Profinit Software Engineering Best Practices

(Praktiky, konkrétní nároky a odborné zdroje pro "minimální" dobrou praxi softwarového inženýrství)

Tomáš Smolík
Profinit, s.r.o.
tomas.smolik@profinit.eu
http://www.profinit.eu

Předmluva

Cíl tohoto textu je dostupným způsobem zprostředkovat lidem z každodenní praxe vývoje softwaru vlastní disciplinu softwarového inženýrství a minimální dobrou praxi softwarového inženýrství. Usiloval jsem, abych to učinil pokud možno stručně nikoli však na úkor nepřípustných zjednodušení.

Text obsahuje:
- Kapitolu č. 1, kde je deklarováno devět podstatných praktik pro začátek, když chceme usilovat o dobrou praxi softwarového inženýrství. U každé praktiky je řečeno znění, důvod a upozornění.
- Kapitolu č. 2, kde jsou popsány konkrétní nároky pro začátek pro danou praktiku. U každé praktiky je řečeno, co chceme konkrétně docílit, vlastní konkrétní nároky a upozornění pro interpretaci. Vlastní konkrétní nároky jsou stanoveny pomocí referování veřejně dostupných (zadarmo), odborně kvalitních a jak jen možno stručných textů, které dávají k dispozici university, firmy, americká armáda, NASA atd.
- Kapitolu č. 3, kde jsou strukturovaným způsobem zpřístupněny vybrané odborné materiály. Zpřístupněné odborné materiály pokrývají téměř celé softwarové inženýrství. Odborné materiály jsou stručně komentované. Jsou v naprosté většině zadarmo dostupné přes Internet a je uvedeno, kde je lze získat. Snažil jsem se zpřístupnit užitečný průřez skrze texty stručné i rozsáhlé, široce zaměření i úzce vymezené, přehledové i detailní atd., z velké většiny přímo zaměřené na dobrou praxi softwarového inženýrství. Texty, jak už bylo řečeno, pochází z univerzit, firem, armády, různých výzkumných institucí, NASA atd. Tento seznam odborných zdrojů je daleko rozsáhlejší, než je potřeba pro podporu konkrétních nároků z kapitoly číslo 2.
- Přílohu č. 1, kde je shrnuta počáteční předepsaná praxe softwarového inženýrství.
- Přílohu č. 2, kde je uveden seznam vybraných míst (URL adres) se zdroji pro softwarové inženýrství.
- Další přílohy obsahující různé doplňují informace, které mohou být užitečné.

Text lze využít mnoha způsoby:
- Jako zdroj referencí odborných textů k danému tématu, k řešení daného problému atd. v podstatě z celé oblasti softwarového inženýrství. Je-li málo času vezme se kvalitní článek o 4 stranách, je-li nutno věc systematicky nastudovat, vezme se příručka o desítkách/ stovkách stran.
- Jako výchozí bod pro systematické vzdělávání v softwarovém inženýrství.
- Jako zdroj vyžadované praxe, konkrétních nároků a dostupných informací pro všechny zúčastněné a celou dobu trvání projektu vývoje softwaru.
- Jako výchozí návrh definice softwarového procesu organizace, který zajistí minimální dobrou praxi softwarového inženýrství (pozn. toto je míněno vzhledem k vlastní disciplině a k zásadním principům a toto byl prapůvodní záměr tohoto textu).
- Jako zdroj pomoci pro konsolidaci nějaké oblasti, např. požadavky zákazníka, softwarová architektura atd. (míněny jsou jak příslušné činnosti, tak odpovídající pracovní produkty).

Poznámky:
- I když se mluví o minimální dobré praxi softwarového inženýrství, tak to neznamená, že to, co nárokuje tento text v kapitolách č. 1 a 2, je jednoduché nebo že je toho málo. Není to jednoduché a není toho málo. Proto nároky tohoto textu může být potřeba a samozřejmě lze do praxe zavádět postupně, podle konkrétní situace. Text je opravdu velmi stručný. Jedině téma vedení projektu bylo pojednáno trochu podrobněji kvůli své výlučnosti. Text je ve své koncepci i formulaci zcela původní. Jinak stojí na vynikajících odborných zdrojích, které nemá smysl přepisovat anebo opakovat.
- Zkratky a neobvyklé termíny jsou stručně vysvětleny při prvním výskytu. Je-li to třeba, jsou podrobněji komentovány později na místě, kde se to hodí.
- Kriterium pro uváděné zdroje je, aby byly odborně kvalitní, přímo orientované nebo jinak přínosné pro dobrou praxi softwarového inženýrství a veřejně (zadarmo) komukoli přístupné. Z tohoto důvodu ve zdrojích nejsou uváděny standardní monografie na dané téma vydávané nakladatelstvími typu Prentice-Hall, Addison-Wesley atd. a produkce ACM a IEEE CS (to platí až na výjimky, kdy text buď je veřejně přístupný nebo byl stažen z ACM Digital Library a je k dispozici v adresáři Odborné_zdroje).
- Pokud se během doby změní místa uložení některých uváděných odborných zdrojů (např. standard ESA PSS-05-0), neměl by být problém jejich nové či jiné místo uložení zjistit pomocí vyhledávací služby typu AltaVista.
- Vztah k CMM, ISO 9000-3 atd.: Tento text se primárně zaměřuje na základní principy softwarového inženýrství a z toho plynoucí dobrou praxi. Může bez problémů sloužit jako pomůcka k ustanovení procesu o jaký CMM, ISO 9000-3 atd. v principu jde. Nicméně CMM i ISO mají širší záběr a mnoho dalších nároků.

