KAPITOLA 3

Základy metody

V této kapitole:

- Klíčové koncepty
- Obsah metody
- Proces
- Shrnutí

Po seznámení s vybranými základními koncepty můžeme nyní přejít k některým podrobnostem, podpírajícím celý proces navrhování softwaru, jako například výsledkům práce, úkolům a rolím. Při přípravě tohoto přehledu jsme zvážili různá doporučení, týkající se navrhování a používaná v softwarovém průmyslu, a současně jsme čerpali z prvků různých přístupů, včetně RUP (Rational Unified Process), IBM Unified Method Framework, OpenUP, XP (eXtreme Programming), Scrum, FDD (Feature-Driven Development) a Lean. Samozřejmě jsme vzali v potaz i související standardy, k nimž patří například SPEM (Software and Systems Process Engineering Metamodel Specification).

Hlavním cílem této kapitoly je nabídnout vám přehled klíčových prvků metody, z nichž tato kniha vychází. Lze tedy říci, že tato kapitola je jakýmsi základem pro kapitoly zbývající, neboť v nich budeme stavět na těchto konceptech. Přehled prvků metody, popsaných v této knize, najdete v příloze C „Přehled metody".

Klíčové koncepty

Aby další diskuze v této kapitole vůbec měly nějaký smysl, musíme si nejprve vyjasnit některé základní koncepty, které jsou součástí metody. Přitom nám může pomoci standard SPEM 2007 (Software and Systems Process Engineering Metamodel Specification Object Management Group), neboť definice těchto konceptů jsou jeho součástí a my je budeme v této knize používat. Standard SPEM je ovlivněn několika existujícími metodami vývoje softwaru a ty také podporuje. Patří mezi ně OpenUP (OpenUP 2008), RUP 2008 (Rational Unified Process), IBM Unified Method Framework, Fujitsu DMR Macroscope a Unisys QuadCycle.

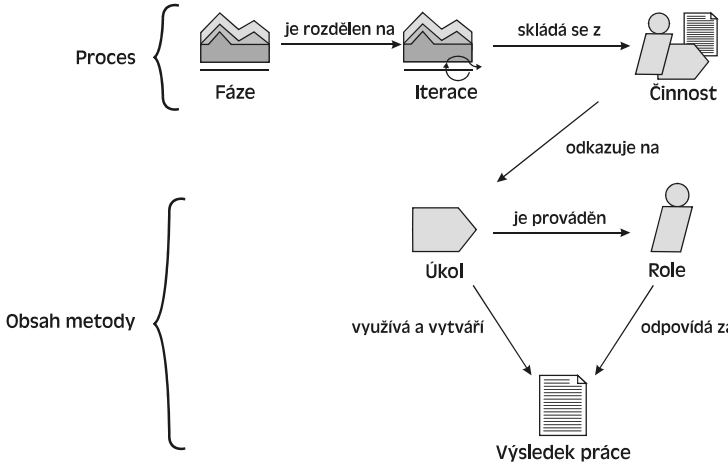
Standard SPEM definuje různé koncepty do značné hloubky. Pro účely této knihy jsme však převzali pouze menší část a současně jsme přijali některé zjednodušující předpoklady. Na obrázku 3.1 vidíte přehled základních pojmů, používaných v této knize. Obrázek je převzat ze standardu SPEM, a to včetně vztahů a ikon (pokud ovšem byly ve standardu definovány).

V podstatě lze říci, že efektivní metoda vývoje softwaru by měla stanovovat kdo, kdy, jak a co dělá. Tato kniha to činí, a to pomocí následujících klíčových konceptů:

- Role: kdo
- Výsledky práce: co
- Úkoly: jak
- Fáze, iterace a činnosti: kdy

Kromě toho by součástí metody mělo být i jakési vodítko v podobě šablon, příkladů a postupů. Vrátíme-li se zpět k obrázku 3.1, je zřejmé, že projekt vývoje softwaru obvykle prochází několika *fázemi*, z nichž každá je rozdělena do několika *iterací* (ačkoliv, jak uvidíte později, ne všechny procesy jsou uspořádány do fází a iterací). V rámci každé iterace pak hovoříme o různých *činnostech* a *úkolech*, na které se odkazují, přičemž ty jsou prováděny za účelem dosažení určitého výsledku. Úkoly jsou prováděny příslušnými *rolemi* a v rámci úkolů jsou využívány a vytvářeny *výsledky práce*.

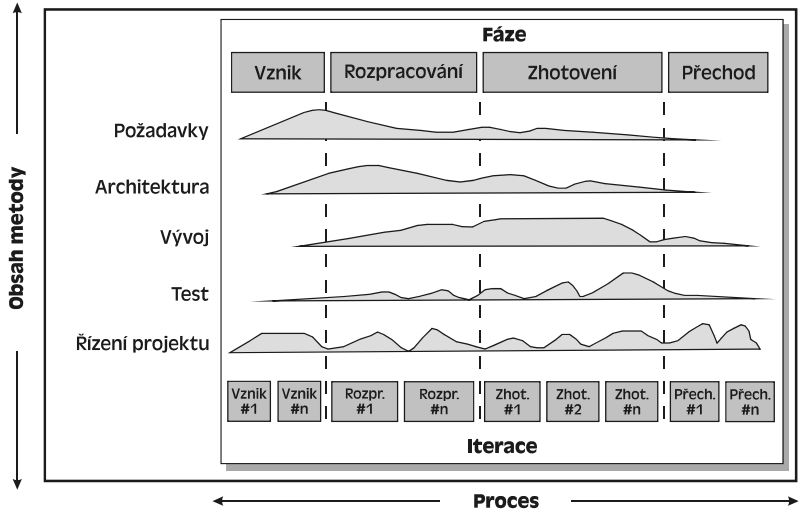
Z obrázku 3.1 také vyplývá, že o metodě lze říci, že se skládá z *obsahu metody* a *procesu*. Obsah metody popisuje prvky, nezávislé na životním cyklu projektu, jako například role, úkoly a výsledky práce. Následně proces přebírá tyto prvky a určuje posloupnost, v níž jsou použity, a to v závislosti na podstatě konkrétního projektu. Proces pak pracuje s takovými koncepty jako například fázemi, iteracemi a činnostmi. Příklad oddělení obsahu metody a procesu nabízí OpenUP, jehož vizualizaci najdete na obrázku 3.2. Svislá osa, kterou vidíte na tomto obrázku, představuje obsah metody, seskupený podle disciplín, zatímco vodorovná osa popisuje proces.



