Poznámky k vedení větších projektů vývoje software

Tomáš Smolík

Profinit, s.r.o.
Tychonova 2, 160 00 Praha 6
tomas.smolik@profinit.cz

Abstrakt
Tento článek v první části ukazuje, že mnozí lidé odpovědní za vedení vývoje sw cítí neuchopitelnost všezahrnujících standardů, metodik atd. a dožadují se omezeného počtu konkrétních praktik pokud možno bezprostředně použitelných. Dále ukazuje, že k základní myšlenkové výbavě pro plánování, organizaci a řízení vývoje větších sw systémů neodmyslitelně patří vědomé používání modelu vývoje životního cyklu sw (SDLC), struktury sw systému (tj. sw architektury), průběžného a postupného plánování, a je-li to potřeba, rekurzivní aplikaci modelu sdlc. Po úvodu do tématu je poskytnut popis dobré praxe vedení sw projektů, který akcentuje fakt, že se jedná o vývoj softwarových systémů.

Abstract
This paper illustrates that, in spite of existence all round standards, de facto standards or methods for sw project development, there is a need for basic set of software development management practices which are straightforwardly applicable. Further, the text contends that the mental instruments to manage software development must include the following phenomenon: software development life cycle (SDLC) models, sw system architecture, two-tier approach to the sw planning and, if necessary, this all in a recursive manner. Having been these themes introduced, described and explained is the set of "best" management practices for the sw system development focused on intrinsic software development problems.

Klíčová slova
vedení sw projektu, architektura sw, model životního cyklu vývoje sw

Keywords
sw project management, sw architecture, SDLC model

1 Úvod

Na úvod se podívejme na genezi textu [Brown98]. Nepochybně lze říci, že vývoj software se snaží uceleně uchopit, tj. popsat anebo předepsat, celá řada více méně propracovaných standardů, *de facto* standardů, metodik atd. Příkladem řekněme z posledních 20 let mohou být [ISO9000-3, ISO12207, Paulk93, SPICE95, MIL-STD-498, DOD-STD-2167A, ESA91, BSI95, Jacobson]. Tyto mají různý původ, míru detailu, základní koncepci atd., jedno mají však společné: je velmi těžké je skutečně do důsledků aplikovat v praxi. Co mají tedy ti, kteří vedou projekty vývoje software v prostředí reálného světa, dělat, když nemají reálnou šanci vyvíjet např. na úrovni 3 CMM [Paulk93], ale přesto chtějí v rámci svých podmínek vést vývoj „nějak“ dobře, resp. vyvarovat se základních a hrubých chyb. Zhruba tyto úvahy a tyto stesky vedly v rámci americké armády ke vzniku nejprve Software Program Managers Network, poté Software Acquisition Best Practices Initiative[1], která formulovala tzv. Principal Best Practices a Best Practices publikované právě v [Brown98]. Shrnuto, mnozí odpovědní za vedení vývoje sw cítí neuchopitelnost všezahrnujících standardů, metodik atd. a dožadují se

[1] Např. články *Industrial-Strength Management Strategies* (http://www.stsc.hill.af.mil/crosstalk/1996/08/industri.asp), *High-Leverage Best Practices — What Hot Companies are Doing to Stay Ahead and How DoD Programs Can Benefit* (http://www.stsc.hill.af.mil/crosstalk/1999/10/brown.asp) v časopise Crosstalk. Současný stav těchto aktivit lze získat zde (http://www.thedacs.com/).

omezeného počtu konkrétních praktik pokud možno bezprostředně použitelných. Text [Brown98] obsahuje 9 takových konkrétních praktik[2]. Ještě dodejme, že [Brown98] je sice jen jedna z mnoha příruček pro vedoucí projektů vývoje sw americké armády, svým charakterem je však narozdíl od ostatních ojedinělá.

Proč tento článek? Zjednodušeně lze říci, že ač výše zmíněné Principal Best Practices pro vedení vývoje software jsou inherentně svázány s tím, že se jedná o software, tak přece jen se více zaměřují (zdůrazněno kursivou) na *vedení vývoje* software, kdežto tento článek sleduje především vedení vývoje *software*.

Struktura textu je následující: Sekce č. 2 velmi stručně popisuje podstatu některých principiálních obtíží při vedení sw projektu. Sekce č. 3 uvádí přehled toho, co není vhodné ignorovat pro vedení sw projektu. Konečně sekce č. 4 obsahuje určitý "návod" pro dobrou praxi vedení vývoje sw následující formou: Nejprve je uvedena deklarace praktiky vedení sw projektu, viz sekce č. 4.1, pak je podrobněji popsáno 8 důležitých oblastí, na které je nutno se při vedení vývoje sw soustředit, viz sekce č. 4.2.

2 Co je na vedení větších sw projektů taky z podstaty věci složité

Zkusme nyní odhlédnout od nepřeberného množství rizik a různých problémů, na které nutně při vedení většího sw projektu narazíme. Odhlédněme také od klasických problémů typu změna požadavků při vývoji, řízení rozsahu atd. Zaměřme se výhradně na otázku, jak vůbec zorganizovat vývoj většího a složitějšího sw systému. Nejprve si uveďme "základní" metodu[3] vedení vývoje sw, která spočívá v:

i) ustavení organizace projektu,

ii) zjištění a porozumění „práci co se má dělat“,

iii) (postupném) rozložení „celé práce, co se má dělat“ na jednotlivé úkoly, až jsou identifikovány dostatečně kompaktní jednotky práce, díky a vzhledem k identifikovaným úkolům a jednotkám práce provést odhady (size, effort, cost), přiřadit omezení, nároky na zdroje, provést analýzu závislostí atd. Výsledek je, že existuje WBS, harmonogram, activity network a dále různé možné odvozené věci typu profil nároků na personál atd.

iv) přiřazení zdrojů,

v) měření, monitorování, hlášení a zaznamenávání postupu; přijímání nápravných opatření vzhledem k iii a iv,

vi) zaznamenání a uložení údajů o projektu pro další použití.

Elegantní. Přesto, představme si dále, že umíme odhadovat, monitorovat, známe všechny činnosti sw inženýrství a jejich aspekty a umíme je pro projekt plánovat, máme vzory plánů, výstupů, co mají vznikat. Stále něco chybí. Chybí koncepce, jak postupně rozložit "celou práci, co se má dělat" na jednotlivé dobře definované úkoly. Pravda je zde až bolestně prostá, pokud nemám strukturu systému, který vyvíjím, pak není vzhledem k čemu rozdělovat práci, nejde-li zrovna o analýzu a specifikaci na dané úrovni dekompozice systému. Proto, aby byla definována na dané úrovni dekompozice struktura systému, tak se musí navrhnout, a aby se mohla navrhnout, tak je potřeba znát požadavky relevantní

[2] Jsou to následující Principal Best Practices: 1. Formal Risk Management, 2. Agreement on Interfaces, 3. Formal Inspections, 4. Metrics-based Scheduling and Management, 5. Binary Quality Gates at the Inch-Pebble Level, 6. Program-wide Visibility of Progress vs. Plan, 7. Defect Tracking Against Quality Gates, 8. Configuration Management, 9. People-Aware Management Accountability.