Obsah

1 DEKLARACE PODSTATNÝCH PRAKTIK PRO ZAČÁTEK 1

1.1 POŽADAVKY 1
1.2 SOFTWAROVÁ ARCHITEKTURA 1
1.3 NÁVRH, DETAILNÍ NÁVRH A PROGRAMOVÁNÍ 2
1.4 TESTOVÁNÍ 3
1.5 KONFIGURAČNÍ ŘÍZENÍ 3
1.6 ODBORNÁ PŘEZKOUMÁNÍ 4
1.7 ZAJIŠTĚNÍ NAPLÁNOVANÝCH A PŘEDEPSANÝCH VĚCÍ 5
1.8 VEDENÍ SOFTWAROVÉHO PROJEKTU 6
1.9 MECHANISMUS ŘEŠENÍ PROBLÉMŮ 7

2 KONKRÉTNÍ NÁROKY NA PRAXI SOFTWAROVÉHO INŽENÝRSTVÍ 8

2.1 POŽADAVKY 8
2.2 SOFTWAROVÁ ARCHITEKTURA 9
2.3 NÁVRH, DETAILNÍ NÁVRH A PROGRAMOVÁNÍ 11
2.4 TESTOVÁNÍ 11
2.5 KONFIGURAČNÍ ŘÍZENÍ 13
2.6 ODBORNÁ PŘEZKOUMÁNÍ 13
2.7 ZAJIŠTĚNÍ NAPLÁNOVANÝCH A PŘEDEPSANÝCH VĚCÍ 15
2.8 VEDENÍ SOFTWAROVÉHO PROJEKTU 16
2.9 MECHANISMUS ŘEŠENÍ PROBLÉMŮ 28

3 ODBORNÉ ZDROJE PRO JEDNOTLIVÉ ČINNOSTI/ OBLASTI SOFTWAROVÉHO INŽENÝRSTVÍ 30

3.1 POŽADAVKY NA SOFTWARE 30
3.2 NÁVRH SOFTWARU 32
3.3 IMPLEMENTACE SOFTWARU 33
3.4 TESTOVÁNÍ SOFTWARU 34
3.5 ÚDRŽBA (A ROZVOJ) SOFTWARU 35
3.6 KONFIGURAČNÍ ŘÍZENÍ SOFTWARU 35
3.7 VALIDACE A VERIFIKACE SOFTWARU 36
3.8 ZAJIŠTĚNÍ JAKOSTI SOFTWARU 37
3.9 VEDENÍ SOFTWAROVÉHO PROJEKTU 38
3.10 ŠKOLENÍ A VZDĚLÁVÁNÍ 43
3.11 MATERIÁLY K SOFTWAROVÉMU INŽENÝRSTVÍ JAKO CELKU 44
3.12 VZORY DOKUMENTŮ, PŘEDPISY PRACOVNÍCH PRODUKTŮ, CHECKLISTS ATD. VOLNĚ K DISPOZICI 46
3.12.1 Standard MIL-STD-498 46
3.12.2 NASA-STD-2100-91 (NASA Software Documentation Standard) 47
3.12.3 NASA Manager's Handbook for Software Development 48
3.12.4 Standard ESA PSS-05-0 49
3.12.5 Checklists 49
3.12.6 Další 50

4 PŘÍLOHA Č. 1: POČÁTEČNÍ PŘEDPIS SOFTWAROVÉHO PROCESU V TABULCE 51

5 PŘÍLOHA Č. 2: URL ADRESY ZDROJŮ ODBORNÝCH TEXTŮ 58

6 PŘÍLOHA Č. 3: CO TVOŘÍ DISCIPLINU SOFTWAROVÉ INŽENÝRSTVÍ – PŘEHLED 62

6.1 A SOFTWARE ENGINEERING BODY OF KNOWLEDGE, VERSION 1.0; SEI-99-TR-004 62
6.2 GUIDE TO THE SOFTWARE ENGINEERING BODY OF KNOWLEDGE, ACM & IEEE CS 63
6.3 GUIDELINES FOR SOFTWARE ENGINEERING EDUCATION, VERSION 1.0; WGSEET 63
6.4 SPICE 63

7 PŘÍLOHA Č. 4: JINÉ SEZNAMY PRINCIPIELNÍCH PRAKTIK 64

7.1 PODLE IEEE SOFTWARE 64
7.2 PODLE SOFTWARE PROGRAM MANAGER'S NETWORK 64
7.3 PODLE CMM 64

8 PŘÍLOHA Č. 5: KOŘENY A OKOLNOSTI VZNIKU TOHOTO TEXTU 65

8.1 USTANOVENÍ A ROZVOJ STANDARDNÍHO SOFTWAROVÉHO PROCESU ORGANIZACE + „ASSETS“ 65
8.1.1 Úvod 65
8.1.1.1 Účel textu 65
8.1.1.2 Problém a jeho vymezení 65
8.1.1.3 Přehled navrženého řešení 66
8.1.1.3.1 Počáteční předpis softwarového procesu a Software Process Assets 66
8.2 MOŽNOST VYUŽITÍ NEDOKONČENÉ PRACOVNÍ VERZE RÁMCE SOFTWAROVÉHO INŽENÝRSTVÍ (RSI) 68

9 PŘÍLOHA Č. 6: ÚVODNÍ KAPITOLA Z TEXTU: POZNÁMKY K VEDENÍ PROJEKTU I 69

9.1 „ZÁKLADNÍ“ METODA VEDENÍ PROJEKTU 69
9.2 NASTÍNĚNÍ NUTNÝCH ROZŠÍŘENÍ K ZÁKLADNÍ METODĚ 69
9.2.1 Nezúžený pohled na záběr a rozsah vedení softwarového projektu 69
9.2.2 Zásadní důležitost koncepční organizace primárních aktivit SE – modelů SDLC 69
9.2.3 Danost: Postupné zpřesňování plánů a soustavná úprava a údržba plánů 70
9.2.4 Zásada: umožnit různé způsoby vedení vzhledem k různým částem produktu a projektu 70
9.3 ROLE TOHOTO TEXTU 70
9.4 POZNÁMKY 71
9.4.1 Co se myslí nezjednodušený, nezúžený atd. 71
9.4.2 Rozsah a záběr textu versus konkrétní situace daných projektů 71

1 Deklarace podstatných praktik pro začátek

Následující praktiky jsou součástí jádra dobré odborné praxe softwarového inženýrství. Stejně lze říci, že jsou pro proces vývoje softwaru velmi důležité až naprosto zásadní. Proto pro dobrou praxi softwarového inženýrství anebo pro počáteční předpis softwarového procesu organizace představují jakési minimum a mají se stát co nejrychleji součástí běžné praxe v co nejširším měřítku. Seznam podstatných praktik je v tabulce č. 1-1. V příloze č. 4 jsou pro představu uvedeny další seznamy podstatných praktik.

