pozdějších fázích projektu. Jedná se například o testovací nástroje, překladače zdrojového kódu apod.

9.2.4 Dělení podle délky využívání

Jedná se o dělení na vertikální resp. horizontální CASE nástroje.

- **vertikální** – podporují jen určitou část nebo oblast životního cyklu software, například tvorbu programového kódu, modelování, zjišťování požadavků uživatele, atd.
- **horizontální** – podporují několik částí nebo oblastí životního cyklu software, jedná se například o nástroje určené ke tvorbě dokumentace, řízení konfigurace, apod.

9.2.5 Dělení podle stupně integrace

Toto dělení obsahuje následující kategorie CASE nástrojů:

CASE Tools – nástroje sloužící k automatizované podpoře jakékoli úlohy v životním cyklu software.

CASE Toolkits – jedná se o kolekci integrovaných CASE nástrojů, umožňujících dílčí či komplexní podporu pouze během jedné fáze projektu.

CASE Workbenches – poskytují dílčí nebo úplnou podporu v alespoň dvou fázích životního cyklu software.

I-CASE – jedná se o nejvyšší stupeň integrace. Propojují několik CASE Tools, Toolkits a Workbenches.

9.3 Použití CASE nástrojů v průběhu vývoje IS

Obecně k vlastnostem CASE nástrojů:

- účel CASE nástroje
  - vývoj informačních systémů a software v architektuře klient/server
  - vývoj aplikací speciálně pro Internet
  - vývoj aplikací v prostředí datových skladů
  - modelování a optimalizace podnikových procesů
- integrace CASE nástroje s ostatními nástroji
  - např. tvorba požadavků na systém a testování systému
- soulad CASE nástroje s používanou metodikou a metodami projektování IS
  - objektový a strukturovaný přístup
- používaná notace
  - dnešní standardy - UML, BPMN
- sledování požadavků specifikace a jejich řešení
- modularita CASE nástroje, jeho otevřenost a modifikovatelnost
  - customizace – přizpůsobení uživateli
- repository CASE nástroje
  - databáze, soubor, platforma OS, …
- sdílení komponent a správa projektu
  - podpora týmové práce

9.4 Příklady nástrojů CASE

9.4.1 Enterprise Architect (Sparx Systems)

Jedná se o velice sofistikovaný CASE nástroj, s použitím během celého životního cyklu software.

Modelování během celého životního cyklu zahrnuje použití pro:

- podnikové a IT systémy
- vývoj systémů a software
- vývoj v reálném čase a vývoj pro embedded platformy

Enterprise Architekt umožňuje modelování pomocí diagramů (UML), výstupy zdrojových kódů a propojení s řadou programovacích jazyků, sofistikované testování a monitorování.

Mezi podporované programovací jazyky patří například:

- ActionScript
- Ada
- C and C++
- C#
- Java
- Delphi
- Verilog
- PHP
- VHDL
- Python
- System C
- VB.Net
- Visual Basic
- a další…

Více informací na stránkách výrobce:
https://www.sparxsystems.com.au/products/ea/index.html

9.4.2 Case Studio

Tento CASE nástroj slouží především k navrhování databázových struktur.

V programu je možné snadno vizuálně vytvářet ER diagramy pro různé typy databází (Oracle, MS SQL, DB2, Firebird, Advantage DB server, Interbase, MaxDB, MS Access, MySQL, PostgreSQL a další).

Kromě ER diagramů nástroj umožňuje také tvorbu diagramu datových toků (DFD).

Je rovněž silným nástrojem pro reverse engineering - umožňuje vytvořit model struktury již existující databáze;

Nástroj slouží také jako správce verzí - umožňuje porovnávat jednotlivé verze modelů. Mezi další silné stránky nástroje patří rovněž velice detailní logické i fyzické HTML reporty,

vytvoření galerií pro uložení nejčastěji používaných částí modelů, podpora uživatelů, uživatelských skupin a uživatelských práv, uživatelsky definovaných šablon, možnost tvorby tzv. "ToDo" seznamu, vytvoření datového slovníku, atd.

Více informací na stránkách výrobce: http://www.casestudio.com/enu/index.aspx

9.4.3 Oracle Designer (Oracle)

Oracle Designer je dodáván v rámci balíku Oracle Developer Suite. Designer zahrnuje podporu pro modelování podnikových procesů, systémovou analýzu, návrh software a tvorbu systémů. Poskytuje víceuživatelský repozitář založený na Oracle SCM, který je úzce integrován s nástrojem Oracle Forms Developer (nástroj pro vytváření deklarativních databázových aplikací). Tímto způsobem Oracle designer umožňuje organizacím navrhovat a rychle dodávat systémy v architektuře klient-server, které jsou přizpůsobitelné měnícím se potřebám podniku.