[3] Pozn. body (i) až (vi) nejsou míněny *a priori* sekvenčně. Tuto „základní“ metodu vedení lze v různých podobách a vyjádřeních nalézt např. v NASA příručce *Software Management Guidebook*, STSC *Report on Project Management*, SPICE, CMM*, Software Project Management Curriculum Module* (od SEI), ISO 12207 atd.

pro strukturu systému na dané úrovni dekompozice. A aby toto všechno mohlo být, tak se to musí naplánovat, zorganizovat a provést. Zdánlivě jsme tam, odkud jsme v úvahách vyšli. Zde docházíme k poznání, že pro organizaci a vedení vývoje sw jsou podstatná následující témata, jejich vzájemné vztahy, jejich pochopení a vědomé využití při vedení vývoje sw:

- "základní" metoda,
- názor na uspořádání mezi: kalendářním časem vs. dekompozičními jednotkami sw systému vs. primárními činnostmi sw inženýrství, tj. model životního cyklu vývoje software (SDLC),
- struktura sw systému, tj. architektura sw systému,
- průběžné a postupné plánování,
- rekurzivní aplikování "základní" metody.

Následující části článku rozvádějí v této sekci nastolená témata.

3 Co je vhodné neignorovat pro vedení projektů vývoje sw

3.1 Nutná "encyklopedická" výbava

Pro vedení projektu vývoje sw je minimálně nutné mít představu o následujících tématech:

- standardní pojednání vedení sw projektu, viz např. [Metzger96],
- praktické pojednání vedení sw projektu, viz např. [NASA96],
- co je to softwarové inženýrství, viz např. [SWEBOK],
- co je to software process improvement, viz např. [SEL95],
- co je to softwarová architektura, viz např. [Boehm95a],
- co je to model životního cyklu vývoje systému, viz např. [Boehm95b],
- jaké jsou základní standardy, *de facto* standardy, metodiky pro vývoj software; viz sekce č. 1.

Bez výše uvedeného vlastně člověk nemá ani pojmovou výbavu, jak o vývoji sw v nezúženém pohledu (což vedení sw projektu je) uvažovat, natož to plánovat, organizovat a řídit.

3.2 Postačující "metodická" výbava

Pro vedení projektu vývoje sw je potřeba aplikaci "základní" metody vedení vývoje uvažovat a vzhledem k dekompozičním jednotkám sw systému rekurzivně realizovat v kontextu a pomocí:

- modelu životního cyklu vývoje software (SDLC)[4],
- architektury sw,
- průběžného a postupného plánování.

Zde už jen poznamenejme, že debaty a otázky kladené v textech typu[5] [Nord04] jsou přesně ty, co nás zajímají, např. kolik zdrojů je potřeba věnovat k získání jakých požadavků, aby bylo možné na dané úrovni dekompozice navrhnout kvalitní architekturu, aby bylo možné organizovat vlastní vývoj. Na

[4] Model SDLC může být různý pro různé dekompoziční jednotky systému; prakticky to však má význam na hrubé úrovni granularity dekompozice systému anebo pro dekompoziční jednotky, které mají různý charakter, např. dbs subsystem a web subsystem.

[5] Je vcelku nepodstatné, že debaty se odehrávají v kontextu "buzzwords" typu *eXtreme Programming*, podstatné je, že směřují k podstatě problému organizace vývoje sw systémů a roli sw architektury při tomto.

rekurzivní podstatu organizovaní vývoje většího sw systému, konkrétně na rekurzivní aplikaci modelu SDLC, upozorňuje [Carlin90].

4 Dobrá praxe činnosti vedení sw projektu

4.1 Deklarace praktiky

Je nutno, aby činnost/ proces vedení projektu splňovala následující náležitosti:

(i) byl vytvořen a udržován plán projektu, který nezúženě popisuje všechny činnosti při projektu prováděné (např. návrh, konfigurační řízení, monitorování průběhu projektu) z pohledů technického, organizačního tak i úzce plánovacího ve slova smyslu WBS (work breakdown structure, struktura dekompozice práce), odhadů a časového plánu (harmonogram);

(ii) plán projektu musí obsahovat předpisy specifikací důležitých pracovních produktů, dále sám plán projektu včetně svých výrazných dílčích částí/ plánů (např. plán testování, plán konfiguračního řízení atd.) musí být vytvořen na základě kvalitního předpisu/ vzoru; tyto předpisy pracovních produktů včetně plánů zajišťují, že se na nic podstatného nezapomene, usnadňují práci a vlastně představují určitý nárok na odpovídající činnosti;

(iii) je nutno sledovat průběh projektu a na základě dosavadního průběhu a na základě stupně rozpracovanosti vyvíjeného sw produktu nebo jeho určitých částí provádět nutná doplánování, přeplánování a korektivní akce, což se vše projeví změnami a dodělávám plánu projektu po celou dobu projektu; tento nárok platí obecně pro všechny činnosti a pro všechny části plánu s tím, že se projeví různě pro různé věci (např. pro činnost testování lze na začátku projektu naplánovat přístup nikoli však testovací případy pro integrační testování, obdobně to platí pro V&V, SQA, CM a vlastní úzce chápané plánování ve smyslu WBS, odhadů a časového plánu); konkrétně pro WBS, odhady a časový plán to znamená, že je třeba je aktualizovat podle dosavadního průběhu a zdetailňovat podle stavu rozpracovanosti vyvíjeného sw produktu, resp. jeho částí (např. je zjevné, že v době, kdy není hotov návrh, tak nelze detailně plánovat programování – není prostě vzhledem k čemu); u WBS v případě tzv. product-oriented úkolů (právě třeba návrh, programování; ne CM, V&V, SQA, vedení atd. – to jsou tzv. process-oriented úkoly) je třeba dojít k uchopitelným úkolů v rozsahu několika člověko-týdnů; (pozn. tento bod č. iii schematicky rozvíjí, co znamená nárok na průběžné monitorování a reagování a tzv.“two-tier approach to planning“, což bude vysvětleno později);

(iv) pro projekt je nutno zvolit a popsat vhodný model nebo více modelů SDLC (model postupu vývoje, model životního cyklu), který tvoří koncepční rámec pro vedení projektu jmenovitě pro konkrétní volbu a plánování postupu vývoje; dále vedení projektu, konkrétně postup vývoje musí odpovídat zásadám popsaným v článku Anchoring the Software Process od Boehma a tomu musí také odpovídat volba a aplikace konkrétních modelů SDLC; zmíněný článek vyslovuje zásadní nároky na postup vývoje, které jsou smysluplné pro v podstatě všechny typy projektů, kde nejde jen o údržbu ve smyslu malých oprav anebo změn;

(v) je nutno, aby proces vedení projektu při jeho plánování a provádění, byl prostředkem na prosazení těchto podstatných praktik pro začátek do reálné praxe;

(vi) z průběhu projektu, tj. naměřených kvantitativních dat, zjištění kvalitativních dat a dalších postřehů, je nutno vypracovat historii projektu, která musí být k dispozici dalším projektům.

