STRUKTUROVÁNÍ PRVKŮ METODIK TVORBY SOFTWARU

Robert Pergl

Česká zemědělská univerzita, Provozně ekonomická fakulta, Katedra informačního inženýrství

pergl@pef.czu.cz

ABSTRAKT:

V současné době existuje celá řada metodik tvorby softwaru, klasických, agilních i různých speciálních firemních. Ne všechny metodiky však kladou důraz na všechny aspekty vývoje softwaru, typická je specializace pro různé podmínky a charakter projektů. Tento příspěvek ukazuje, jakým způsobem je možné provést strukturování prvků metodik pro potřeby porovnání, klasifikace, kvantifikace a dalšího zpracování (např. uzpůsobení metodiky či obohacení prvky jiné metodiky, atd.).

KLÍČOVÁ SLOVA:

softwarové inženýrství, metodiky tvorby softwaru, agilní metodiky

ÚVOD

Hlavní výzvou softwarového inženýrství je dnes značná variabilita charakteru projektů. Softwarové inženýrství je inženýrská disciplína s nejdramatičtějším vývojem vůbec. Za oficiální počátek tohoto oboru můžeme považovat konferenci NATO o softwarovém inženýrství r. 1968. Za necelých 40 let prodělalo softwarové inženýrství značný vývoj. Za tuto skutečnost vděčí neuvěřitelnému tempu vývoje hardwaru, který velmi rychle rozšiřuje možnosti využití výpočetní techniky. Postupně padají technologická a implementační omezení a hlavním omezením při tvorbě softwaru se stává schopnost definovat, co by měl program dělat, program dobře navrhnout a uřídit jeho vývoj.

V současné době je vývoj softwaru značně různorodou oblastí, do které spadají např. vývoj systémových aplikací pro jednočipové počítače, značně se rozvíjející vývoj aplikací pro mobilní zařízení, technologické aplikace a řízení výroby, aplikace pro podporu vědy a výzkumu, webové aplikace, „krabicový“ software, software na zakázku pro malé firmy, sofware na zakázku pro velké firmy.

Každá tato oblast má svá specifika z hlediska zaměření, technologických otázek, marketingových požadavků na výsledný produkt i tržních principů. Při realizaci projektu navíc u všech kategorií figuruje celá řada dalších parametrů jako např. rozsah projektu, zkušenost týmu, kritické oblasti projektu (bezpečnost/rychlost/uživatelská přítulnost, atp.)

Všechny tyto aspekty musí zohledňovat i metodika řízení projektu. V současné době existuje řada metodik pro řízení softwarových projektů, klasických i agilních. Z hlediska výše popsaného můžeme prohlásit, že metodiky můžeme rozdělit na:

- obecné metodiky,
- metodiky zaměřené na určitý aspekt vývoje.

Obecné metodiky typu RUP či agilní DSDM mají často charakter rozsáhlých průmyslových metodik. Metodiky zaměřené na určitý aspekt vývoje se snaží adresovat nějaký důležitý prvek řízení. Příkladem může být metodika Lean Development zaměřující se na maximální efektivitu vývojového procesu či Adaptive Software Development pracující s dynamickým vývojovým cyklem. Použití těchto specializovaných metodik však často brání absence určitých praktik či vývojových fází. Poté je třeba sáhnout do jiných metodik či vlastních zkušeností. Podobně v případě obecných metodik mohou nastat situace, kdy je třeba vývojový proces obohatit o určitý prvek (např. formální dokumentace u procesu vedeného extrémním programováním). V takových případech je třeba se umět zorientovat v metodologii softwarového inženýrství.