Obrázek 3.1: Klíčové koncepty metody a jejich vztahy

[Disciplínou je] Primární třídící mechanismus pro uspořádání úkolů, definujících hlavní „oblast zájmu", případně součinnost pracovního úsilí. (OpenUP 2008)

Pojmem *disciplína* tedy můžeme označit sadu činností, týkajících se hlavní oblasti zájmu v rámci celého projektu. Z obrázku 3.2 vyplývá, že metoda OpenUP je uspořádána okolo 5 disciplín, jejichž přehled najdete v tabulce 3.1.



Obrázek 3.2: Obsah metody a dimenze procesu v OpenUP

„Hrboly" na obrázku 3.2 znázorňují to, že v průběhu života projektu se relativní důraz jednotlivých disciplín mění. Například v prvních iteracích se obvykle více času stráví prací nad požadavky, zatímco v pozdějších iteracích se více času věnuje vlastnímu vývoji.

Tabulka 3.1: Přehled disciplín OpenUP

Disciplína OpenUP	Popis
Požadavky	Tato disciplína říká, jakým způsobem mají být zjištěny, analyzovány, specifikovány, ověřeny a řízeny požadavky vyvíjeného systému.
Architektura	Tato disciplína popisuje způsob vytvoření softwarové architektury z těch požadavků, které jsou pro architekturu zásadní. Architektura je integrální součástí disciplíny Vývoj.
Vývoj	Tato disciplína říká, jakým způsobem má být navrženo a implementováno technické řešení, odpovídající architektuře a podporující požadavky.
Test	Tato disciplína popisuje způsob získání zpětných informací o dozrávajícím systému, a to na základě návrhu, implementace, spuštění a vyhodnocení různých testů.
Řízení projektu	Tato disciplína říká, jakým způsobem lze tým vést a podporovat a pomáhat mu tak při překonávání rizik a překážek, na které v průběhu vývoje narazí.

Definice rolí, úkolů a výsledků práce jsou obvykle považovány za opětovně použitelné, protože na rozdíl od prvků, souvisejících s procesem, se mezi různými projekty a odlišnými typy životních cyklů příliš neliší. Během projektu vývoje softwaru, začínajícího s vývojem od prázdného listu papíru, bude úkol typu **Identifikace funkčních požadavků** prováděn téměř stejně (ve stejných krocích), jako během projektu, měnícího již existující systém; v průběhu životního cyklu projektu však bude důraz na tento úkol velmi odlišný.

Obsah metody

Jak jsme se zmínili v předcházející části této kapitoly, obsahem metody se míní role, výsledky práce a úkoly, tvořící příslušnou metodu.

Role

Pojem *role* definuje odpovědnosti jednotlivce či skupiny jednotlivců, pracujících společně jako tým v rámci kontextu organizace, zabývající se vývojem softwaru. Role je odpovědná za jeden či více výsledků práce a provádí sadu úkolů. Například role **Obchodní analytik** je odpovědná za výsledek práce **Funkční požadavky** a provádí úkol **Identifikace funkčních požadavků**. V této knize jsme k rolím přistoupili tak, že pokud role provádí nějaký úkol, pak daná role může být buď primární nebo sekundární. Více informací najdete v poznámce „Koncept: Primární a sekundární role".

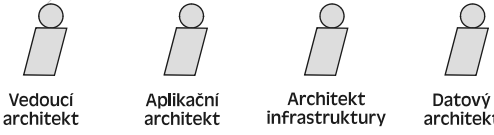
Je nezbytné zdůraznit, že role nejsou totožné s jednotlivci. Jednotlivci mohou zastávat více rolí (neboli mohou sedět na několika židlích) a několik jednotlivců může vykonávat jedinou roli. **Projektový manažer** je odpovědný za provedení mapování jednotlivců na role, a to v rámci plánování projektu a vytváření jeho týmu; samozřejmě však platí, že **Projektový manažer** tuto otázku konzultuje i s ostatními. Role, související s architekturou a používané v této knize, jsou znázorněny na obrázku 3.3. Přitom platí, že ne všechny projekty vyžadují tyto specializované role; mnohdy stačí jediná role *architekta*.

Vedoucí architekt má celkovou odpovědnost za nejdůležitější technická rozhodnutí, definující architekturu systému. Tato role je také odpovědná za přípravu veškerých podkladů pro tato rozhodnutí, vyvážení zájmů různých investorů, řízení technických rizik a problémů a také za zajištění toho, že tato rozhodnutí jsou efektivně komunikována, ověřena a dodržována.

Aplikační architekt se zaměřuje na ty prvky systému, které zajišťují automatizaci procesů a splnění obchodních požadavků. Jinými slovy řečeno, tato role se zaměřuje především na funkcionalitu, požadovanou obchodníky, ale zajímá se i o to, jakým způsobem prvky, související s aplikací, plní nefunkční požadavky systému (kvality a omezení).

Ve středu zájmu role **Architekt infrastruktury** leží ty prvky systému, které nezávisí na obchodní funkcionalitě, jako například hardware, middleware či datová úložiště. Tyto prvky podporují spouštění prvků, souvisejících s aplikací. Popisovaná role se zaměřuje na prvky, mající zásadní dopad na kvality, vykazované systémem, tedy i na rozsah splnění určitých nefunkčních požadavků.