4.2 Konkrétní naplnění praktiky

4.2.1 Tvorba, používání a údržba plánu projektu

I) Je třeba docílit, aby v rámci činnosti vedení sw projektu byl vytvořen plán projektu, aby byl používán (dodržován) a aby byl udržován. Plánem projektu je zde obecně myšlen plán, který stanovuje všechny potřebné aspekty všech činností, které se mají provádět při vývoji softwaru (pozn. plán projektu dokumentuje softwarový proces projektu). Všemi potřebnými aspekty lze

schematicky rozumět technickou stránku věci, organizační stránku věci a úzce plánovací stránku věci (ve smyslu struktury dekompozice práce (work breakdown structure – WBS), odhadování, časového plánu (harmonogram)).

II) Plán projektu, resp. to, co je v té době principielně možné, je třeba vypracovat v době iniciálního plánování na začátku projektu souběžně se zjišťováním požadavků zákazníka, jejich (předběžnou) analýzou a příp. (předběžným) návrhem.

III) Při projektu je třeba usilovat o dodržování plánu projektu (promyšlené jasně stanovené technické záležitosti např. metoda návrhu, vzor specifikace požadavků, postup pro přezkoumání atd. by kromě velmi závažných důvodů neměly být měněny; odhady, harmonogram atd. pochopitelně žijí).

IV) Plán projektu je od chvíle jeho vzniku po celou dobu projektu třeba vhodně udržovat. Tato údržba má několik principielních příčin a podob. Vybrané principielní příčiny údržby plánu projektu: (i) mnoho věcí lze z principu rozhodovat, stanovovat a plánovat až na základě existence určitých technických artefaktů jako je např. softwarová architektura atd.; (ii) je třeba reagovat na skutečný vývoj událostí, např. ukáže se, jak co dlouho trvá atd.; (iii) lépe se poznávají anebo se mění požadavky zákazníka. Podoby údržby plánu projektu: (i) dopracování technických záležitostí např. konkrétní testovací případy integračního testování nemohou být stanoveny před dekompozicí softwaru atd.; (ii) zpřesnění, dopracování a opravy WBS, odhadů a harmonogramu, např. v době, kdy existuje detailní návrh, lze příslušné odhady dělat daleko přesněji a WBS mít daleko detailnější než v době, kdy neexistuje ani architektura atd.; (iii) další nutné změny. Tedy údržba plánu projektu vlastně v různé míře a v různou chvíli znamená jeho dopracování, zpřesnění, změnu, opravu.

V) Plán projektu je jeden ze základních prostředků činnosti vedení sw projektu a je třeba, aby svým obsahem odpovídal kvalitním standardním vzorům a předpisům pro tento typ pracovního produktu (zde je pochopitelně míněn kompletní plán projektu včetně všech dílčích plánů typu plán konfiguračního řízení, testování, SQA atd.). Dále pak je třeba, aby plán projektu odpovídal předem přijatému vzoru, který vychází a respektuje kvalitní standardní vzory/ předpisy pro tento typ pracovního produktu (a je-li to třeba, vhodně respektuje typ a kontext projektu).

VI) Poznámky:

1) Plán projektu je v různých textech různě nazýván: Project Plan u Metzgera[6], *Management Plan* v NASA Software Documentation Standard, *Software Development/ Management Plan* v NASA příručce Manager's Handbook for Software Development (k tomu nedílně patří ještě plán testování), *Software Development Plan* ve standardu MIL-STD-498 (k tomu nedílně patří plán testování, plán instalace a plán přechodu na nový systém), *Software Project Management Plan* ve standardu ESA PSS-05-0 (k tomu nedílně patří plány pro konfigurační řízení, V&V a SQA), *Software Plan* nebo *Project Software Plan* nebo *Software Management Plan* v NASA příručce Software Management Guidebook. V tomto textu se používá termín plán projektu nebo plán softwarového projektu.

2) Co přesně konkrétně lze nebo co se má z plánu projektu vypracovat a kdy záleží pochopitelně na konkrétní situaci. Tj. na zvoleném celkovém postupu vývoje (zjednodušeně modelu životního cyklu (SDLC)), na tom co už je hotovo, na požadavcích zákazníka hlavně co se způsobu dodání týká, na zkušenostech z dřívějška a vůbec celkovém kontextu projektu. V různých textech lze najít různé rady/ heuristiky. Tak Metzger6 schematicky radí následující sekvenci: definování problému, analýza, přechod k návrhu, definování plánu, soupis akceptačních kriterií; s tím dodáním, že činnost plánování se děje od samého začátku projektu a nikdy nekončí, provádí se různé předběžné studie, analýzy atd., přechod k návrhu

[6] Zde je myšlena „klasická" knížka *Managing a Programming Project: People and Process* (Metzger and Boddie, Prentice-Hall, poslední 3. revidované vydání je z roku 1996,)