FÁZE TVORBY INFORMAČNÍHO SYSTÉMU

Obecně se každý proces tvorby informačního systému skládá ze čtyř životních fází:

1. Příprava (Initiate)
2. Vývoj (Construct)
3. Nasazení (Deliver)
4. Správa a podpora (Maintain and Support)

Dle charakteru projektu mají jednotlivé fáze různou míru důležitosti a zabírají různý časový i finanční prostor:

- **Příprava:** důkladná příprava je nutná především u rozsáhlých projektů se složitou infrastrukturou. Rovněž tak u projektů, kde jsou očekávána zvýšená rizika. Investice do přípravné fáze se však musí vyplatit, jinak nemá cenu se do projektu pouštět.
- **Vývoj:** fáze vývoje relativně převažuje ostatní fáze u malých až drobných projektů, kde ostatní fáze je možné minimalizovat.
- **Nasazení:** nasazení je třeba věnovat zvýšenou pozornost v případě, kdy podmínky provozování jsou nestandardní či komplikované, ať už z hlediska technologického (např. extrémní podmínky provozu), procesního (začlenění do procesu firmy) či uživatelského (např. uživatelé mají minimální praxi s výpočetní technikou).
- **Správa a podpora:** míra důrazu na správu a podporu záleží především na
    - *míře využívání aplikace*: příležitostná utilita nebude vyžadovat intenzivní správu a podporu
    - *kritičnosti aplikace*: kritické aplikace typu řízení leteckého provozu vyžadují perfektně nastavené mechanismy správy a podpory.
    - *předpokládané délce života aplikace*: správa a podpora vyžaduje zdroje. Pokud očekáváme, že aplikace bude mít omezenou dobu používání, je neekonomické investovat prostředky do rozsáhlé správy a nových verzí.

Ve složitějších případech dochází k tomu, že jednotlivé fáze se navzájem ovlivňují a prolínají.

Z důvodu omezeného rozsahu příspěvku se budu zabývat detailněji pouze prvníma dvěma fázemi. Praktiky pro fáze nasazení a údržby lze podobným způsobem shromáždit z doporučené literatury.

PŘÍPRAVA

Ve fázi přípravy jsou prováděny tyto činnosti:

- **Definice a ověření počátečních požadavků**. Různé metodiky mají svůj specifický přístup ke shromáždění, dokumentaci a ověření uživatelských požadavků. Přístupy se liší především důkladností, mírou podrobnosti a formalismu. Některé příklady struktury uživatelských požadavků (detaily lze nalézt v literatuře):
    - kontextový diagram (context diagram) [14],
    - případy užití (use-cases) [14],
    - CRC modelování [5],
    - User stories (metodika Extrémní programování) [2],
    - Product backlog (metodika SCRUM) [15],
    - Feature (metodika Feature Driven Development) [12],

  Pro získání uživatelských požadavků jsou používány různé techniky, např.
    - interview,
    - JAD Sessions: zkratka Joint Application Development. Označuje koordinovaný meeting, kdy si uživatelé a vývojáři navzájem ujasňují požadavky pomocí společného modelování.
    - prototypování uživatelského rozhraní.

  Podrobněji o problematice uživatelských požadavků viz [8].
- **Klasifikace a prioritizace počátečních požadavků.** Požadavky je třeba klasifikovat (viz např. [9]) a pokud není možné z důvodu omezených časových a finančních zdrojů uspokojit naráz všechny požadavky, je třeba, aby byly přiřazeny k požadavkům priority. Priority určuje primárně zadavatel ve spolupráci s dodavatelem, jelikož mezi požadavky mohou být technické a organizační závislosti. Některé metodiky mají systémy prioritizace požadavků, které jsou obecně založeny na ordinálních stupnicích:
    - Extrémní programování: očíslování User stories
    - Dynamic Software Development Method: technika MoSCoW (Must-Could-Should-Won't)

  Existují i samostatné techniky pro prioritizování požadavků, např. Kanova analýza [7].
- **Posouzení projektu**. Ne každý projekt je rozumné realizovat. Některé projekty mohou být ekonomicky ztrátové a mohou přinést značné problémy dodavateli i zákazníkovi. Statistiky úspěšnosti softwarových projektů ukazují, že takové projekty nejsou výjimečné (statistiky Standish i ostatní). Proto je zvláště u rozsáhlejších projektů vždy třeba posoudit, jestli je vhodné projekt realizovat. K tomu slouží především **studie proveditelnosti** (Feasibility Study) [10]. Skládá se z:
    - identifikace alternativ realizace,
    - studie operační proveditelnosti,
    - studie ekonomické proveditelnosti,
    - studie technické proveditelnosti,
    - výběru alternativy či zamítnutí projektu.