Více informací na stránkách výrobce: http://www.oracle.com/technetwork/developer-tools/designer/overview/index-082236.html

9.4.4 Další CASE nástroje

Mezi další nástroje pro podporu modelování systémů patří například:

- MagicDraw (No Magic) - https://www.nomagic.com/products/magicdraw
- Powerdesigner (Sybase) - https://www.sap.com/cz/products/powerdesigner-data-modeling-tools.html
- Rational Rose (IBM) - http://www-03.ibm.com/software/products/en/enterprise
- Microsoft Visio (Microsoft) - https://www.microsoft.com/en-us/store/collections/visio

Shrnutí pojmů 9.1.

V této kapitole jsme se dozvěděli informace o tom, co jsou CASE (Computer Aided System (Software) Engineering) nástroje. Víme, že se často jedná o množinu integrovaných softwarových nástrojů.

Účelem jejich použití je zejména automatizace některých fází vývoje software. Jinými slovy, CASE nástroje pomáhají při tvorbě grafických modelů softwarových systémů.

CASE nástroje řadíme do kategorií podle různých kritérií:

- podle životního cyklu software **pre**, **upper**, **middle**, **lower**, **post**
- podle interaktivity

- podle fáze projektu - **front-end**, **back-end**
- podle délky využívání – **vertikální**, **horizontální**
- podle stupně integrace - **CASE Tools**, **CASE Toolkits, CASE Workbenches**, **I-CASE**

V kapitole jsme si uvedli rovněž příklady některých komerčních CASE nástrojů: MagicDraw, Powerdesigner, Rational Rose (IBM), Microsoft Visio, Enterprise Architekt, Oracle Designer a Case Studio.

Otázky 9.1.

1. Co jsou CASE nástroje a k čemu slouží?
2. Jak dělíme CASE nástroje?
3. Jaký je rozdíl mezi horizontálními a vertikálními CASE nástroji?
4. Jaké CASE nástroje znáte?
5. Jaký CASE nástroj byste použili pro modelování ER digramu a diagramu datových toků (DFD)?

10 TRENDY V OBLASTI MODELOVÁNÍ SW: AKTUALITY, VÝVOJ, VÝZKUM, TECHNICKÉ NOVINKY V OBORU SW INŽENÝRSTVÍ.

V této kapitole se seznámíme s některými trendy v oblasti softwarového inženýrství. Seznámíme se s rozšířením jazyka UML – MDA a jeho hlavních vlastnostech. V několika bodech rovněž uvedeme hlavní body současného vývoje oboru softwarového inženýrství. V závěru kapitoly se stručně seznámíme s výhodami/nevýhodami automatizace modelování a důvodech prospěšnosti využívání šablon.

Čas ke studiu: 2 hodiny

Cíl: Po prostudování této kapitoly budete umět

- Objasnit vztah UML a MDA.
- Uvést hlavní prvky MDA a jejich notaci.
- Orientovat se ve výhodách automatizace modelování a využití šablon.

Výklad

10.1 Obecné trendy v oboru softwarového inženýrství

Obecně je možné trendy v oblasti modelování SW shrnout do několika bodů, které stručně vyjadřují aktuální směr a vývoj v tomto oboru:

- **použití iterativních postupů** - aplikováno v podobě spirálového modelu a techniky prototypování.
- **pronikání objektových metod, technik a nástrojů**
- **„globalizace“ pojetí analýzy** - před automatizací podnikových procesů je nutné je optimalizovat a teprve pak provést analýzu…
- **posun od „hard“ k „soft“ metodám** - Software je nutno chápat jako sociálně-technický. Současné metodiky tedy zahrnují i techniky analýzy zájmů a postojů různých skupin uživatelů, školení, konzultací…prostě – i lidská dimenze je důležitá.