se má dít v souběhu s analýzou. V NASA příručce Recommended Approach to Software Development schematicky popisují následující situaci: v době definice požadavků tým vedení má plán pro tuto fázi; v době analýzy požadavků tým vedení připraví plán projektu; v době předběžného návrhu tým vedení přehodnotí harmonogramy, personální nároky, požadované zdroje; v době detailního návrhu tým vedení připravuje plány jednotlivých inkrementů (zde build). (Pozn. tato NASA příručka počítá s velmi konzervativním modelem vývoje typu inkrementální, kde ovšem inkrementy jsou až po detailním designu; pouze v případě obrovských projektů umožňuje inkrementy po architektuře.) V jiné NASA příručce Software Management Guidebook radí, aby se plán projektu začal psát, jakmile se zná definice a rozsah projektu (prostě ty podstatné požadavky uživatele, týkající se nejen samotného softwaru, ale i nároků na způsob a harmonogram dodávky – může ovlivnit model SDLC, na jakost atd.) a to, co je možné by mělo být k dispozici v prvních 30 až 60 dnech projektu. Schematicky popsaný přístup pro zahájení projektu (Project start-up) popisovaný Nesim7 vypadá následovně (použitá notace: [fáze/ proces] -> (hlavní produkt fáze/ procesu)): [Získání požadavků a předběžná (high-level) analýza] -> (Profil projektu) -> [Identifikace subsystémů] -> (Předběžné subsystémy) -> [Analýza subsystémů] -> (První obecný soubor tříd) -> [Předběžná analýza znovupoužitelnosti] -> (Pokyny pro znovupoužívání) -> [Výběr nástrojů] -> (Nástroje) -> [Hrubé plánování projektu] -> (Vazby mezi úkoly) -> [Analýza zdrojů potřebných pro subsystémy] -> (Náklady na subsystémy) -> [Plánování projektu] -> (Plánování) -> [Hodnocení/ výběr zdrojů] -> (Alokace zdrojů) -> (Detailní plán). Přístup nárokovaný ve standardu ESA PSS-05-0: a) aby plán projektu pro „fázi“ požadavky na software vznikl v období „fáze“ definování požadavků uživatele; b) aby dokument s uživatelskými požadavky byl vypracován před vlastním zahájením sw projektu; c) aby plán projektu pro „fázi“ architektonický návrh a hrubý celkový plán projektu vznikl v době „fáze“ specifikování požadavků na software; d) aby ve „fázi“ architektonický návrh byl vypracován detailní plán projektu pro zbytek projektu; e) aby tento byl udržován a dále zpřesňován. Obecně lze říci, že platí následující zásada8: od chvíle, kdy to je možné (znají se podstatné požadavky zákazníka) mít celkový plán projektu, kde pochopitelně mnoho věcí je zatím předběžně. Dále mít vždy pro nadcházející období detailní plán, resp. detailně rozpracovanou příslušnou část celkového plánu, která vzniká dopracováváním a upravováním celkového plánu. Dopracování, zpřesnění, úpravy se pochopitelně týkají WBS, odhadů, harmonogramu, ale částečně se týkají i technických věcí.

3) Tématu, co vše má určitě vedení obsáhnout a tím i plán projektu obsahovat, se také věnují body č. 4.2.2 a 4.2.3 z těchto cílů pro začátek. Tématu údržby a tomu co a kdy má být stanoveno se také věnují body č. 4.2.4 a 4.2.5 z těchto cílů pro začátek. Tématu celkového postupu vývoje, zjednodušeně model životního cyklu, se také věnuje bod č. 4.2.6 z těchto cílů pro začátek.

4.2.2 Předpisy/ vzory pro zásadní pracovní produkty

Je třeba docílit, aby jako nedílná součást stanovení technických náležitostí byly v plánu projektu obsaženy (můžou být pochopitelně referovány) vzory, resp. předpisy pro zásadní pracovní produkty, typu softwarová architektura atd. včetně seznamů kontrolních otázek (checklists). Tyto vzory, resp. předpisy musí být předem stanoveny a musí vycházet ze standardních kvalitních vzorů, resp. předpisů, které jsou k dispozici (a je-li to třeba, vhodně respektovat typ a kontext projektu). Pozn. o vzoru (template) má smysl mluvit, pokud má korespondující pracovní produkt formu klasického dokumentu; o předpisu má smysl mluvit, pokud má korespondující pracovní produkt jinou formu např. je obhospodařován v nástroji typu CASE (předpisy pracovních produktů obsažené ve standardu MIL-

[7] Jedná se o článek „Managing OO Projects Better“ v *IEEE Software*, July/August 1998.

[8] Této zásadě, tomuto konceptu se v knížce *Software Quality: A Framework for Success in Software Development and Support* (Sanders and Curran, ACM Press & Addison-Wesley, 1995) říká „Two-tier approach to planning“ a „Progressively refine the plans“.

STD-498 jsou univerzálním východiskem jak pro předpisy, tak vzory pracovních produktů na úrovni projektu; to samé platí i o ostatních zdrojích vzorů a předpisů jako jsou standardy NASA-STD-2100-91, ESA PSS-05-0 atd.)

4.2.3 Nezúžený záběr plánování

Je třeba docílit, aby plánování mělo ve svém celku dostatečně široký záběr, to jest, aby nebylo chápáno zúženě. Je to míněno v následujícím smyslu: Vedení projektu se týká veškeré práce co se má v daných podmínkách udělat. Ta je dána požadavky zákazníka, omezeními danými zákazníkem, plněním standardů, plněním nároků definujících dobrou odbornou praxi softwarového inženýrství atd. Plánování, součást vedení projektu, se tedy přirozeně týká všech činností, uveďme dvě rozdílné: návrh a validace & verifikace. Plánování se týká všech aspektů: např. u návrhu je třeba stanovit, jak se dělá (principy, zásady, metoda, nástroje atd.), nároky na technické artefakty, které vyprodukuje (pracovní produkt sw architektura atd.) atd. – to je řekněme technický aspekt; u návrhu je třeba stanovit, kdy se dělá návrh čeho, kdo to dělá, jak dlouho atd. – to je řekněme aspekt úzce plánovací (WBS, odhady, harmonogram, zdroje); u návrhu je třeba stanovit, jak se kontroluje – to je řekněme aspekt kvality. Je třeba se vyvarovat zúženého vidění plánování pouze několika „typických" činností a aspektu úzce plánovacího. Nezúžený záběr plánování, tedy určení, stanovení, předepsání i „normální" naplánování (ve smyslu dekompozice práce, odhadů, zdrojů a časového plánu) příslušným způsobem pro všechny činnosti tvořící dobrou praxi sw inženýrství a pro všechny jejich aspekty je první nezbytný krok na cestě k dobré odborné praxi softwarového inženýrství při projektech. Pozn.: nárokování standardního kvalitního vzoru/ předpisu pro plán softwarového projektu (včetně všech příp. dílčích plánů typu CM, testování atd.) v bodě č. 4.2.1 výše a nárokování standardních kvalitních vzorů/ předpisů pro zásadní pracovní produkty (typu sw architektura) v bodě č. 4.2.2 výše představuje jednoduše formulovatelný nárok, který požaduje principielní věci a který se také částečně podílí na nárokování nezúženého záběru plánování. (Pozn. Starost o nezúžený záběr plánování se velmi sníží pokud existuje předepsaný softwarový proces organizace (též standardní softwarový proces organizace), který odpovídá požadovaným nárokům. Předpis softwarového procesu organizace může být v případě potřeby několikaúrovňový/ hierarchický (organizace je velká anebo vývoj probíhá v různých kontextech). V případě, že existuje standardní softwarový proces organizace, je tímto obecně zajištěno, že pro všechny činnosti prováděné při sw projektu jsou anebo budou stanoveny všechny náležitosti (samozřejmě záleží na jeho záběru). Nicméně tento standardní softwarový proces se musí pro daný softwarový projekt v iniciálním plánování přizpůsobit[9]; plán softwarového projektu pak principielně obsahuje to, co je pro daný projekt odlišné a jedinečné samozřejmě včetně WBS, odhadů, sítě aktivit a časového plánu pro daný projekt[10].