Datový architekt je odpovědný za datové prvky systému, především tedy ta data, která jsou dlouhodobě uchovávána pomocí nějakého vhodného mechanismu, jímž může být například databáze, souborový systém, systém pro správu obsahu (CMS – Content Management System) či nějaké jiné úložiště. Tato role definuje vhodné vlastnosti, týkající se dat, jako například strukturu, zdroj, umístění, integritu, dostupnost, výkon a stáří.



Obrázek 3.3: Role, související se softwarovou architekturou

Koncept: Primární a sekundární role

Primární role určitého úkolu je považována za odpovědnou za tento úkol a je vždy povinná. Naopak u sekundární role se předpokládá, že pouze přispívá k plnění úkolu, a proto není za úkol odpovědná. Z tohoto důvodu je možné ji považovat za volitelnou.
Úplnější charakteristika rolí vyplývá z klasifikace RACI, v níž každá role může být Odpovědná za úkol (z anglického **R**esponsible), Schvalující úkol (z anglického **A**ccountable for), Konzultovaná během úkolu (z anglického **C**onsulted; názor nositele této role je žádoucí a je vždy brán v potaz) a Informovaná o úkolu (z anglického **I**nformed; nositel této role je průběžně informován o postupu prací na úkolu). V naší zjednodušené charakterizaci je primární role tou rolí, která je jak odpovědná, tak i schvalující; sekundární role pak je jak konzultovaná, tak i informovaná.

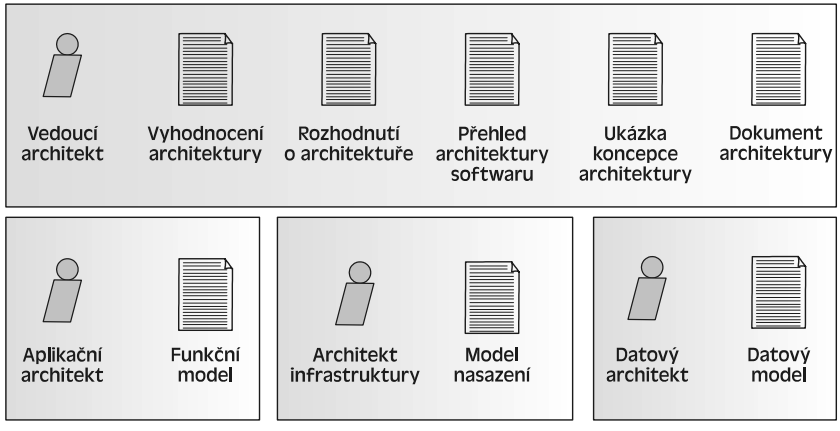
Výsledek práce

Výsledek práce je částí informace či fyzického celku, který je v průběhu procesu vytvářen, případně využíván. Typickými příklady výsledků práce mohou být modely, plány, kód, spustitelné soubory, dokumenty, databáze apod. Výsledek práce je *odpovědností* jediné role, byť na *vytvoření* jediného výsledku práce může spolupracovat několik rolí. Role využívají výsledky práce jako vstupy pro plnění úkolů, přičemž role také vytvářejí či upravují výsledky práce během úkolů, které plní. Jak se dozvíte dále v této kapitole a v kapitolách, věnovaných případové studii (což jsou kapitoly 6 až 9), architekt se účastní několika úkolů, během nichž jsou vytvářeny a využívány výsledky práce.

SPEM definuje celkem tři druhy výsledků práce: artefakty, předměty plnění a výsledky. *Artefaktem* se rozumí hmotný výsledek práce, jako například dokument, model, zdrojový kód, spustitelný soubor či projektový plán. Pojmem *předmět plnění* se také označuje hmotný výsledek práce, představovaný ovšem nějakým obsahem, připraveným (zabaleným) pro dodávku. Tímto pojmem se obvykle označují ty výsledky práce, které mají nějakou hodnotu (materiálovou či ji-

nou) pro investora. *Výsledek* pak představuje výsledek či nějaký stav, který je výsledkem provedení nějakého úkolu. Na rozdíl od artefaktů a předmětů plnění výsledky nepředstavují prvky, které by bylo možné opětovně použít.

Přístup, který jsme použili v této knize, se primárně zaměřuje na artefakty. Kromě artefaktů je architekt zodpovědný také za předmět plnění **Dokument architektury softwaru**. Výsledky práce, které jsou výlučnou odpovědností architekta, jsou ukázány na obrázku 3.4. V něm jsou ukázány i různé role architekta. Rádi bychom poznamenali, že jakákoliv jiná metoda může do tohoto obrázku přidat další výsledky práce či z něj naopak některé výsledky práce ubrat.



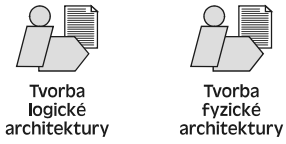
Obrázek 3.4: Výsledky práce, vlastněné rolemi souvisejícími s architekturou

Tento obrázek ve vás může vyvolávat dva zcela nesprávné dojmy. Zaprvé se vám může zdát, že všechny výsledky práce jsou dokumenty (a to proto, že SPEM používá k označení výsledku práce ikonu, která vypadá jako dokument!). Toto však není pravda. Například **Funkční model** a **Model nasazení** mohou být představovány formou UML modelů, vytvořených ve vhodném modelovacím nástroji, zatímco **Rozhodnutí o architektuře** může být popsáno na wiki stránkách projektu. A zadruhé se vám může zdát, že *úroveň formálnosti*, související s každým výsledkem práce, značně závisí na kontextu (jako například složitosti systému, okamžiku v životním cyklu projektu a podobně). Avšak například výsledek práce **Přehled architektury** může mít rozsah jediné strany A4 a výsledek práce **Rozhodnutí o architektuře** mohou tvořit tři položky v tabulce.