- **aplikace metodik pro implementaci typového aplikačního SW (TASW)** – např. zavádění ekonomického systému SAP/R3, produktů Oracle, atd.
- **vznik a rozvoj agilních metodik** – metodik, které se snaží oprostit od náročnějších postupů vývoje v zájmu rychlosti vývoje…

Mimo tyto body lze v této souvislosti nastínit i skutečnosti uvedené dále v této kapitole:

10.2 UML a MDA

V minulosti byly používány verze UML s číselným označením nižším než 2. Schválená verze 2 normy UML s označením UML 2.0 zavádí změny, které se týkají především modelování činností (aktivit), komponent a interakcí. Pořád však existuje mírná „neomalenost" UML 2.0, vzhledem k tomu, že je směsí nejednotných způsobů notace pro tvorbu real-time a podnikových systémů, což jej činí složitějším v používání, pro pochopení a zvládnutí.

Uvedená skutečnost vedla konsorcium OMG (jakožto vývojáře jazyka UML) k vývoji novější architektury, s názvem MDA (Model Driven Architecture/Modelem řízená architektura).

10.2.1 Modelování v MDA a členění modelů

MDA umožňuje standardizaci dvou rovin modelování:

- členění modelů během vývoje
- obecné zásady transformace modelů (jednoho modelu na druhý)

V MDA existují tři typy modelů:

- **ICIM** – zkratka pro Computation Independent Model, model, který je nezávislý na počítačovém zpracování
- **IPIM** – zkratka pro Platform Independent Model, model, který je nezávislý na platformě pro implementaci
- **IPSM** – zkratka pro Platform Specific Model, model určený pro specifickou platformu

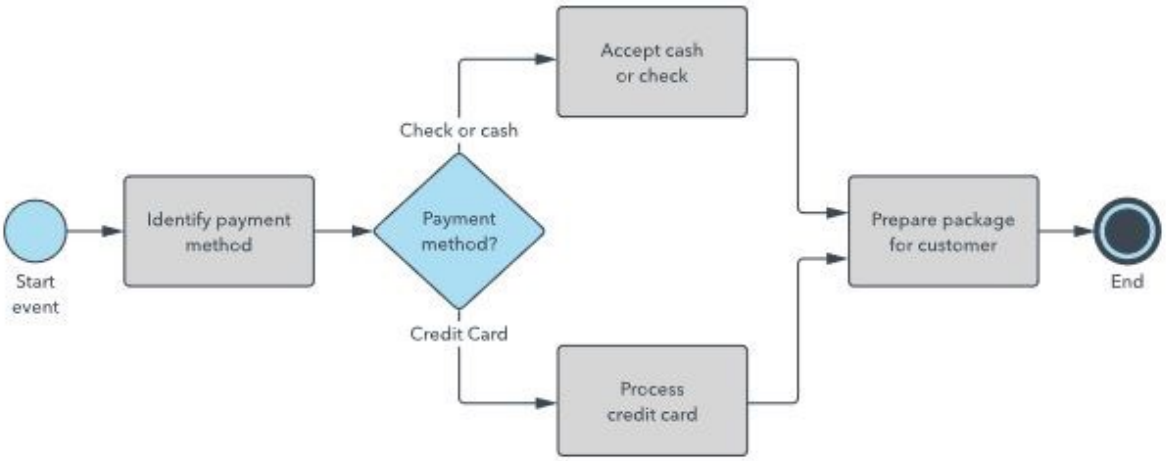
Tyto tři modely mají z hlediska tvorby modelů aplikací své specifické vlastnosti. S těmito vlastnosti se podrobněji seznámíme v dalším textu.

ICIM – tento model vznikl na základě začlenění normy iniciativy BPMI (Business Process Modeling Initiative) pod správu konsorcia OMG. Iniciativa BPMI vytvořila normu BPMN – Business Process Modeling Notation, kterou je možné využít k modelování

podnikových procesů. Dosud používané nejednotné notace byly touto normou ve značné míře standardizovány a ujednoceny. BPMN nejenže klade důraz na standardizovanou vizualizaci procesu, ale rovněž na mapování symbolů BPMN do jazyka BPEL (Business Process Execution Language), používaného pro tvorbu vizuálních modelů. Toto mapování by mělo ulehčit automatizaci procesu při modelování v systémech workflow a také při jejich případné simulaci.

Zatím se nedá hovořit o přímém začlenění BPMN do jazyka UML, nicméně BPMN je zavedena jako rozšíření (rozšiřující norma) jazyka UML určená pro modelování podnikových procesů.

V roce 2011 byla společností OMG vydána verze BPMN 2.0, která je využívána do současné doby (2017).