- **Definice dokumentace projektu**. Je třeba definovat obsah projektové dokumentace a zvolit vhodnou formu. Jedná se především o:
    - plán projektu: rozsah, úkoly, odhady, atp.,
    - harmonogram projektu,
    - dokumentace řízení rizik,
    - plán zajištění kvality (Quality Assurance),
    - dokumentace vývojového procesu,
    - manažerské statistiky a reporty.

  Dokumentace se obecně dělí na dvě kategorie:
    - *výstupní* (deliverables): je předávána v průběhu projektu (např. průběžné zprávy) či na jeho konci (např. uživatelská dokumentace)
    - *interní*: slouží jako „paměť projektu“ pro potřeby řízení i pozdějšího vyhodnocování projektu.

  Těžší metodiky typicky předepisují více dokumentace a kladou důraz na formální stránku. Lehké metodiky jdou cestou účelnosti a ponechávají volnost ve výběru množství a struktury dokumentace. [17]
- **Definice infrastruktury projektu**. Před zahájením projektu je třeba vytvořit potřebnou infrastrukturu. Ta se bude velmi lišit především v závislosti na rozsahu projektu. Obecně do infrastruktury patří**:**
    - definice projektového týmu,
    - subcontracting,
    - úprava softwarového procesu,
    - komunikace v projektovém týmu a se zákazníkem: na komunikaci kladou důraz agilní metodiky [4],
    - specifikace standardů, směrnic a metodických pokynů pro tým týkajících se modelování, konvencí pojmenovávání a štábní kultury, přístupu k testování, designu uživatelského rozhraní, atd.
    - výběr nástrojů: nástroje pro psaní kódu, sdílení kódu, tvorbu dokumentace, vzájemnou komunikaci, atd.

VÝVOJ

Do fáze vývoje se vstupuje ve chvíli, kdy je jasné:

- Co se bude vyvíjet.
- Plán vývoje.
- Co bude k vývoji třeba a jak bude organizován.

Ve fázi vývoje vzniká vlastní produkt. V nejjednodušších případech vývoj sestává z programování, obecně však tato fáze může být značně komplexní a programování v ní může hrát i méně kritickou úlohu, jelikož se do značné míry jedná o mechanický postup.

Vývoj může dle typu vývojového cyklu (software development life cycle) probíhat jako sekvenční či iterativní proces [5].

Vývoj sestává z těchto navzájem úzce provázaných základních činností:

- analýza,
- návrh,
- implementace,
- testování.

K tomu navíc probíhá řada podpůrných procesů, jako

- správa verzí,
- řízení týmu,
- dokumentace,
- řízení znovupoužitelnosti.

Podle charakteru metodiky jsou jednotlivé činnosti prováděny s různou granularitou. Tradiční, těžké metodiky provádějí důkladnou komplexní a kompletní analýzu, návrh, implementaci a poté hotový produkt testují. Agilní přístupy oproti tomu nastavují co nejkratší trvání těchto činností a co nejvíce iterací. Důvodem jsou kalkulace se změnami zadání, minimalizace nepochopení zadání a eliminace dalších rizik.

Komplexní **analýza** je prováděna tímto způsobem:

1. modelování,
2. identifikace funkčních celků,
3. rozhodnutí o znovupoužití/nákupu/vývoji celku.

Z hlediska efektivity vývoje a ušetření nákladů je ideální:

1. Použít co nejvíce modulů, které jsou hotovy (znovupoužitelnost),
2. získat či nakoupit moduly, které jsou k dispozici, a to na co nejvyšší úrovni, tedy postupně:
    - hotová aplikace k upravení (která je k dispozici např. ve formě open-source),
    - framework,
    - moduly,
    - rutiny.
3. zbytek vyvinout vlastními silami.