Ačkoliv jsme je na obrázku 3.4 neukázali, existují také další výsledky práce, na nichž se architekt podílí, ale není jejich vlastníkem. Typickým příkladem může být **Seznam prioritizovaných požadavků**. Dalším příkladem může být **Protokol RAID**, na němž architekt také spolupracuje, ale je vlastněn **Projektovým manažerem**. Jak již asi tušíte, existuje vazba mezi rolí, odpovědnou za úkol, a vlastnictvím jakéhokoliv výsledku práce, který je výstupem daného úkolu. Například role **Obchodní analytik** je odpovědná za úkol **Sběr požadavků investorů** a je vlastníkem výsledku práce **Požadavky investorů**.

Činnost

Činnost představuje seskupení úkolů. Architekt provádí úkoly v rámci činností, ukázaných na obrázku 3.5. Jejich podrobnější popis najdete v kapitole 8 „Tvorba logické architektury“ a kapitole 9 „Tvorba fyzické architektury“.

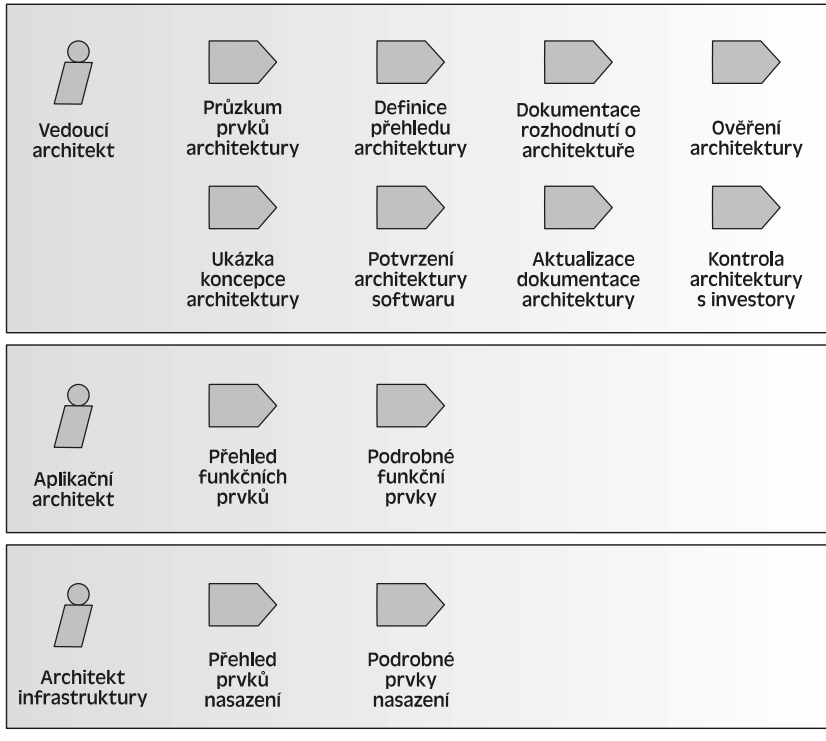


Obrázek 3.5: Činnosti, související s architekturou

Úkol

Úkolem je jednotka práce, mající z hlediska kontextu projektu nějaký smysluplný výsledek. Úkol má jasný cíl, jehož součástí obvykle je vytvoření či aktualizace nějakých výsledků práce. Všechny úkoly jsou prováděny příslušnými rolemi.

Jednotlivé úkoly mohou být několikrát opakovány, a to především tehdy, je-li zvolen iterativní proces vývoje (iteracím se budeme věnovat dále v této kapitole). Na obrázku 3.6 vidíte přehled všech úkolů souvisejících s architekturou. U každého úkolu je současně uvedena i příslušná primární role. Podrobnější popis těchto úkolů najdete v kapitole 8 „Tvorba logické architektury" a kapitole 9 „Tvorba fyzické architektury". Měli byste si ale zapamatovat, že znázorněná sada úkolů představuje pouze základní úkoly, týkající se architektury, a že architekt se obvykle účastní i dalších úkolů, jako například vývoje technologické strategie či rozvoje znalostí. Jak uvidíte dále, architekt také pomáhá dalším rolím, jako například kontrolou vývojového procesu, identifikací vlastníků, definicí a prioritizací požadavků, tvorbou různých odhadů a plánováním.



Obrázek 3.6: Úkoly související s architekturou

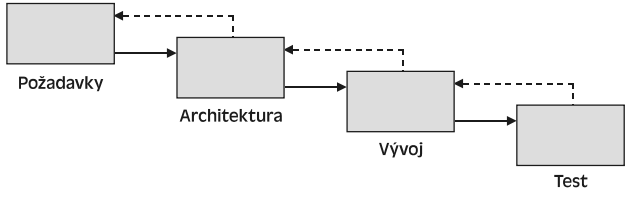
Proces

Svoji pozornost nyní přesuneme k aplikaci obsahu metody, a to z hlediska posloupnosti, v níž jsou jednotlivé úkoly prováděny. Jak uvidíte, mnohé z rozdílů mezi jednotlivými metodami, používanými v softwarovém průmyslu, se týkají především procesu, podle kterého se postupuje, nikoliv rolí, výsledků práce, činností a prováděných úkolů.

V této části se budeme zabývat celkem třemi typy procesu, označovanými jako *sériový proces* (*proces vodopádu*), *iterativní proces* a *agilní proces*. Přitom se nebudeme věnovat pouze klíčovým charakteristikám každého typu, ale současně zdůrazníme i ty postupy, které se nejvíce týkají právě softwarového architekta, a také prozkoumáme některé mýty, prostupující podobnými diskuzemi.