4.2.4 „Základní" metoda vedení sw projektu

I) Je třeba docílit, aby ty činnosti, které bezprostředně a prakticky umožňují provedení, resp. vykonání11 všeho, co se má udělat a tím realizaci projektu, byly dostatečně, vhodně a rutinně

[9] Problematikou přizpůsobení standardního softwarového procesu organizace pro konkrétní projekt se např. zabývá technologická zpráva Process Tailoring for Software Projects Plans ze STSC. (Pozn. Zpráva je k dispozici na serveru STSC; její autoři napsali na toto téma také dva delší články do CrossTalku s obdobným názvem.)

[10] Článek A Process or a Plan? od Humphrey(e) stručně a výstižně popisuje role a vztahy mezi definicí procesu, plány a skutečnou prací při projektu. (Pozn. článek je k dispozici na serveru SEI na stránce Featured Articles.)

[11] Umožnit vykonání, resp. provedení zde v souvislosti se základními činnostmi vedení je myšleno následovně: Na jedné straně stojí požadavky zákazníka a požadavky na dobrou praxi softwarového inženýrství, na druhé straně pro každodenní praxi potřebujeme dobře definované úkoly rozsahu několika málo člověko-týdnů s přidělenými zdroji a uspořádanými v harmonogramu. V tomto smyslu o tom ilustrativně píše standard ESA PSS-05-0 v sekci 2.2.5 Planning, scheduling and budgeting the work: „... Odhadování zdrojů a časových rozsahů potřebných pro aktivity je jedna klíčová část plánování jejich vykonání. Základní přístup k odhadování je rozložit projekt do úloh (task), které jsou dost malé pro snadné a přesné ohodnocení jejich náročnosti. ... Každá úloha by měla být vztažena k nějaké vhodné části toho, co se dodá v dané fázi. Na příklad úlohy ve fázi definice požadavků na software mohou být založeny na požadavcích, zatímco ve fázi architektonického návrhu

prováděny. Pracovně a schematicky těmto činnostem říkejme základní činnosti vedení. Základní činnosti vedení podstatnou měrou po celou dobu projektu realizují „základní“ metodu vedení. „Základní“ metodou vedení zde budeme rozumět triviální schéma, jak popsat realizaci a vedení prakticky jakéhokoli projektu. Zjednodušeně řečeno „základní“ metoda vedení projektu spočívá v: (i) ustavení organizace projektu; (ii) zjistit a porozumět „práci, která se má udělat“; (iii) (postupném) rozložení „celé práce, která se má dělat“, na jednotlivé úkoly až jsou definovány dostatečně kompaktní jednotky práce; díky a vzhledem k identifikovaným úkolům a jednotkám práce provést odhady (velikosti - size, úsilí - effort, nákladů - cost), přiřadit omezení, nároky na zdroje, provést analýzu závislostí atd.; výsledek je, že existuje WBS, harmonogram, síť aktivit a dále různé možné odvozené věci typu profil nároků na personál atd.; (iv) přiřazení zdrojů; (v) spolu s prováděním vlastní práce provádět měření, monitorování (sledování), hlášení a zaznamenávání postupu; přijímání nápravných opatření vzhledem k iii a iv (tj. jedná se o doplánování nebo přeplánování); (vi) zaznamenání a uložení údajů o projektu pro další použití. Pozn. body (i) až (vi) nejsou míněny a priori sekvenčně. Tuto „základní“ metodu vedení lze v různých podobách a vyjádřeních nalézt např. v NASA příručce Software Management Guidebook, STSC Report on Project Management, SPICE, CMM, Software Project Management Curriculum Module (od SEI), ISO 12207 atd. Shrnuto, je třeba provádět základní metodu vedení. Dále nás budou zajímat tři základní činnosti vedení: plánování, sledování (monitorování), vyhodnocování. Pozn. plánováním je myšlena činnost plánování pro účely základní metody vedení, tj. to, čemu v tomto textu také říkáme úzce pojaté plánování (ve smyslu WBS, odhady, harmonogram).

II) II) Je třeba provádět plánování pro zajištění „základní“ metody vedení. Toto (minimálně) znamená vytvořit a udržovat strukturu dekompozice práce (WBS – Work Breakdown Structure) spolu se sítí aktivit (activity network), vytvořit a udržovat odhady, vytvořit a udržovat časový plán (harmonogram). WBS se myslí pro všechny činnosti (tj. např. i vedení a nejen programování). Odhady se myslí odhady velikosti (size), úsilí (effort) a nákladů (cost). Udržovat pro WBS znamená postupně v průběhu projektu zdetailňovat. Udržovat pro odhady znamená postupně v průběhu projektu zpřesňovat. Udržovat pro harmonogram znamená aktualizovat na základě dosavadního průběhu a na základě přesnějších odhadů, příp. detailnější WBS. Takto chápané plánování („základní“ plánování, úzce chápané plánování) realizuje iniciální plánování na počátku projektu a dále potřebné doplánování anebo přeplánování po celou dobu projektu (pozn. základní plánování tedy pomáhá vytvořit a poté

mohou být založeny na komponentách. Tradičně, odhady pro detailní návrh a produkci jsou založeny na řádcích kódu. ... Struktura dekompozice práce (WBS) je jeden z nejzákladnějších nástrojů pro plánování a řízení aktivit projektu. WBS popisuje hierarchii úloh seskupených do pracovních celků (work package), které mají být provedeny při projektu. WBS koresponduje se strukturou práce, která má být provedena a odráží způsob, kterým projektové náklady budou sumarizovány a hlášeny. ... Trvání produktově orientovaných pracovních celků by mělo být dostatečně krátké, aby se zajistila viditelnost procesu produkce (např. měsíc ve fázi detailní návrh a produkce). Procedurově orientované pracovní celky, např. vedení projektu, mohou mít rozsah přes délku celého projektu. ...“. Umožnit vykonání všeho co se při projektu vykonat má, ve skutečnosti takto přímočaře jednoduché není. Za prvé, možnost konstruovat WBS do značné míry, tedy přesně pro tzv. product-oriented work packages logicky závisí na existenci pracovních produktů specifikace požadavků zákazníka, sw architektura a detailní návrh. Dále musí existovat názor na celkový postup vývoje, jinými slovy a zjednodušeně musí být vybrán vhodný model životního cyklu – SDLC, který odpovídá na otázky: Jaké požadavky je třeba znát, aby šla dělat architektura? Může se iterovat, když je hotová architektura? atd. Všeobecně známá jména pro modely životních cyklů jsou model vodopád, model inkrementální, model evoluční a model spirálový; nicméně toto jsou jen „nálepky“ pro určité koncepty. Existuje celá třída modelů inkrementálních, evolučních atd. Dále při rozsáhlých projektech anebo členitých projektech je vhodné mít pro různé úrovně dekompozice (systém, softwarový produkt, komponenta sw produktu) různé modely a pro různé dekompoziční jednotky na stejné úrovni mít různé modely; s tímto počítá a k tomuto nabádá např. standard MIL-STD-498. Pro toto všechno je použito spojení „základní činnosti vedení“ a „základní metoda vedení“, protože je to opravdu jen základ. Z jistého pohledu v podstatě jen mechanický realizující koncepci zvoleného modelu SDLC, koncepci postupného a průběžného plánování.