K samotné formě popisu podstatných praktik. Název je orientační pomůcka pro snadné zapamatování a odpovídá oblasti anebo činnosti softwarového inženýrství, které se podstatná praktika týká. Je to lepší než jen číslo. Nebylo snahou, a taky tomu tak zcela není, aby se vyhlášení podstatných praktik pro začátek strukturálně krylo s členěním kapitoly č. 3. Obě strukturování mají totiž jiné cíle. Kapitola č. 3 Odborné zdroje pro jednotlivé činnosti/ oblasti softwarového inženýrství vychází z členění *Guide to the Software Engineering Body of Knowledge* a je konstruována tak, aby co nejlépe umožnila zařadit znalosti o softwarovém inženýrství. Kdežto podstatné praktiky pro začátek vychází z výše uvedených pohnutek.

Činnosti/ oblasti, kterých se týkají dále uvedené podstatné praktiky pro začátek
1. Požadavky
2. Softwarová architektura
3. Detailní návrh a programování
4. Testování
5. Konfigurační řízení
6. Odborná přezkoumání
7. Zajištění naplánovaných a předepsaných věcí
8. Vedení softwarového projektu
9. Mechanismus řešení problémů

Tabulka 1-1: Seznam podstatných praktik

1.1 Požadavky

Praktika číslo 1
Název: Požadavky

Znění:
Je nutno co nejlépe zjišťovat, formulovat, zaznamenávat a udržovat požadavky uživatele anebo zákazníka. Tyto se týkají (i) požadovaného software, příp. systému a software a (ii) způsobu, průběhu, časování a dalších náležitostí softwarového projektu a dodávky vyvíjeného softwarového produktu.

Důvod:
Špatně zjištěné, nevhodně formulované, nevhodně zaznamenané a neudržované požadavky zákazníka vedou k velkým časovým a tím i finančním ztrátám v průběhu celého života vyvíjeného sw produktu. Prvním varovným znamením je, že paralelně se specifikací požadavků nevzniká a poté s životem požadavků není udržován plán akceptačního testování, na základě kterého zákazník převezme vyvíjený sw produkt, příp. jeho část.

Upozornění:
Výše uvedené nic neříká a ani nepožaduje ohledně nějaké definitivní celkové specifikace požadavků. Je samozřejmé, že pro různé situace a různé modely postupu vývoje se k tomu, co se má znát a kdy, přistupuje různě. Je jasné, že architektura může, a často má být, navržena a vyzkoušena na základě požadavků podstatných pro architekturu s tím, že všechny ještě přesně vycizelovány nejsou, protože nejsou stejně důležité. Pochopitelně přístup k tomuto problému je jasně stanoven a popsán v plánu vývoje sw projektu a se zákazníkem je to dohodnuto.

1.2 Softwarová architektura

Praktika číslo 2
Název: Softwarová architektura

Znění:
Je nutno vědomě nakládat s architekturou, zjednodušeně řečeno strukturou, vyvíjeného softwarového produktu. Při (novém) vývoji to znamená architekturu navrhnout, dokumentovat a ověřit, že splňuje nároky na softwarový produkt, resp. softwarový produkt a nadřazený systém. Při (novém) vývoji to dále znamená stanovit jak se určité momenty na úrovni návrhu - např. komunikace mezi subsystémy, moduly, vyvolání služeb, atd. - budou konkrétně realizovat a toto dokumentovat. Pro detailní design a programování to po celý život daného sw produktu, tj. při jeho vývoji, rozvoji či údržbě, znamená navrženou nebo už existující architekturu striktně respektovat a striktně respektovat stanovené momenty na úrovni návrhu. Dále při návrhu architektury je třeba dodržovat základní zásady dobrého návrhu (blíže viz podstatná praktika č. 3).

Důvod:
Softwarová architektura je pracovní softwarový produkt (tj. jeden z výsledků práce při procesu vývoje softwaru, stejně tak např. plán testování je pracovní sw produkt; lze říkat softwarový pracovní produkt nebo jen pracovní produkt; nezaměňovat se softwarovým produktem zkráceně softwarem) bezpečně prvořadého významu co se pracovních produktů, které představují reprezentaci vyvíjeného sw produktu, týká. Nemá smysl porovnávat, co je důležitější, jestli architektura nebo specifikace požadavků, architektura je totiž na straně *jak* se to udělá, kdežto specifikace požadavků je na straně *co* se má udělat. Architektura, struktura sw produktu, představuje základní rámec jak technický, tak organizační pro plánování (zde ve smyslu: Work Breakdown Structure, odhady, harmonogram, zdroje) další práce primárních činností sw inženýrství (detailní analýza požadavků neovlivňujících architekturu, návrh/ detailní návrh, programování a unit testování, integrační testování) a tím pádem dalších souvisících činností (např. odborných přezkoumání). Pokud architektura není dokumentovaná, pokud nejsou stanoveny základní momenty na úrovni návrhu, pokud architektura není respektovaná a udržovaná, pokud není dodržováno řešení základních momentů, pak to vede často k velkému nárůstu času řešení drobných problémů, kdežto k jistému obrovskému nárůstu času dlouhodobé údržby či rozvoje, vede to k zvýšení pravděpodobnosti chyb při údržbě. To vše platí i pro samotný vývoj, je-li jen trochu rozsáhlejší anebo prostorově distribuovaný a časově delší. Delší čas, více defektů – horší kvalita, toto vše znamená zbytečně velké finanční ztráty.

Upozornění:
Výše uvedené neznamená, že by se architektura nemohla v čase vyvíjet (tj. měnit), a to buď vlivem požadavků anebo vlivem technických problémů při realizaci sw produktu. Architektura se vyvíjet může, s tím, že platí vše, co bylo řečeno výše. Musí existovat udržovaný popis vztahu mezi specifikací požadavků a mezi architekturou. Dokumentace architektury musí obsahovat zdůvodnění zvolených řešení a důkaz, že realizuje požadavky zákazníka.