Sériový proces (proces vodopádu)

Schéma tradičního sériového procesu je znázorněno na obrázku 3.7. Při vytváření tohoto obrázku jsme názvy jednotlivých disciplín převzali z OpenUP. Při popisovaném přístupu je každá z disciplín považována za dokončenou až ve chvíli, kdy jsou vytvořeny a schváleny všechny výsledky práce, související s danou disciplínou. Například disciplína požadavků může být považována za dokončenou až ve chvíli, kdy jsou všechny požadavky identifikovány, podrobně definovány a zkontrolovány. Výstup z požadavků se pak stane vstupem pro disciplínu architektury. Tento proces pokračuje až do chvíle, dokud systém není podrobně navržen a jeho kód není napsán (v rámci disciplíny vývoj), poté otestován a zpřístupněn koncovým uživatelům. Změny výsledků práce (znázorněné čárkovanými šipkami, směřujícími zpět) jsou obvykle zpracovávány formálním procesem řízení změn.



Obrázek 3.7: Sériový proces

Tento přístup se využívá velmi často, a to především v rámci takových projektů, které představují spíše jen nevelká rozšíření nějakého již existujícího systému, či v rámci projektů vývoje takových systémů, které nezahrnují příliš mnoho rizik. Avšak v případě projektů, začínajících na *zelené louce* (při těchto projektech architekt začíná svoji práci s prázdným papírem v ruce), či v případě projektů rozsáhlých změn existujících systémů může být tento přístup problematický, a to hned z několika důvodů:

- **Postup prací na projektu nelze přesně měřit,** neboť proces je založen na vytváření výsledků práce a nikoliv na dosahování výsledků. Například kompletace všech požadavků na časový stroj bez jakékoliv přípravy architektury, vývoje či testování vám nemůže poskytnout přesnou představu o době trvání projektu, protože na základě požadavků prostě nemůžete usoudit, zda vůbec existuje nějaké přijatelné řešení.
- **Zpětnou vazbu od uživatelů lze získat až ve velmi pozdní fázi projektu,** kdy je již systém připraven k užívání. Jenomže v důsledku toho je odloženo konečné přizpůsobení požadavkům, vyplývajícím z reálného provozu.
- **Řešení některých rizik je odloženo až do velmi pozdní fáze projektu,** do doby, kdy je systém již sestaven, integrován s dalšími systémy a otestován. Jenomže tyto činnosti mnohdy vedou k odhalení chyb v návrhu či dokonce chybně specifikovaných požadavků. Z tohoto důvodu jsou některé typy projektů, realizovaných sériovým procesem, velmi náchylné k termínovým skluzům.

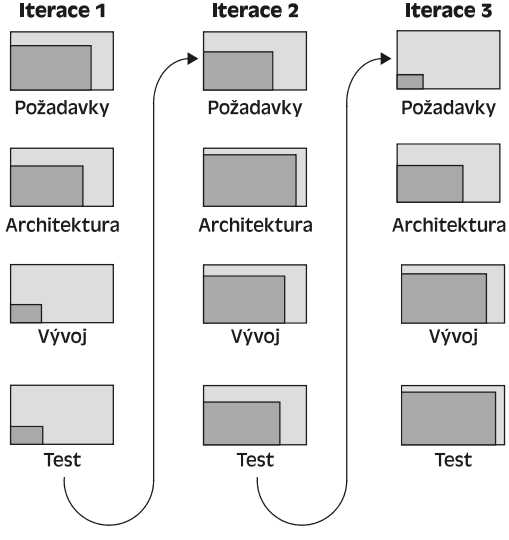
Alternativním přístupem je použití iterativního procesu, jemuž se budeme věnovat v následující části.

Iterativní proces

> *[Iterací je] Krátká část projektu, mající danou dobu trvání. Iterace umožňují demonstraci přírůstkové hodnoty a získání včasné a nepřetržité zpětné vazby. (OpenUP 2008)*

V rámci jedné iterace jsou provedeny všechny disciplíny, tedy požadavky, architektura, vývoj i test. Přitom *Iterací* je jasně definovaná a časově omezená posloupnost činností, vedoucích k uvolnění (internímu či externímu) spustitelného produktu. Z hlediska funkcionality se během postupu prací na projektu jednotlivé uvolňované verze stále zlepšují, přičemž až konečná verze je úplná. Iterativní proces vývoje se podobá rostoucímu softwaru, v němž konečný produkt během doby „zraje a dospívá". Každá iterace vede k lepšímu pochopení požadavků, robustnější architektuře, zkušenější vývojové organizaci a úplnější implementaci. Pojetí rostoucí architektury prostřednictvím řady iterací a verzí spustitelných souborů se zdá být vcelku běžné především v těch organizacích, které bychom z hlediska navrhování jejich vlastních softwarových systémů mohli považovat za produktivní a úspěšné.

Z obrázku 3.8 je zřejmé, jak se střed pozornosti projektu mezi jednotlivými úspěšnými iteracemi přesouvá. Jak vidíte, během každé iterace jsou řešeny všechny disciplíny, přičemž velikost tmavě vybarveného obdélníku v rámci každé disciplíny ilustruje relativní množství času, strávené prováděním souvisejících činností (a úkolů). Z naší jednoduché ukázky vyplývá, že iterace 1 je zaměřena především na definici požadavků, ale současně je věnováno i určité množství času tvorbě architektury (přičemž architekt se v této iteraci zaměřuje pouze na požadavky s nejvyšší prioritou), menší množství času pak je věnováno vývoji a testování. Iterace 2 se soustředí na stabilizaci architektury a z tohoto důvodu vidíte v našem schématu důraz na činnosti, související s architekturou. A konečně hlavním cílem iterace 3 je dokončení řešení na základě relativně stabilní sady požadavků a architektury. Proto jsou v této iteraci zdůrazněny činnosti, týkající se vývoje a testování.