udržovat plán projektu; principielní důvody pro nutnost postupného a průběžného plánování a tím i údržby plánu projektu byly uvedeny výše v bodě č. 4.2.1)

III) Je třeba průběžně a po celou dobu trvání projektu provádět sledování (monitorování) průběhu projektu. Pro účely vedení projektu je třeba provádět vhodná měření (typicky měření nějak souvisící s velikostí pracovních produktů (size), s vynaloženým úsilím (effort), s běžným kalendářem/ harmonogramem, s problémy/ defekty). Dále je třeba sledovat výsledky, které jsou k dispozici díky jiným činnostem: odborným přezkoumáním, testování, zajištění jakosti, řešení problémů a dalším.

IV) Je třeba průběžně a po celou dobu trvání projektu provádět vyhodnocování průběhu projektu. K vyhodnocování průběhu je třeba použít údaje a informace získané sledováním průběhu projektu. K vyhodnocování je třeba dále používat specielně připravené „testy“ stavu projektu (jde o zvláštní druh checklistu). K vyhodnocování je třeba používat předem připravené a promyšlené indikátory, cílové hodnoty, varující hodnoty. Pro vyhodnocování je třeba používat tzv. koncept Earned Value (spolu s Binary Quality Gates at Inch Pebble Level). Pozn. koncept Earned Value se ve skutečnosti týká všech základních činností vedení, tj. plánování, sledování a vyhodnocování a velmi dobře je integruje. Je třeba, aby výsledkem vyhodnocení, je-li to třeba, byly vhodné nápravné akce realizované činností „základní“ plánování.

4.2.5 Postupné a průběžné plánování

Je třeba docílit, aby plánování bylo prováděno v průběhu celého projektu – tedy průběžně a aby bylo prováděno v souladu se stavem v jakém se projekt nachází – tedy postupně. Plánování je zde myšleno nezúženě ve smyslu bodu č. 4.2.3 výše (pochopitelně včetně „základního“, úzce chápaného plánování pro účely „základní“ metody vedení). Postupné a průběžné plánování je v konečném důsledku realizováno údržbou plánu softwarového projektu (co to schematicky znamená bylo řečeno nebo ilustrováno v bodech č. 4.2.1 a 4.2.4 výše). Principielní důvody, proč je třeba provádět postupné plánování, byly uvedeny v bodě č. 4.2.1 výše (schematicky: (i) pracovní produkty, na kterých záleží plánování vznikají postupně; (ii) nutnost reagovat na skutečný průběh projektu; (iii) postupné chápaní anebo změna požadavků; (iv) další důvody). Sanders tento rys vedení softwarových projektů nazývá postupné zpřesňování plánů (progressively refine the plans) a v rámci sekce principy vedení projektu k tomu píše[12] „Plánování je pokus předpovědět budoucnost a vždy je založeno na nepřesných informacích. Proto je obtížné být úplně přesný a čím je budoucno plánování vzdálenější, tím obtížnější se plánování stává. Obtížnost spočívá částečně ve faktu, že není možné předpovědět každou eventualitu a částečně ve faktu, že vnímání plánovaného úkolu je měněno a zpřesňováno tím, jak je úkol vykonáván. Proto je zde třeba dvousložkového přístupu k plánování [two-tier approach to planning] . V jakémkoli daném bodě by měly být plány na dvou úrovních: přehledový plán pokrývající dlouhodobé záměry, aktivity a priority na vysoké úrovni [abstrakce]; a detailní plán, který je zpřesněním přehledového plánu pokrývající bezprostřední budoucnost na vyšší úrovni detailu.” (Pozn. k části výňatku “... vnímání plánovaného úkolu je měněno a zpřesňováno tím jak je úkol vykonáván”, tento druhý fakt, kromě chápání anebo změny požadavků, prostě znamená, že plánování a vedení projektu je velmi závislé na technických artefaktech reprezentujících vyvíjený softwarový produkt, které jsou v danou chvíli k dispozici. Pro plánování je velký rozdíl, zda je k dispozici pracovní produkt typu „koncept fungování“ nebo softwarová architektura. Tato skutečnost je velmi akcentována ve standardu ESA PSS-05-0, což je velmi cenné, protože to není úplně obvyklé. Standard ESA PSS-05-0 například obsahuje pro všechny plány dohromady tvořící plán projektu (Software Project Management Plan (SPMP), Software Quality Assurance Plan (SQAP), Software Validation and Verification Plan (SVVP), Software Configuration Management Plan (SCMP)) instrukce, jak mají vypadat pro určitou rozpracovanost vyvíjeného sw produktu (konkrétně se jedná o „fáze“ požadavky uživatele, požadavky na software, architektura, detailní návrh; na slovo „fáze“ pozor, různé části vyvíjeného produktu mohou být v různých stavech rozpracovanosti).

[12] Sanders, sekce 5.3, strany 73-75.