1.3 Návrh, detailní návrh a programování

Praktika číslo 3
Název: Návrh, detailní návrh a programování

Znění:
Při návrhu, detailním návrhu a programování je nutno dodržovat základní zásady dobrého návrhu a programování v malém. Základní zásady návrhu jsou modularita, vhodně volené abstrakce, jasně definovaná rozhraní, information hiding *(daná dekompoziční jednotka zbytku systému nabízí jen vhodně volené a vhodně abstraktní rozhraní, vše ostatní je její vnitřní věc, o které nikdo nic neví a nepředpokládá, tzn. algoritmy, data, reprezentace dat atd. jsou lokální; pokud není možno fyzicky, tak logicky). Pokud je nutné použití globálních dat, tak přístup k nim musí být pouze pomocí striktně definovaných služeb nebo striktně definovaným způsobem (to pro případ nutných ohledů na výkon) a nijak jinak! Základní zásady programování v malém jsou správné používání konceptu podprogramů (lokalita dat atd.), strukturované programování, zdrojový kód splňující náležitosti ohledně jmenných konvencí, vzhledu, počtu vnoření, čitelnosti atd. (prostě minimální programovací standardy).*

Důvod:
Nedodržování základních principů a zásad návrhu a základních zásad programování v malém vede k velkému zvýšení pravděpodobnosti vzniku chyb, vede k špatné čitelnosti kódu, vede ke zhoršení udržovatelnosti. Vše má vliv na čas strávený při rozvoji či údržbě a na počet defektů – tedy kvalitu. Toto vede k zbytečným finančním ztrátám.

Upozornění:
Prostředky implementace se od sebe liší, např. C++, Delphi, COBOL, SQL spolu s 4GL atd. Každý poskytuje jiné jazykové konstrukty o různé mocnosti. Stejně tak se od sebe liší metody a notace používané pro návrh,

pokud jsou vůbec nějaké používány. Proto je vhodné obecné nároky na dobrý design a programování převést do řeči konkrétního implementačního prostředí, vývojového prostředí a případné metody a notace používané pro návrh. Čili mít programové standardy a standard pro design pro daný kontext. Příklad programovacích standardů pro C++ je uveden v kapitole č. 3

1.4 Testování

Praktika číslo 4
Název: Testování

Znění:
Je nutno provádět testování na úrovni programové jednotky (software unit - samostatně přeložitelná část programu/ jednotka zadání práce pro kódování), na úrovni integrování (celý sw produkt je postupně skládán z jednotlivých dekompozičních jednotek), na úrovni celého sw produktu (zde je základem kvalifikační testování, které musí být provedeno na straně dodavatele před předáním k akceptačnímu testování na straně zákazníka, dále existuje mnoho různě zaměřených typů testování na úrovni celého systému). Testování softwarového produktu je nutno systematicky postupně a průběžně plánovat a ve správný čas systematicky postupně a průběžně provádět. Schematicky řečeno to znamená: (i) spolu s psaním specifikace požadavků začít plánovat kvalifikační testování na straně dodavatele a začít formulovat akceptační kriteria pro převzetí vyvíjeného sw produktu nebo jeho části, (ii) spolu s dekompozicí vyvíjeného sw produktu, tj. v době návrhu architektury, příp. v době jemnější dekompozice, začít plánovat integrační testování (koncepční podklady musí dodat ti, kteří provádějí dekompozici a určují náležitosti vztahu a komunikace jednotlivých částí sw produktu – platí obecně pro všechny úrovně dekompozice, které se ještě nepovažují za unit ve slova smyslu unit testování), (iii) spolu s detailním návrhem jednotlivých programových jednotek (software unit) je třeba stanovit, jak se mají units testovat, (iv) před daným typem testování musí být jasně stanoveno, co se má testovat (testovací případy) a jak se to má testovat (testovací postupy), (v) vlastní proces testování je dokumentován a prováděn následovně: (a) po naprogramování zadané práce programátorem je prováděno testování na úrovni unit (může být provedeno stejným programátorem), (b) při skládání sw produktu z jednotlivých částí je prováděno testování na úrovni integrace, které vrcholí testováním na úrovni celého sw produktu, (c) před předáním sw produktu je prováděno kvalifikační testování. Jedná-li se o rozvoj nebo údržbu existujícího softwarového produktu, pak vše výše uvedené platí pro části přímo zasažené a související s pracemi konanými rozvojem či údržbou.

Důvod:
Čím déle chyba setrvává v systému, tím je obecně dražší ji odstranit. Čím blíže k svému vzniku je chyba objevena, tím snazší je ji odstranit. Koncepční podklady pro testování musí postupně dodat ti, kteří specifikují požadavky, dekomponují systém a provádí detailní návrh. Pokud se proces testování těmto zásadám zpronevěřuje, pak dochází zbytečně k pozdnímu objevování chyb a tím časovým ztrátám při jejich analýze a odstranění, dochází k nárůstu neobjevených chyb a tím zhoršení kvality. Ve svém důsledku dochází k zbytečným finančním ztrátám. Např. pokud se testování odkládá až na konec před vlastní dodání systému, s tím, že se průběžně netestovalo na úrovních unit a integrace nebo nedostatečně a nebylo-li kvalifikační testování včas plánováno dle specifikace požadavků, potom v časové tísni před dodáním, nejen že se neotestuje dostatečně na úrovni systému, ale objevené chyby může dlouho trvat analyzovat a můžou se ukázat závažné problémy, které měly být objeveny dříve. Toto vede ve svém důsledku k daleko větším časovým zdržením a finančním ztrátám, než to, co se předtím „ušetřilo“ na pořádné přípravě a provádění testování.

Upozornění:
Výše popsaný nárok na proces testování říká, co je principielně nutno: principielně je nutno testovat na úrovni unit, integrace a celého systému a toto je nutno včas plánovat a systematicky provádět. Zde nenárokujeme ani konkrétní role ani konkrétní míru formality ani konkrétní techniky, např. pro unit testování.

1.5 Konfigurační řízení

Praktika číslo 5
Název: Konfigurační řízení

Znění:
Je nutno provádět konfigurační řízení (alespoň v minimální podobě).