Obrázek 3.8: Iterativní proces

Níže následuje přehled některých základních charakteristik iterací těch projektů, které lze z hlediska iterativního procesu považovat za úspěšné:

- Každou iteraci lze vyhodnotit na základě jasných kritérií.
- Cílem každé iterace je dosažení nějaké plánované funkcionality či schopnosti, kterou lze předvést.
- Každá iterace je zakončena nějakým méně významným milníkem, přičemž výsledek iterace je vyhodnocován relativně vůči objektivním kritériím úspěšnosti dané iterace.
- V průběhu iterace dochází k aktualizaci výsledků práce (výsledky práce se vyvíjejí spolu se systémem).
- V průběhu iterace je systém integrován a testován.

Iterativní vývojový proces je zajímavý především pro architekta, neboť tento přístup otevřeně a jasně říká, že v rámci postupu prací na projektu bude nutné provádět úpravy architektury. Současně však platí, že množství změn architektury by mělo s časem klesat. Důvodem je to, že tyto změny nevyplývají z nějakých dodatečných myšlenek; jedná se o základní součásti životního cyklu projektu.

Avšak součástí iterativního vývojového procesu je více než jen proud iterací. Tento proces musí mít i nějaký celkový rámec, v němž jsou jednotlivé iterace prováděny. Tento rámec současně představuje i strategický plán projektu a je zdrojem cílů pro každou iteraci. Popisovaný rámec je tvořen fázemi.

> *[Fází je] Specializovaný typ činnosti, představující významné období v projektu, obvykle zakončené dosažením nějakého rozhodujícího kontrolního místa, významného milníku či sady předmětů plnění. Fáze obvykle mají dobře definované cíle a jsou základem pro další strukturování práce na celém projektu. (OpenUP 2008)*

Fáze jsou také zdrojem dobře definovaných obchodních milníků, zajišťujících to, že iterace postupují vpřed a přibližují se k řešení namísto toho, aby jenom donekonečna vznikaly další a další iterace. Iterace jsou časově omezené (doba trvání jednotlivých iterací je tedy dána), zatímco fáze vycházejí z dosažení cílů. Fáze není časově omezená, protože dokončení fáze je vyhodnoceno na základě statusu projektu. Některé organizace si ovšem tento velmi zásadní princip neuvědomují, čemuž se věnuje naše poznámka „Úskalí: Předčasné vyhlášení vítězství".

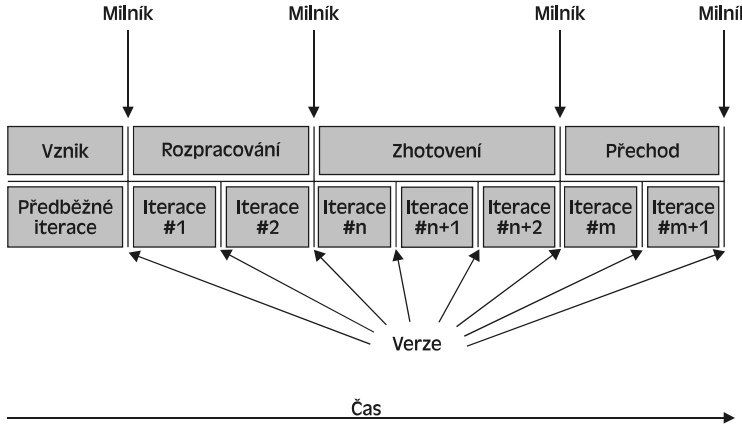
Úskalí: Předčasné vyhlášení vítězství

Velmi častou chybou **Projektového manažera** je vyhlášení vítězství tvrzením, že určitá fáze již byla dokončena, i když v realitě ještě dokončena nebyla; **Projektový manažer** obvykle tuto chybu dělá proto, že bylo dosaženo data, kdy by fáze měla být dokončena. Tento scénář lze v softwarovém průmyslu přirovnat k falšování účetnictví. Avšak **Projektoví manažeři**, kteří takto upravují „účetnictví" svých projektů, obvykle nedopadnou dobře, a to proto, že jejich chybná rozhodnutí se nakonec obrátí vůči nim.

Pro **Projektového manažera** je obvykle daleko lepší kousnout do hořké pilulky a přiznat to, že se danou fázi nepodařilo v plánovaném termínu dokončit, než klamat sebe samého a všechny ostatní tvrzením, že všechno pokračuje podle plánu – taková situace totiž obvykle pouze vede ke zbytečnému zvýšení tlaku na projekt.

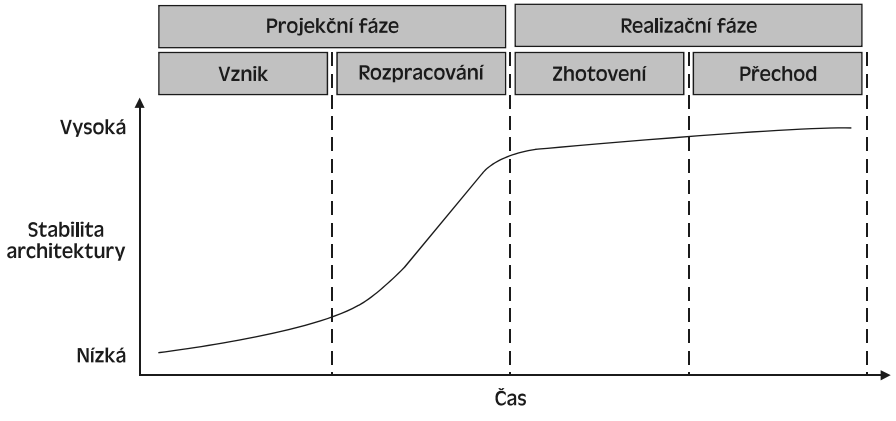
Fáze a iterace dohromady vytvářejí základ iterativního vývojového procesu. Cíle každé fáze jsou dosahovány provedením jedné či více iterací v rámci fáze. Každá fáze je zakončena nějakým významným milníkem a vyhodnocením, jehož cílem je určit, zda se podařilo dosáhnout stanovených cílů. Teprve na základě uspokojivého vyhodnocení pak projekt může přejít do další fáze.