4.2.6 Vhodný a správný celkový postup vývoje softwaru, který zahrnuje vhodný(é) model(y) životního cyklu

I) Je třeba docílit, aby v době iniciálního plánování byl pro projekt vybrán vhodný model životního cyklu. Tento model musí být dostatečně popsán (nebo popis, existuje-li, referován). Je třeba docílit, aby se vybraný a popsaný model také správně použil. To znamená, aby byl po celou dobu trvání projektu pro plánování a „základní" metodu vedení projektu koncepčním návodem, jak postupovat (a to je asi nejpravdivější funkční definice modelu životního cyklu: být v daném kontextu a daných omezujících podmínkách pro plánování a „základní" metodu vedení sw projektu koncepčním návodem jak postupovat). Modelem životního cyklu rozumíme koncepční určení postupu vývoje sw produktu vzhledem k primárním činnostem softwarového inženýrství (tj. získání požadavků, analýza, návrh architektury, detailní návrh, implementace, testování, uvedení do provozu) a příslušným pracovním produktům. Často uváděné modely jsou: vodopád, inkrementální, evoluční, spirálový atd. Bohužel jejich vymezení, aspoň prostředních dvou, jsou často tak vágní, že přináší více otázek než jich řeší. Ve skutečnosti tyto názvy typu „inkrementální", „iterativní", „evoluční" atd. jsou jisté koncepty, které vlastně nazývají třídy modelů životních cyklů. Proto je vždy třeba v plánu projektu jasně popsat co se stanoveným modelem životního cyklu vlastně přesně myslí. Konkrétní dobře popsaný model životního cyklu jasně stanovuje věci typu – jaké požadavky je třeba znát, aby se mohlo přikročit k architektuře? může se po návrhu architektury postupovat postupně (inkrementálně) a co to vlastně znamená? může se po detailním návrhu postupovat postupně a co to vlastně znamená? atd. (Pozn. od určité míry detailu model životního cyklu úzce souvisí s použitou technologií. Např. objektově orientovaný přístup, díky své přirozené modularitě, přinesl velký potenciál flexibilních možností vývoje, což se odráží v používaných a hlavně proklamovaných modelech životního cyklu. Vždy se musí přesně říci, co se myslí pod pojmem inkrement, iterace, evoluce atd.) Vezme-li se jeden běžně vnímaný, model životního cyklu, tak sám o sobě nemusí zdaleka vždy stačit, aby splnil roli koncepčního návodu pro postup plánování a „základní" metodu vedení. Principielní důvody pro to jsou následující: (i) Je-li vyvíjený sw produkt rozsáhlý anebo je-li součástí vývoje nějakého nadřazeného systému, potom je obecně možné a často nezbytné mít různé modely životního cyklu pro různé úrovně dekompozice (např. pro úroveň nadřazeného systému; pro úroveň částí systému, které jsou realizovány sw produktem). Jedná se o prostou rekurzivní aplikaci pravidla rozděl a panuj. Tento postup, kdy různé úrovně dekompozice (systém, CSCI ~ sw produkt, část sw produktu), je-li to třeba, mají mít různé modely životního cyklu, velmi zdůrazňuje a podporuje standard MIL-STD-498.13 Principielně o to samé, ale trochu jinak podávané, jde v diskusích14 týkajících se vhodného postupu vývoje při objektově orientovaném vývoji, kdy na jedné straně je potřeba mít model vývoje pro celý projekt a na druhé straně model, který zvládá a konsoliduje vývoj jednotlivých dekompozičních jednotek střední granularity (např. class cluster) anebo jednotlivých iterací (tyto iterace bývají chápany většinou tak, že po architektuře zajišťují vývoj klasicky chápaných inkrementů – ať už postupně nebo souběžně – nebo před architekturou „evolučně" dospívají k robustní funkční architektuře) anebo obecně řečeno „každodenního života vývojáře"; (ii) Je-li systém členitý ve smyslu nestejnorodý, tj. mají-li dekompoziční jednotky na stejné úrovni dekompozice výrazně odlišný charakter, pak může být nezbytné a jenom přirozené zvolit pro každou jiný model

[13] O konkrétním použití tohoto rysu se lze např. dočíst v článku „MIL-STD-498: What's New and Some Real Lessons Learned" v *CrossTalk*, March 1996.

[14] Diskuse tohoto tématu lze nalézt např. v článcích „The Object Oriented Systems Life Cycle" v *Communication of the ACM*, September 1990; „Managing OO Projects Better" v *IEEE Software*, July/August 1998; „Process Control for Error-Free Software: A Success Story" v *IEEE Software*, May/June 1999; *Rational Unified Process* (Rational white paper); „A Rational Development Process" v *CrossTalk*, July 1996 (taky jako Rational white paper); článek „Using Win Win Spiral Model: A Case Study" od Boehma v *Computer*, July 1998 atd.

vývoje15 (např. jedna část představuje uživatelské rozhraní a druhá programy obsluhující databázový stroj); (iii) Pro různé časové úseky životního cyklu může být vhodné použít různé modely. Typicky, jeden časový úsek představuje doba před tím, než je k dispozici kvalitní architektura a druhý časový úsek je od doby, kdy k dispozici je (přičemž např. pro získání architektury lze použít prototypování a po architektuře použít konzervativní inkrementální model vývoje). Tuto možnost velmi zdůrazňuje Boehm v textu Anchoring the Software Process (viz zdroje), dále lze např. takto velmi přirozeně použít proces fy Rational. Poznámky k principielním důvodům i) – iii): a) situace popisované v bodech ii) a iii) se obecně může vyskytovat na několika úrovních dekompozice, jak se o tom mluví v bodě i); b) když se na začátku tohoto cíle č. 4.2.8 mluvilo o (jednom) běžném modelu životního cyklu, tak šlo vlastně o abstrakci, která buď zastupuje opravdu jeden model životního cyklu, který pro potřeby daného projektu stačí nebo je třeba vzít v úvahu potřebu více vzájemně skloubených modelů životního cyklu, jak se o tom mluví v bodech i) – iii); a takto to bude chápáno i nadále; c) model životního cyklu je koncepční návod jak postupovat s plánováním a „základní“ metodou vedení projektu a toto je třeba při plánování skutečně respektovat; nevhodné časové plánování a nevhodný kalendář pro dodání pracovních produktů může zcela zmařit16 veškerou flexibilitu, souběžnost atd., které při rozhodování o modelu životního cyklu hrála roli; toto přirozeně platí pro všechny úrovně dekompozice a všechny části, které mají „svůj“ model životního cyklu.

II) Je třeba docílit, aby celkový postup vývoje splňoval nároky17 popsané a vysvětlené v textu Anchoring the Software Process od Boehma. Tento text nárokuje a vysvětluje co by měl splňovat v podstatě každý proces vývoje (softwarový proces projektu) – co se jeho postupu týká. Nenárokuje konkrétně, žádný konkrétní model životního cyklu, naopak povzbuzuje pro různé modely životního cyklu v různých situacích. Boehm(ův) text tedy představuje nároky, kterým musí vyhovět použitý model životního cyklu (ať už se jedná skutečně jen o jeden či jejich soustavu, viz výše) a které musí být realizovány vhodným plánováním a „základní“ metodu vedení sw projektu. Nebo to lze taky chápat tak, že naopak zvolené modely životního cyklu musí kromě jiného odpovídat nárokům Boehmova textu a zajistit jejich realizaci. Text Anchoring the Software Process se zabývá vskutku tématy, která jsou klíčová pro koncepční otázky Co je invariantní v tom, jak vlastně projekt vést? Co je invariantní v tom, jak při vývoji postupovat?: rolí a vztahem požadavků vzhledem k vedení projektu, vztahem požadavků vzhledem k sw architektuře, rolí a vztahem sw architektury k vedení sw projektu atd. (pozn. Boehm mimo jiné ukazuje velký význam pracovního produktu sw architektura pro činnost vedení sw projektu18 a vyžaduje vzhledem k tomuto příslušným způsobem postupovat). Boehmovy nároky v jisté fázi formulace přijal i Rational Unified Process nakonec popsaný v [Jacobson].