Důvod:
Konfigurační řízení zajišťuje pořádek a řád v pracovních softwarových produktech (technických artefaktech), které tvoří vyvíjený softwarový produkt. Pokud vyvíjený sw produkt existuje ve více verzích, je udržován na

více místech, vyvíjen na více místech atd., tak bezprostřední nutnost elementární kázně, kterou zajišťuje konfigurační řízení velmi roste. Bez disciplíny konfiguračního řízení může docházet ke zmatku co vlastně tvoří vyvíjený sw produkt verze x, co je otestováno a co není, co je opraveno a co není, co je změněno a co není atd.

Upozornění:
V tomto podstatném nároku jde primárně o zajištění, že se ví, co tvoří vyvíjený sw produkt daných verzí a daného stupně rozpracovanosti.

1.6 Odborná přezkoumání

Praktika číslo 6
Název: Odborná přezkoumání

Znění:
Pro určitou minimální množinu softwarových pracovních produktů včetně plánů je nutno provádět odborná přezkoumání a následné nápravné akce. Tato přezkoumání a nápravné akce je třeba provést nejpozději předtím, než na základě daného sw pracovního produktu je prováděno podstatné množství práce anebo jsou činěna zásadní rozhodnutí (např. na základě relevantních požadavků zákazníka se navrhuje architektura, na základě plánu testování se testuje, na základě plánu sw projektu probíhá celý projekt, na základě architektury a detailního designu pracují programátoři, odladěný a otestovaný unit kód se integruje, atd.). (Pozn. softwarovým pracovním produktem se myslí cokoli, co vzniká v průběhu projektu vývoje softwaru, tj. specifikace požadavků zákazníka, návrh sw architektury, detailní design, zdrojový kód, plán testování, naměřené údaje, plán sw projektu, plán konfiguračního řízení, hlášení problému atd., lze říkat jen pracovní produkt; je vidět, že mezi pracovní produkty patří i plány, nicméně pro jistotu bude někdy použita formulace sw pracovní produkty včetně plánů). Odborným přezkoumáním pro účely těchto podstatných praktik pro začátek budeme rozumět přezkoumání sw pracovního produktu osobou nebo osobami, které mají dostatečné odborné znalosti za účelem zjištění, zda sw pracovní produkt neobsahuje chyby a problémy (tj. že odpovídá na něj kladeným kriteriím, která jsou několikerého druhu: (a) vyplývají z podstaty věci, tj. kód realizuje detailní design, integrační testování odpovídá dekompozici atd.; (b) vyplývají z obecných nároků na daný pracovní produkt, tj. předpis pracovního produktu, standard pracovního produktu, zásady pro provádění příslušné činnosti produkující daný pracovní produkt atd.; (c) další). V rámci odborných přezkoumání může být případně i doporučení řešení problému.

Důvod:
Základní důvod, proč se mají provádět odborná přezkoumání, je zamezit šíření jednou vnesených chyb průběhem života vyvíjeného sw produktu a sw projektu. Čím déle zůstává chyba neobjevena, tím je těžší a dražší ji odstranit. Jednoduchá logika říká a empirie potvrzuje, že věnovat pár chvilek poctivému přezkoumání např. zadání práce pro programátora, podle kterého má pracovat např. týden se v případě, že zadání není zcela v pořádku, musí velmi vyplatit; není třeba komentovat příklad, že neobjevený problém ve specifikaci zákazníků přejde až k akceptaci. Další důvod proč dělat přezkoumání je zajistit dodržování zásad a standardů, které byly stanoveny, např. bylo z velmi dobrých důvodů stanoveno, že návrh a posléze zdrojový kód musí splňovat ty a ty zásady, např. přístup ke globálním proměnným se zásadně děje jen a pouze přes definované služby. V případě, že se sw pracovní produkty systematicky nepřezkoumávají a tím pádem podklady pro další práci nejsou kontrolovány a není kontrolováno, zda jsou dodržovány stanovené zásady, tak to vede k prodloužení doby života chyb a problémů v životě vyvíjeného sw produktu a projektu vývoje softwaru, což ve svém důsledku vede ke zhoršení kvality vyvíjeného sw produktu, k časovým a tím i finančním ztrátám – opět zbytečně.

Upozornění:
Dobrou pomůckou pro provádění odborných přezkoumání jsou kontrolní seznamy (checklists), které obsahují seznam věcí, na které se zaměřit, resp. na které nezapomenout, při odborném přezkoumání dané věci. Typické kontrolní seznamy jsou dány k dispozici (viz kapitola č. 3) pro převzetí do dané konkrétní praxe (princip zůstává, konkrétnosti se přizpůsobí). Další významnou roli v odborných přezkoumáních hrají předpisy sw pracovních produktů – přezkoumání musí zajistit, že přezkoumávaný pracovní produkt odpovídá předpisu na tento typ pracovního produktu. Předpisy pro všechny softwarové pracovní produkty včetně plánů jsou dány k dispozici (viz kapitola č. 3).

Z toho, že např. pracovní produkt požadavky zákazníka má být přezkoumán před tím, než se na základě něj začne tvořit architektura sw produktu atp. nevyplývá nic o modelech postupu vývoje (modelech SDLC – Software/ System Development Life Cycle, modelech životního cyklu) a už vůbec ne, že by měl být typu *waterfall*. Naopak odborná přezkoumání mají být plánována v závislosti na použitém typu modelu postupu vývoje tak, aby odrážela vzájemný vztah činností a pracovních produktů, který použitý model SDLC předepisuje!

Odborných přezkoumání existuje několik typů (inspekce, review, walkthrough) lišících se od sebe mírou formálnosti, postupem a zaměřením. Zde v podstatné praktice pro začátek není nárokován žádný konkrétní typ. Jediné, o co teď jde, je, aby kritické sw pracovní produkty byly poctivě i když jednoduše přezkoumány než se na základě nich bude dále pracovat a rozhodovat. (Pozn. empirie ukázala, že nejefektivnější je nejrigoróznější typ odborných přezkoumání a to formální inspekce, nicméně tento typ je spíš výsadou zkonsolidovaného uvědomělého prostředí než hromadný jev; domnívám se, že v běžné organizaci by se mělo začít poctivě s jednoduchým typem odborného přezkoumání – lepší něco jednoduchého avšak skutečně praktikovaného než nepovedený pokus o formální inspekce Faganova typu.)