OpenUP definuje celkem 4 po sobě následující fáze, které budeme v této knize používat: Vznik, Rozpracování, Zhotovení a Přechod (viz obrázek 3.9).



Obrázek 3.9: Fáze projektu

Přístup, založený na fázích, podporuje postupnou konvergenci několika prvků v průběhu projektu. Například rizika se po dobu života projektu snižují, zatímco odhady nákladů a termínů se zpřesňují. Z obrázku 3.10 vyplývá, že dochází i ke stabilizaci architektury. Většina procesů vývoje softwaru, vycházející z fází, obvykle obsahuje i nějakou speciální fázi pro stabilizaci architektury. V případě naší knihy je jí fáze Rozpracování, která jednoznačně má pro architekta zásadní význam.

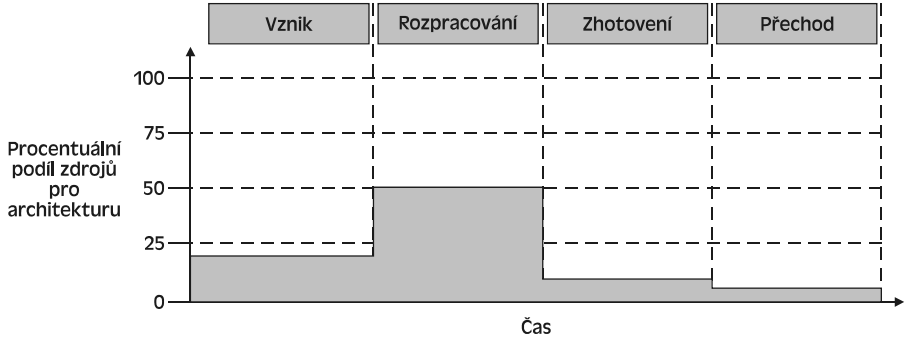


Obrázek 3.10: Stabilita architektury v průběhu času

Obrázek 3.10 také charakterizuje fáze Vznik a Rozpracování jako fáze projekční, zatímco fáze Zhotovení a Přechod popisuje jako fáze realizační. Toto rozdělení je podrobně popsáno v knize *Software Project Management: A Unified Framework* (Royce 1998). Zajímavý je především přechod z fáze Rozpracování do fáze Zhotovení (tedy z fáze projekční do realizační), neboť tento přechod představuje zásadní milník z hlediska obchodního. Obecně platí, že fáze realizační je lépe předvídatelná než fáze projekční, neboť v tuto chvíli již všichni dobře rozumí jak podstatě problému, tak i způsobu jeho řešení. Z toho vyplývá vyšší věrnost jakýchkoliv odhadů nákladů i termínů a v otázce financování lze tedy rozhodovat s daleko větší důvěrou.

Tím samozřejmě nechceme říci, že na konci projekční fáze jsou známé veškeré požadavky či že se požadavky nikdy nemění; rozhodně ale platí, že v tuto dobu by již měla být definována alespoň většina požadavků. Současně se předpokládá, že jakékoliv změny již budou mít pouze minimální dopad na architekturu. Pokud by tomu tak nebylo (například v případě přidání nějakého zásadního požadavku), je samozřejmě nutné vyvolat jednání s cílem kontroly projektu a jeho časového plánu a uvážit případné změny tohoto plánu. Může se ovšem stát, že toto jednání povede k opakovanému spuštění projektu od začátku, tedy i k opakování fáze vzniku a rozpracování.

Z obrázku 3.11 vyplývá, že v každé z fází existuje přímá vazba mezi důrazem na architekturu a množstvím zdrojů, věnujících se architektuře (ve skutečnosti tento údaj samozřejmě značně závisí na podstatě vyvíjeného systému). Uvedený graf je odvozen z informací, publikovaných v knize *Software Project Management: A Unified Framework* (Royce 1998).



Obrázek 3.11: Procentuální podíl zdrojů, zabývajících se v průběhu času architekturou

Je zřejmé, že přechod z jedné fáze do druhé představuje v životním cyklu projektu významnou událost. Proto je naprosto nezbytné porozumět kritériím, podmiňujícím zmíněný přechod z jedné fáze do druhé. Podrobný přehled primárních cílů každé z fází a kritérií pro vyhodnocení souvisejících milníků najdete v příloze C „Přehled metody". Níže následuje stručná charakteristika každé z fází:

- **Vznik** je tou fází, v níž je vytvořen či definován obchodní případ projektu a v níž musí mezi všemi vlastníky dojít k dosažení souhlasu o cílech projektu. **Vznik** je fází, v níž je důraz kladen především na zajištění jak smyslu či hodnoty projektu, tak i jeho realizovatelnosti.
- Ve fázi **Rozpracování** je vytvořena architektura, která má zajistit stabilní základ pro činnosti, prováděné v rámci fáze **Zhotovení**. To znamená, že toto je fáze, která je pro architekta nejzajímavější: během ní bude muset vynaložit nejvíce svého úsilí.
- V rámci fáze **Zhotovení** dochází jak k ujasnění definice zbývajících požadavků, tak i k dokončení vývoje systému na základě architektury, připravené během fáze **Rozpracování**. Mezi fázemi **Rozpracování** a **Zhotovení** se pozornost přesouvá z porozumění problému a identifikace klíčových prvků řešení k vývoji nasaditelného produktu.
- Smyslem fáze **Přechod** je zpřístupnit vyvinutý software koncovým uživatelům a zajistit jeho přijetí. Toto je tedy fáze, během níž je software nasazován do prostředí uživatelů pro účely vyhodnocení a testování. Hlavní pozornost se soustředí na doladění produktu a na vyřešení veškerých otázek souvisejících s konfigurací, instalací a použitelností. Na konci popisované fáze by se projekt měl blížit okamžiku, kdy jej bude možné uzavřít.