Obrázek 10-1 Příklad notace BPMN. Zdroj: https://www.lucidchart.com/pages/bpmn

10.1.1.1 Notace v BPMN

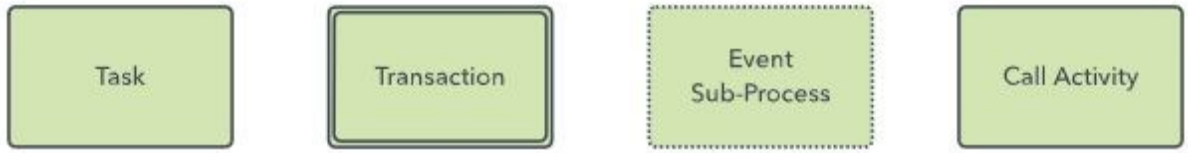
Pro diagramy podnikových procesů BPMN definuje čtyři typy elementů:

1. **Objekty toků** – události, činnosti a brány
2. **Spojovací objekty** – sekvenční toky, toky zpráv a asociace
3. **Plavecké dráhy** – bazén nebo pruh (cesta)
4. **Artefakty** – datový objekt, skupina, anotace

Události – jsou spouštěče, které spouští, modifikují nebo ukončují proces. Typy událostí jsou např. zprávy, časovače, chyby, signály, zrušení apod. Zobrazují se jako kruhy obsahující další symboly v závislosti na typu události. Jsou také dále klasifikovány jako události vyvolávající nebo reagující.

Obrázek 10-2 Události. Zdroj: https://www.lucidchart.com/pages/bpmn

Činnosti (aktivity) – zobrazení konkrétní činnosti nebo úlohy prováděné osobou nebo systémem. Značí se obdélníkem se zaoblenými rohy.



Obrázek 10-3 Činnosti. Zdroj: https://www.lucidchart.com/pages/bpmn

Brány – jsou rozhodovacími body, které nastavují cestu v závislosti na podmínkách nebo událostech. Jejich notací je čtverec otočený vrcholem dolů. Mohou být exkluzivní nebo inkluzivní, paralelní, složené nebo založené na datech či událostech.

Obrázek 10-4 Brány. Zdroj: https://www.lucidchart.com/pages/bpmn

Sekvenční toky – znázorňují pořadí prováděných činností. Značí se rovnou čárou zakončenou šipkou. Používá se pro znázornění podmíněného toku nebo běžného toku.

Obrázek 10-5 Sekvenční toky

Toky zpráv- označují zprávy, které jsou předávány přes „bazény“ (pooly) nebo hranice organizací (např. jednotlivá oddělení). Neměly by být používány pro spojení mezi událostmi a činnostmi v poolu. Znázorňují se přerušovanou čarou s kroužkem na jedné straně a šipkou na straně druhé.

Obrázek 10-6 Toky zpráv. Zdroj: https://www.lucidchart.com/pages/bpmn

Asociace – asociují artefakt nebo text s událostí, činností nebo bránou. Značí se tečkovanou čarou.

Obrázek 10-7 Asociace. Zdroj: https://www.lucidchart.com/pages/bpmn

Bazén (pool) a plavecká dráha – pool znázorňuje hlavní účastníky procesu. Rozdílný pool může představovat jinou společnost nebo oddělení, které jsou však stále zapojeny do procesu. Plavecké dráhy v bazénu znázorňují činnosti a toky pro určitou roli nebo účastníka, čímž definují odpovědnost – kdo je odpovědný za dané části nebo proces.

Artefakt – dodatečné informace přidávané vývojáři za účelem zpřístupnění dalších podrobností. Používají se tři typy artefaktů: datový objekt, skupina nebo anotace. Datový objekt ukazuje potřebnost dat pro činnost. Skupina znázorňuje logické seskupení činností, avšak nemění toky diagramu. Anotace poskytuje další vysvětlení k části diagramu.

IPIM – v současné době je dominujícím nástrojem pro tvorbu platformě nezávislého modelu v etapě analýzy jazyk UML. I přesto, že UML 2.0 zavádí přehlednější členění a lepší provázanost diagramů při modelování interakcí, stále existuje mezera v podobě formálního

specifikačního jazyka, který by byl vhodný i pro účely analýz. Současná specifikace UML sice umožňuje využití doplňku OCL (Object Constraint Language), ale ten je z hlediska čitelnosti běžnými uživateli nevhodný, neboť se podobá objektově orientovaným jazykům, do kterých je zasvěcena pouze úzká skupina uživatelů, zejména programátorů a informatiků. Z vizuálního hlediska jsou sice analytické modely normovány, avšak popisy jednotlivých elementů jsou zcela na uvážení tvůrců těchto modelů. Tento neduh se, bohužel, objevuje i nadále v MDA, neboť ani tato architektura neobsahuje přesná vymezení toho, co by mělo být obsahem na platformě nezávislého modelu. Určitým přínosem a zlepšením MDA v této oblasti je alespoň začlenění opakovaně použitelných částí – iterace se tak díky nim stávají rovněž součástí analýzy a mají zjevně podstatný vliv na náklady projektu, management jakosti a postup prací na projektu.