Celková zastřešující činnost (proces), která se stará, aby jednotlivé pracovní produkty v průběhu projektu odpovídaly nárokům na ně kladeným a aby vyvíjený sw produkt jako celek splňoval to, co od něj zákazník/ uživatel očekávají, se nazývá validace a verifikace (V&V). Zde v počátečních nárocích tento „honosný" termín nebyl záměrně použit, aby zbytečně nevyvolával pocit něčeho divného co se nás netýká. Ve skutečnosti nárok na plánování a provádění kvalifikačního testování (viz číslo 4), nárok na průběžné plánování a provádění testování (viz číslo 4) a nárok na odborné přezkoumávání kritických pracovních produktů, než jdou ve vývoji dále, tedy nároky, o nichž nikdo nemůže říci, že jsou nerozumné a nepotřebné, tak tyto nároky tvoří podstatu toho co se nazývá V&V.

Osoby odpovědné za odborná přezkoumání nejsou totožné s osobami odpovědnými za daný pracovní produkt. Toto však nevylučuje účast osob odpovědných za pracovní produkt z účasti na odborných přezkoumáních.

1.7 Zajištění naplánovaných a předepsaných věcí

Praktika číslo 7
Název: Zajištění naplánovaných a předepsaných věcí

Znění:
Při projektu je nutno průběžně zajišťovat, že každá činnost vyžadovaná plánem projektu anebo jiným způsobem, je prováděna v souladu s plánem projektu a dalšími platnými nároky. Při projektu je nutno průběžně zajišťovat, že každý softwarový pracovní produkt vyžadovaný plánem projektu anebo jiným způsobem existuje a podstoupil testování, přezkoumání (zhodnocení/ evaluaci) a nápravné akce vyžadované plánem projektu a dalšími platnými nároky. Dalšími platnými nároky se rozumí nároky obsažené v předpisu softwarového procesu (nyní tento počáteční předpis, který se časem může rozšiřovat a prohlubovat), nároky požadované v kontraktu se zákazníkem a příp. další nároky. Zhodnocením/ evaluací se rozumí proces rozhodování zda činnost nebo pracovní produkt splňují na ně kladená kriteria (pozn. takto se vyhýbáme diskusím o rozdílech, vhodnosti atd. týkajících se různých typů přezkoumání, inspekcí atd.)

Důvod:
Rozhodnout se a deklarovat a vyžadovat v počátečním předpisu softwarového procesu určité věci je první krok. Vytvořit plán projektu, který odpovídá nárokům počátečního předpisu softwarového procesu a tím nárokuje dobrou praxi sw inženýrství při projektu je druhý krok. Zavázat se v kontraktu plnit určité věci také lze. Tato činnost má zajistit, že se skutečně bude dít, co je naplánováno a vyžadováno. Tato kapitola „Podstatné věci pro začátek" u každé praktiky, kterou vyžaduje, vysvětluje proč tomu tak je a co se stane, když není prováděna.

Upozornění:
Celková zastřešující činnost zajišťující, že vyžadované aktivity jsou prováděny tak. jak mají být, že vyžadované pracovní produkty existují a prošly vyžadovanými kontrolami, se nazývá zajišťování jakosti softwaru (Software Quality Assurance, SQA). Zde v „podstatných praktikách pro začátek" nebyl tento „honosný" termín záměrně použit, ze stejných důvodů jako nebyl použit termín V&V.

Tak, jak je zde tato činnost nárokována, odpovídá užšímu záběru SQA, který je použit třeba v MIL-STD-498, Software Management Guide (NASA). Existuje také jiná škola, která do SQA zahrnuje i hodnocení adekvátnosti naplánovaných testovacích aktiv, naplánovaných V&V aktivit a příp. další věci. Šíře je SQA chápána, např. v ESA PSS-05-0 standardu a v NASA Assurance Standard (a doprovodné Guidebook). V tomto počátečním předpisu se nárokuje užší vnímání SQA s tím, že adekvátnost je pro začátek bohatě zajištěna usilováním dosáhnout „podstatných praktik pro začátek" v reálné praxi.

Prostředkem pro zajištění naplánovaných a předepsaných věcí jsou v principu přezkoumání (může se jim říkat audit, certifikace, evaluace atd.). Je nutno mít prostředek, kterým se zjistí, zda se něco dělá či nikoli a zda se to dělá tak, jak se má, či nikoli. Přezkoumání, resp. odborná přezkoumání jsou stejně tak jedním ze základních prostředků V&V, zde v této kapitole dokonce jednou z podstatných praktik pro začátek. Velký rozdíl spočívá v tom, že v případě SQA jde při přezkoumáních - ať už jakéhokoli typu a jména - hlavně o disciplinu a poctivost, v případě V&V jde při přezkoumáních – ať už jakéhokoli typu a jména – hlavně o co nejefektivnější způsob zjištění chyb a problémů.

1.8 Vedení softwarového projektu

Praktika číslo 8
Název: Vedení softwarového projektu

Znění:
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 doděláváním 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.

Důvod:
Činnost (proces) vedení projektu je naprosto zásadní už v tom smyslu, že vedení projektu je odpovědné za plánování a provádění všech činností se všemi náležitostmi a vznik všech produktů se všemi náležitostmi, které tvoří dobrou vyžadovanou praxi softwarového inženýrství při projektu. Pro projekt je třeba mít plán, který obsahuje popis procesu vývoje softwaru při projektu, jinak se dopředu přesně neví, jak se práce bude provádět. Plán se logicky musí jasně vyjádřit ke všem činnostem, které se mají provádět, a to ke stránce technické, organizační a časové (pozn. nárokované předpisy pracovních produktů včetně plánů představují vlastně elementární způsob, jak nárokovat technický aspekt činností – co má být jejich výsledkem). Protože není možné na začátku projektu z principu vše detailně naplánovat, musí vedení zajistit, že plán se průběžně dodělává a zdetailňuje v souladu s tím, co už je hotovo (existuje architektura, existuje detailní návrh atd.). Protože skutečný průběh bývá jiný než plán, pak je třeba práce provádět tak, jak je naplánováno, zároveň průběh projektu sledovat, vyhodnocovat a provádět potřebné korekce plánu. Základní prostředek jak to, co je technicky a organizačně naplánováno (tedy určeno), zachytit, je: WBS – struktura dekompozice práce (která všechny práce, které se mají provést, ať už jde o programování, analýzu, monitorování či přezkoumání, dekomponuje do potřebných zvladatelných úkolů); odhadování; časový plán (harmonogram); případně síť aktivit. Je zřejmé, že WBS a z toho odvozené odhadování a časové plánování se u činností typu analýza, návrh architektury, detailní návrh, programování, testování atd. vázaných na pracovní produkty typu specifikace požadavků zákazníka, sw architektura, detailní návrh atd. musí postupně zdetailňovat a zpřesňovat spolu s tím, jak tyto produkty vznikají (např. pokud není sw architektura, tak lze těžko přesně odhadovat náročnost programování natož ho detailně plánovat). Koncepční prostředek pro plánování postupu vývoje musí být vhodný model životního cyklu, příp. více vzájemně skloubených modelů postupu vývoje (týkajících se různých úrovně abstrakce (nadřazený systém,