Vždy je ale samozřejmě třeba zvážit jestli integrace a úprava určitého hotového prvku nebude představovat větší rizika než vlastní vývoj. Rizika je třeba hodnotit komplexně, tedy

- pracnost úpravy a integrace,
- technologická rizika (kompatibilita),
- spolehlivost dodavatele hotového řešení a podpora.

Modelování dělíme na:

- *podnikové modelování*, kdy modelujeme širší kontext v podniku. Příkladem metodiky pro podnikové modelování je BORM [3] či ARIS.
- *doménové modelování* modelující problémovou doménu, pro kterou je produkt používán,
- *modelování architektury* produktu jako celku,
- *detailní implementační modelování* částí produktu.

Modelování je komplexní problematika, které je věnována řada literatury a diskusí. Existuje řada metodik, které zahrnují jeden či více typů modelování zmíněných výše. Z hlediska notace je v současné době průmyslovým standardem objektově orientované UML [14], které je používáno v rámci metodiky Unified Process i agilních metodik.

Jednotlivé metodiky poté doporučují různý přístup k modelování s tím, že obecně opět platí, že těžší metodiky kladou větší důraz na modelování a jeho formální stránku, zatímco lehčí metodiky ponechávají modelování pouze jako doporučený nástroj pro analýzu. Výjimkou je metodika Feature Driven Development [12], která je řízena modelováním i přesto, že ji řadíme mezi agilní metodiky.

Za zmínku ještě stojí metoda agilního modelování vyvinutá Scottem Amblerem [1], která uvádí pravidla efektivního modelování použitelná v rámci obecné metodiky.

Další činností ve fázi vývoje je **implementace**, které se v širším chápání skládá z následujících činností:

- psaní funkčního kódu,
- integrace hotových komponent, a to
  - interně vytvořených v rámci předchozích projektů (znovupoužitelnost),
  - koupených,
- návrhu funkčnosti uživatelského rozhraní,
- návrh grafického designu pro program, dokumentaci, krabici, atd.,
- psaní dokumentace:
  - interní dokumentace postupu prací,
  - kontextová nápověda aplikace,
  - příručka uživatele a další dokumenty pro uživatele (FAQ, řešení problémů, atp.),
- tvorba plánu migrace starých dat do nové aplikace a vývoj potřebných utilit.

Testování bývá v tradičních metodikách prováděno po programování, v současné době se však prosazuje tzv. *test-driven development* přístup, který se vyznačuje následujícím:

- Testovány jsou co nejmenší celky (jednotlivé metody).
- Testy jsou psány před implementací, což umožňuje ujasnit si, co bude testovaná část přesně dělat.
- Testy jsou psány, spouštěny a vyhodnocovány s pomocí podpůrných nástrojů.

Test-driven development je přímo součástí některých metodik, např. extrémního programování.

Kromě testování funkčnosti aplikace může být třeba začlenit i další typy testů, např. zátěžové testy, operační testy, testy uživatelské dokumentace.

Na závěr bývá prováděno komplexní *uživatelské testování* (alpha-testování, beta-testování, pilotní testování).

NASAZENÍ, SPRÁVA A UŽIVATELSKÁ PODPORA

K fázi nasazení se přistupuje v okamžiku, kdy je celý release systému (tj. část systému k dodání pokud bude systém dodáván inkrementálně) připraven k nasazení. V nejjednodušším případě je systém pouze předán k používání, obecně však řešíme tyto úkoly (z důvodu rozsahu článku již jen heslovitě):

- Testování celého systému jako celku a oprava případných defektů.
- Příprava k předání
    - Školení a tréninky uživatelů.
    - Migrace starých dat.
- Předání
    - Instalace, zprovoznění.
    - Trénink uživatelské podpory.
    - Nastartování podpůrných procesů.
- Odstartování používání
- Vyhodnocení projektu.

Po zbytek života aplikace je po nasazení prováděna správa a uživatelská podpora sestávající z řešení operativních problémů a identifikace podnětů pro další verzi (software change management, SCR).