IPSM – v současné době různá IDE (Integrated Development Environment, integrovaná vývojová prostředí) nabízí programátorům vizualizaci programového kódu ve formě modelu UML. Nezbytnou součástí aktivních znalostí vývojáře se tedy stává i UML. Modely UML, vygenerované ve vývojových prostředích jsou svou strukturou shodné se zdrojovým kódem, a tudíž nepříliš čitelné pro běžného uživatele, který nemá znalosti programátora. Analytik většinou tyto modely s obtížemi pochopí, pro běžného uživatele aplikace je však porozumění takovýmto modelům nad rámec jeho chápání. Jistým řešením je synchronizace programových kódů s modely prostřednictvím různě sofistikovaných nástrojů CASE, s využitím XMI (XML Metadata Interchange) – standardů zavedených a vypracovaných sdružením OMG. Současným trendem je využívání diagramu interakcí (Diagram Interchange), který rozšiřuje specifikaci formátu XMI scílem standardizace přenosu a prezentace diagramů UML.

10.3 Automatizace modelování

Automatická tvorba různých částí modelů pomocí modelovacích nástrojů (CASE) je dalším trendem současné doby (2017). Tyto nástroje se mohou pyšnit stále se rozšiřující podporou a integrací nástrojů automatizace tvorby modelů.

Obvyklým fenoménem při práci vývojářů je propracované vytvoření diagramu či modelu tříd na úkor ostatních modelů systému. Zmíněný fenomén je častým důsledkem nedostatku času a ve vytvářených modelech jsou pak zahrnuty pouze nejdůležitější části systému a podrobnější zpracování chybí. To má za následek zvýšené riziko chybovosti a nekonzistentnosti, protože provádět důkladné automatické testování těchto nekompletních

modelů je často nemožné. Vytváření kompletních modelů však stojí mnoho času a jedná se o velice pracnou a zdlouhavou a nepříliš oblíbenou činnost.

Určitým řešením tohoto problému je automatizace na základě předem připravených šablon, a to zejména u mnoha rutinních činností. Šablony jsou však často závislé na mnoha faktorech a proto jsou využívány zejména při modelování pro specifické platformy. Velmi obvyklá je také tvorba šablon na základě návrhových vzorů. Pro společnosti, zabývající se častým vývojem software pro specifické platformy, může být investice do vytvoření šablon pro používané stereotypy modelů velice přínosná. Šablony využijí totiž nejen při vlastním modelování systému a s ním související důkladném automatizovaném testování konzistence modelů, ale i později ve fázi údržby sytému, protože jsou do vytvořeného modelu zahrnuty veškeré závislosti.

Shrnutí pojmů 10.1.

V kapitole zabývající se trendy v oblasti modelování software jsme se dotkli některých nedokonalostí UML a uvedli si jejich řešení (do určité míry), zaváděná MDA.

Popsali jsme tři hlavní typy modelů v MDA:

- **ICIM**
- **IPIM**
- **IPSM**

Ukázali jsme si také způsoby notace standardizované v BPMN a popsali čtyři typy prvků, které BPMN využívá:

- **Objekty toků,** mezi které patří události, činnosti a brány
- **Spojovací objekty,** mezi které patří sekvenční toky, toky zpráv a asociace
- **Plavecké dráhy,** mezi které patří bazén nebo pruh (dráha)
- **Artefakty,** mezi které patří datový objekt, skupina a anotace

V posledním odstavci o automatizovaném modelování jsme nastínili důvody využití šablon a výhody jejich využívání při častém modelování procesů softwarových systémů. Řekli jsme si také, k čemu vede modelování s neuvedením podrobností do dostatečné hloubky.

Otázky 10.1.

1. Co znamená zkratka MDA?
2. Jaké byly důvody vzniku MDA?
3. Jaké jsou hlavní typy modelů v MDA?
4. Proč je současným trendem posun od hard k soft metodám?
5. Uveďte typy prvků používané v notaci BPMN a popište jejich značení a význam.
6. Jaké výhody má využití šablon modelů?