celý sw produkt, od detailní analýzy po implementaci v rámci částí daných celkovou sw architekturou), různých období (před/ po architektuře), různých částí sw produktu anebo systému (různé modely pro různé části)).

Bez kompletního udržovaného plánu, bez názoru na celkový postup vývoje, bez bezprostředního mechanismu jak dekomponovat a plánovat vykonání práce, bez sledování průběhu projektu, bez učení se z minulých zkušeností, nelze efektivně vést projekt tak, aby vedení zajišťovalo predikovatelnou dobu vývoje, náklady vývoje a kvalitu (počty defektů) vyvíjeného sw produktu.

Upozornění:
„Všezahrnující" plán projektu (lze též softwarový plán projektu, plán softwarového projektu atd.) může mít pochopitelně různé logické a různé fyzické členění (např. fyzicky samostatně může stát plán konfiguračního řízení, plán testování atd.). Důležité je, aby ve svém celku plán obsahoval všechny podstatné věci. Z toho důvodu je třeba vyjít z osvědčených předpisů anebo návrhů pro vzor plánu projektu (které jsou k dispozici ve standardu MIL-STD-498, standardu NASA-STD-2100-91 atd.) a tyto případně přejmout anebo upravit.

Dále to, že plán se má vyjádřit ke „všemu" není třeba a ani moc možné chápat tak, že je vše v plánu popsáno. Spousta věcí se při projektech opakuje a je daná kontextem. Např. postupy pro odborná přezkoumávání stačí mít jednou definované a napsané a pak je stačí jen referovat atd. Takto předpřipravené popisy postupů, metod, předpisy produktů atd. mají tvořit základ „podpůrných věcí" pro softwarový proces (které tvoří část software process assets dané organizace).

1.9 Mechanismus řešení problémů

Praktika číslo 9

Název: Mechanismus řešení problémů

Znění:
Při projektu je nutno mít ustanoven fungující mechanismus řešení problémů. Problémy se týkají vyžadovaných činností a vyžadovaných pracovních produktů. Problém může být identifikován při testování, při odborných přezkoumáváních, při zajišťování naplánovaných a předepsaných věcí, při společných přezkoumáních se zákazníkem atd. Identifikované problémy musí být evidovány. Musí být zahájeno jejich řešení. A musí být vyřešeny.

Důvod:
Všechny dobré praktiky softwarového inženýrství, které usilují o dobrou kvalitu (malý počet defektů) a nízké náklady, tj. správný proces testování, odborná přezkoumání atd., aby byly vskutku účinné, musí být završeny systematickým vyřešením odhalených problémů počínaje jejich evidencí a konče analýzou trendů.

Upozornění:
Tato podstatná věc (praktika) pro začátek se ode všech ostatních liší tím, že jde výhradně o věc disciplíny a poctivosti, nejde zde o žádnou ryze odbornou věc. Řešení různých typů problémů, např. zjištěných testováním a zjištěných odborným přezkoumáním může být pochopitelně administrováno konkrétně různě. Jediné, co je podstatné je, aby poctivě a včas zjištěné problémy nebyly „zapomenuty" a byly včas vyřešeny. Proto také byla tato praktika vytknuta a řečena samostatně (stejný přístup má vojenský standard MIL-STD-498 a mezinárodní standard ISO 12207).

Pro konkrétní realizaci mechanismu řešení problémů je třeba vhodně stanovit klasifikaci problémů, prioritu problémů, způsob evidence problémů, způsob sledování stavu problémů.

2 Konkrétní nároky na praxi softwarového inženýrství

Tato kapitola obsahuje formulované konkrétní nároky na praxi softwarového inženýrství vzhledem k deklarovaným podstatným praktikám pro začátek. Seznam podstatných praktik a k nim vznesených konkrétních nároků je v tabulce č. 2-1.

K způsobu stanovení konkrétních nároků: Tyto nároky jsou stanoveny tak, aby co nejvíce podpořily rychlé zavedení dané věci do praxe. Zpočátku se bude zřejmě jednat o jednoduché a základní provádění stanovených podstatných praktik. Nějak se začít musí. Takže tyto nároky jsou velmi skromné a základní, žádné moc „složité“, „vědecké“ atp. věci. To by zpočátku většinou nemělo šanci na úspěch a bylo by to ve svém důsledku kontraproduktivní.