ZÁVĚR

V článku byly představeny činnosti a praktiky, které se objevují v současné praxi softwarového inženýrství. Každá z nich má charakter propracovaného a osvědčeného „best-practice“. Ne vždy jsou ale všechny prováděny, a to obecně z těchto důvodů:

- Vzhledem k charakteru projektu nejsou třeba.
- Neobsahuje je používaná metodika.
- Vývojový tým praktiku nezná.

Pokud první bod není pravda, tj. určitá praktika by přinesla projektu užitek, je vhodné uvažovat o začlenění prvku do procesu vývoje i přesto, že používaná metodika s ním nepočítá. Při tom je samozřejmě nutné značné obezřetnosti, jelikož většinou nestačí pouze začít něco provádět, ale je nutné prvek organicky začlenit do metodiky. Obecně je třeba provést tyto kroky:

1. Identifikace potřebného prvku.
2. Analýza vazeb na zbytek používané metodiky.
3. Začlenění prvku do metodiky.

4. Úprava metodiky.
5. Vyhodnocení vlivu nového prvku.
6. Úprava metodiky.

Citlivě provedené obohacení metodiky může přinést zlepšení řady parametrů vývojového procesu v konkrétní situaci, stojí tedy za to investovat čas a snahu na seznámení s možnostmi, které nabízí softwarové inženýrství.

LITERATURA

[1] Ambler, S. W.: „The Object Primer – Agile Model-driven Development with UML 2.0“, Cambridge University Press 2005, ISBN 978-0-521-54918-6

[2] Beck K.: „Extrémní programování“, Grada 2002, ISBN 80-247-0300-9

[3] Merunka V., Polák J. ,Carda A.: Umění systémového návrhu. Praha: Grada Publishing 2003, ISBN 80-247-0424-2

[4] Buchalcevová A.: „Agilní metodiky“. Sborník konference Objekty 2002, ISBN 80-21360947-4

[5] Ambler S.,W.:„Process Patterns : Building Large-Scale Systems Using Object Technology“, Cambridge University Press 1998, ISBN 0-521-64568-9

[6] Ambler S.,W.:„ More Process Patterns : Delivering Large-Scale Systems Using Object Technology“, Cambridge University Press 1999, ISBN 0-521-65262-6

[7] Kanova analýza: http://www.pragmaticmarketing.com/productmarketing/magazine/4/3/0605ss.asp

[8] Thayer R.H. et al.: *Software Requirements Engineering,* Wiley-IEEE Computer Society Pr; 2 Sub edition, 1997, ISBN 0818677384

[9] Metoda klasifikace uživatelských požadavků FURPS+: http://www-128.ibm.com/developerworks/rational/library/4706.html

[10] Keyes J.: *Software Engineering Handbook*, AUERBACH, 2002, ISBN 0849314798

[11] Beck K.: *Test Driven Development: By Example***,** Addison-Wesley 2002, ISBN 0321146530

[12] Buchalcevová, A.: *Metodika feature-driven development neopouští modelování a procesy, a přesto přináší výhody agilního vývoje*, sborník konference Tvorba softwaru 2005, Ostrava

[13] Jacobson I., Booch G., Rumbaugh J.: *The Unified Software Developement Process*, Addison Wesley, 1999, ISBN 0-2015-7169-2

[14] Rumbaugh, I. Jacobson, G. Booch: *The Unified Modeling Language Reference Manual,Addison*-Wesley, 1998, ISBN 0-2013-0998-X

[15] Schwaber K., Beedle M.: *Agile Software Development with SCRUM*, Prentice Hall 2001, ISBN 0130676349

[16] Pergl, R., Struska, Z.: *Agilní modelování a metodika BORM.* Tvorba software 2006. Ostrava 2006. ISBN 80-248-1082-4

[17] Poppendieck M., Poppendieck T.: *Lean Software Development: An Agile Toolkit for Software Development Managers*, Addison-Wesley 2003, ISBN 0321150783