Iterativní a sériový vývoj

Na rozdíl od sériového procesu vývoje softwaru, vycházejícího z dosahování výsledků práce, je iterativní proces založen na dosahování výsledků (jinými slovy řečeno, cílem je dosažení určitých výsledků v rámci dané iterace). Z tohoto důvodu také obvykle poskytuje přesnější měřítka pro hodnocení postupu prací na projektu. Iterativní přístup umožňuje brzké získání zpětné vazby od uživatelů, a to proto, že uživatelé mohou pracovat s přírůstkovými vydáními systému. Takto získaná zpětná vazba urychluje přizpůsobení systému reálným požadavkům. Iterativní přístup zajišťuje také to, že v rámci každé fáze dochází k integraci a testování a že výsledkem každé fáze je vytvoření spustitelné verze softwaru. Rizika jsou řešena již v časném období projektu, čímž je podporován proces rozhodování ještě před tím, než jsou do projektu investovány významné částky. Jak Royce říká:

> *Vytvořte iterativní proces po celou dobu životního cyklu, včas řešící jakákoliv rizika. U soudobých sofistikovaných softwarových systémů nelze definovat celý problém, navrhnout celé řešení, sestavit software a poté jej jako celek otestovat v krocích, následujících postupně za sebou. Namísto toho vyvážené zpracování všech cílů investorů podporuje iterativní proces, zlepšující porozumění problému, umožňující jeho efektivní řešení a rozplánování celého projektu do několika iterací. Hlavní rizika musí být řešena včas, aby se zvýšila předvídatelnost a snížilo riziko vzniku odpadu a oprav. (Royce 1998)*

Agilní proces

V posledních letech také vzrostl zájem o agilní procesy, popsané metodami jako například eXtreme Programming (XP), Scrum, Lean a Feature-Driven Development. Ačkoliv mezi jednotlivými agilními metodami a postupy, které tyto metody obhajují, existují určité rozdíly, všechny vycházejí ze shodných základních principů. Jedním ze známých vyjádření těchto principů je *Agile Manifesto* (Agile Manifesto 2009), uvádějící tyto principy:

- Jednotlivci a interakce mají přednost před procesy a nástroji
- Funkční software má přednost před vyčerpávající dokumentací
- Spolupráce se zákazníkem má přednost před vyjednáním kontraktu
- Reakce na změnu má přednost před postupováním podle plánu

Autoři této knihy se domnívají, že zmíněné principy doplňují ty, z nichž vychází iterativní proces vývoje softwaru. Z Agile Manifesto také zřetelně vyplývá, že hlavním rozdílem je důraz, kladený na principy, používané pro řízení procesu, a nikoliv nový či změněný obsah metody. Důkaz přináší například Scrum:

> *Scrum je proces vedení a řízení, protínající složitost a zaměřující se tak na tvorbu softwaru, vyhovujícího obchodním potřebám. Scrum se nachází nade všemi stávajícími postupy, vývojovými metodikami a standardy, které také obsahuje. (Schwaber 2002)*

Scrum je také důkazem důrazu, kladeného na proces. Zaměřuje se na obsah iterace (přičemž iteraci označuje pojmem *Sprint*), seznam prioritizovaných požadavků, které by měly být v rámci iterace řešeny (jedná se o tzv. *Sprint Backlog*), a krátké každodenní jednání o stavu projektu (nazývaného *Scrum*).

Agilní principy se týkají i architektů a je překvapující, jak často jsou tyto principy špatně interpretovány. Velmi častý náhled je ten, že agilní procesy neobhajují návrh, prováděný předem; že architektura jaksi sama vyplyne z kódu. Princip přednosti funkčního softwaru před vyčerpávající dokumentací však rozhodně neznamená, že neexistuje vůbec žádná dokumentace (tedy včetně do-

kumentace, týkající se architektury); tento princip pouze znamená, že existuje taková dokumentace, která je pro účely aktuální iterace dostatečná.

> *Zdá se, že v komunitě vývojářů, vyznávajících principy agilního procesu, existuje obava, že jakmile začneme používat pojmy typu „model" či „dokument", tak najednou „zlí byrokraté" zatnou svoje paráty do našich projektů a donutí nás psát podrobné, rozsáhlé specifikace požadavků či vytvářet rozsáhlé návrhy předem … podivné na tom ovšem je, že agilisté ve skutečnosti modelují pravidelně, i když o tom přímo nemluví. (Ambler 2008)*

Shrnutí

Tato kapitola vám nabídla stručný přehled základů metody, používané v této knize. Pro vysvětlení těchto konceptů jsme použili rámec standardu SPEM (Software and Systems Process Engineering Metamodel Specification). V textu této kapitoly jsme se zaměřili především na rozdíl mezi obsahem metody a procesy, stanovujícími tento obsah. Současně jsme se zabývali různými typy procesů, včetně sériového, iterativního a agilního. Věnovali jsme se důležitosti těchto konceptů pro architekta a pro proces samotného navrhování. Metoda, se kterou budeme v následujících kapitolách této knihy pracovat, vychází ze všech zmíněných typů procesů.

Následující dvě kapitoly – kapitola 4 „Dokumentace architektury softwaru" a kapitola 5 „Opětovně použitelné prvky architektury" – se zaměřují na specifické aspekty metody, zaručující důkladnější a podrobnější diskuzi a prostupující celým životním cyklem projektu.