4.2.7 Ošetření rizik

Je třeba docílit, aby aspoň nejzákladnějším způsobem byla ošetřována rizika (tzv. Risk management).

---

[15] Případ tohoto typu je např. popisován v článku „MIL-STD-498: What's New and Some Real Lessons Learned“ v *CrossTalk*, March 1996.

[16] Před tímto několikrát a velmi důrazně varuje standard MIL-STD-498, konkrétně v příloze H a v samostatné příručce *MIL-STD-498 Overview and Tailoring Guidebook*.

[17] Zjednodušeně řečeno, při vývoji každého systému existují tři zásadní milníky, kterým je nutno rozumět a vědomě s nimi pracovat. Tyto milníky jsou: LCO (Life cycle objectives), LCA (Life cycle architecture), IOC (Initial operational capability).

[18] Tyto trendy, které velmi vyzdvihují roli architektury pro vedení projektu, lze vidět např. v textech „Improving Software Economics in the Aerospace and Defense Industry“ *v Guidelines for Successful Acquisition and Management of Software-Intensive Systems*, Addendum A (také jako Rational white paper); „A New Process for Acquiring Software Architecture“ ve stejné příručce jako minulý text, Addendum G; *On the Definition of Software System Architecture*, technická zpráva Center for Software Engineering; *Architecture-Based Development*, SEI-99-TR-007.Po létech pak shrnuto např. v [Paulish].

4.2.8 Odborné znalosti vedoucího projektu

Je třeba docílit, aby vedoucí projektu, resp. osoby, které projekt výkonně plánují a řídí měli dostatečné odborné znalosti. Praxe je nezastupitelná, ale bez příslušných odborných znalostí nemohou být zkušenosti, které praxe přináší, vždy správně pochopeny a tak v budoucnu co nejlépe zužitkovány. Tato potřeba dostatečných odborných znalostí je umocněna tím, že činnost vedení sw projektu má, ze všech činností, které tvoří dobrou praxi softwarového inženýrství, sama o sobě obecně největší dopad. Vedoucí projektu (je míněn výkonný vedoucí ať už se role nazývá jakkoli) a další osoby, které se fakticky podílí na plánování a řízení projektu musí mít tedy dobré odborné znalosti činnosti vedení sw projektu (prostě je třeba se poučit z odborné literatury, kde jsou koncentrovány empirické zkušenosti za desetiletí spolu se základními principy; a nedopouštět se elementárních prohřešků, či vynalézat kolo). Dále musí znát aspoň základy celé discipliny softwarové inženýrství a základní principy jednotlivých činností, které tvoří softwarové inženýrství.

5 Poděkování

Autor chce tímto poděkovat svým kolegům, se kterými mnoho let vyvíjí softwarové systémy. Díky nim si celá léta mohl výše uvedené zásady pro vedení sw projektu zkoušet a ověřovat v každodenní praxi. Se kterými mohl uprostřed vývoje vést diskuse např. na téma vztahu architektury, požadavků a XP. Diskuse z principu věci neukončitelné jasným závěrem, ale tolik potřebné pro vedení vývoje sw.

6 Reference

[Brown98] Brown, N. *The Program Manager's Guide to Software Acquisition Best Practices.* Software Acquisition Best Practices Initiative, Departmet of Defense, 1998.

[Metzger96] Metzger, P. et al. *Managing A Programming Project: Processes and People*. 3rd ed. Prentice Hall, 1996.

[NASA96] *Software Management Guidebook*. Software Program, NASA Guidebook, NASA-GB-001-96, NASA, Washington, DC, November 1996.

[SWEBOK] *Guide to Software Engineering Body of knowledge*.IEEE Computer Society, 2004.

[SEL95] *Software Process Improvement Guidebook*, Revision 1, Software Engineering Laboratory Series, SEL-95-102, NASA, 1995.

[Boehm95a] Boehm B., et al. *On the Definition of Software System Architecture*. TR USC/CSE-95-TR-500, CSE, University of Southern California, Los Angeles, CA, 1995.

[Boehm95b] Boehm, B. *Anchoring the Software Process*. TR USC/CSE-95-TR-507, CSE, University of Southern California, Los Angeles, CA, 1995.

[Nord04] Nord, R. et al. *Integrating Software-Architecture-Centric Methods into Extreme Programming (XP)*. CMU/SEI-2004-TN-036, Software Technology Institute, CMU, September 2004

[Carlin90] Carlin, K. et al. "A proposal for a recursive object-oriented life-cycle," *In Proceedings of the conference on TRI-ADA '90*. ACM Press, Baltimore, Maryland, United States, 1990.

[BSI95] BSI DISC. *The TickIT Guide: A Guide to Software Quality System Construction and Certification to ISO 9001:1994*. Issue 3.0, British Standards Institution, DISC TickIT Office, 1995.

[ISO9000-3] ISO 9000-3:1991 *Quality management and quality assurance standards - Part 3: Guidelines for the application of ISO 9001 to the development, supply and maintenance of software*. 1991.

[SPICE95] ISO/IEC, JTC1, SC7. *Software Process Assessment. Part 1 – Part 9*, Working Draft V1.00, SPICE Project, 1995.

[Paulk93] Paulk, M., et al. *Key Practices of the Capability Maturity Model for Software*, Version 1.1. CMU/SEI-93-TR-25, SEI, Carnegie Mellon University, Pittsburgh, 1993.

[MIL-STD-498] *Software Development and Documentation. Military Standard*, MIL-STD-498, Department of Defense, USA, December 1994.

[ISO12207] ISO/IEC 12207:1995(E). *Information technology - Software life cycle processes*. Joint Technical Committee ISO/IEC JTC 1, Subcommittee SC 7, Software Engineering, 1st edition, 1995.

[ESA91] *ESA Software Engineering Standards, Issue 2*. ESA PSS-05-0 Issue 2, ESA Board for Software.

[DOD-STD-2167A] *Defense System Software Development*. U.S. Department of Defense, February 29, 1988.

[Jacobson] Jacobson, I. et al. *The Unified Software Development Process*. Addison-Wesley. 2000.

[Beck] Beck, K. *Extreme Programming Explained*. Addison-Wesley. 1999.

[Paulish] Paulish, D.J. Architecture-Centric Software Project Management: A practical Guide. Addison-Wesley, 2002.