Seznam podstatných praktik a konkrétních nároků

1. Praktika: Požadavky
   - 1.1. Dokument „Specifikace požadavků zákazníka“
   - 1.2. Zásady pro tvorbu a psaní dokumentu „Specifikace požadavků zákazníka“
2. Praktika: Softwarová architektura
   - 2.1. Pracovní produkt „Softwarová architektura“
   - 2.2. Respektování navržené, resp. už existující softwarové architektury
   - 2.3. Základní zásady dobrého návrhu
3. Praktika: Návrh, detailní návrh a programování
   - 3.1. Základní zásady dobrého návrhu
   - 3.2. Základní zásady programování v malém
4. Praktika: Testování
   - 4.1. Testování na úrovni software unit, integrace a celého systému
   - 4.2. Systematické plánování a provádění testování
5. Praktika: Konfigurační řízení
   - 5.1. Činnost konfigurační řízení
   - 5.2. Systematické plánování a provádění konfiguračního řízení
6. Praktika: Odborná přezkoumání
   - 6.1. Provádění odborných přezkoumání
   - 6.2. Přezkoumávání zásadních pracovních produktů včetně plánů
7. Praktika: Zajištění naplánovaných a předepsaných věcí
   - 7.1. Provádění činnosti, která zajišťuje naplánované a předepsané věci
   - 7.2. Systematické plánování a provádění zajištění naplánovaných a předepsaných věcí
   - 7.3. Nezávislé provádění
8. Praktika: Vedení softwarového projektu
   - 8.1. Tvorba, používání a údržba plánu projektu
   - 8.2. Předpisy/ vzory pro zásadní pracovní produkty
   - 8.3. Nezúžený záběr plánování
   - 8.4. „Základní“ metoda vedení sw projektu
   - 8.5. Postupné a průběžné plánování
   - 8.6. Vhodný a správný celkový postup vývoje softwaru, který zahrnuje vhodný(é) model(y) životního cyklu
   - 8.7. Ošetření rizik
   - 8.8. Odborné znalosti vedoucího projektu
9. Praktika: Mechanismus řešení problémů
   - 9.1. Evidence problémů
   - 9.2. Řešení problémů

Tabulka 2-1: Seznam podstatných praktik a konkrétních nároků

2.1 Požadavky

Praktika číslo 1: Požadavky
Zaměřeno na:
1. Dokument „Specifikace požadavků zákazníka“
2. Zásady pro tvorbu a psaní dokumentu „Specifikace požadavků zákazníka“

Pro začátek chceme docílit:
1. Dokument „Specifikace požadavků zákazníka“: Je třeba docílit vytvoření a údržbu dokumentu, který obsahuje specifikované požadavky zákazníka. Tento dokument musí odpovídat osvědčenému standardnímu vzoru stanovujícímu vyžadovaný obsah a strukturu.
2. Zásady pro tvorbu a psaní dokumentu „Specifikace požadavků zákazníka“: Je třeba docílit, aby při tvorbě, psaní a formulaci požadavků zákazníka byly dodržovány osvědčené základní zásady vedoucí k jasně, zřetelně formulovaným a logicky strukturovaným požadavkům.

Konkrétní nároky k počátečním cílům:
1. Vzor pro dokument „Specifikace požadavků zákazníka“ musí být přímo odvozen z příslušného předpisu (tzv. DID, Data Item Description) pro tento pracovní produkt v standardu MIL-STD-498 (konkrétně se jedná o Software Requirements Specification (SRS DID), příp. System/ Subsystem Specification (SSS DID)) nebo v standardu NASA-STD-2100-91 (konkrétně se jedná o NASA-DID-P200 Requirements). Pro inspiraci, co se týká struktury a obsahu, mohou dále sloužit předpisy obsažené v standardu ESA PSS-05-0, NASA příručkách Manager’s Handbook/ Recommended Approach, materiálech SEPO (mají vzor (template) na základě příslušného DID z MIL-STD-498) a v knize Software Requirements: A Pragmatic Approach. Část vzoru týkající se externích rozhraní musí odpovídat nárokům Interface Requirements Specification (IRS DID) ze standardu MIL-STD-498. (Pozn.: dále je možné a vhodné se inspirovat seznamy kontrolních otázek – checklists – pro tento typ pracovního produktu; viz kapitola č. 3.)
2. Psaní, formulace a vnitřní struktura zápisu jednotlivých požadavků musí odpovídat zásadám, které jsou popsány a vysvětleny v článku „Writing Effective Natural Language Requirements Specifications“ uveřejněném v časopisu CrossTalk. Dále je třeba znát základní přehled problematiky požadavků popsaný v článku „Making Requirements Management for You“ z časopisu CrossTalk.

Přídavný nárok pro případ existence nadřazeného systému:
3. Znění shodné s tímto nárokem u softwarové architektury v následující sekci č. 2.2.

Upozornění k interpretaci:
Je pochopitelně možné mít více různých konkrétních vzorů pro různé typy projektů. Dokument „Specifikace požadavků zákazníka“ musí však mít jako svůj základ obsah a strukturu odpovídající osvědčeným standardům.

Upozornění k rozsahu nároků:
Zde uvedené konkrétní nároky jsou opravdu velmi skromné. Z celé oblasti označované Requirements Engineering tvořené zjišťováním požadavků, jejich analýzou, jejich specifikací a jejich verifikací vlastně nárokujeme jen poctivou textovou specifikaci (mající náležitý obsah a strukturu) a pár zásad pro psaní této textové specifikace. Dále nijak zde neřešíme vztah mezi požadavky na nadřazený systém a vlastní sw produkt, který je jeho součástí; neřešíme zde rozlišování požadavků uživatele a požadavků na software. I tak, podaří-li se učinit tyto skromné nároky pro začátek součástí široké praxe, bude to dobrý a stabilní základ pro další pokrok v této oblasti. Pozn. analogicky toto platí i pro další podstatné praktiky a příslušné konkrétní nároky pro začátek.

2.2 Softwarová architektura

Praktika číslo 2: Softwarová architektura
Zaměřeno na:
1. Pracovní produkt „Softwarová architektura“
2. Respektování navržené, resp. už existující softwarové architektury
3. Základní zásady dobrého návrhu

Pro začátek chceme docílit:
1. Pracovní produkt „Softwarová architektura“: Je třeba docílit, aby v rámci činnosti návrh softwaru vznikl a po té byl po celý život vyvíjeného sw produktu udržován pracovní produkt softwarová architektura. Pracovní produkt softwarová architektura musí odpovídat standardním požadavkům, které jsou kladeny na tento produkt.
2. Respektování navržené, resp. už existující softwarové architektury: Je třeba docílit, aby při činnostech návrh, detailní návrh a programování po celý život vyvíjeného sw produktu, tj. při jeho vývoji, rozvoji či údržbě, navržená, resp. už existující softwarová architektura byla striktně respektována včetně stanovených momentů na úrovni návrhu.
3. Základní zásady dobrého návrhu: Je třeba docílit, aby návrh softwarové architektury respektoval základní zásady dobrého návrhu.

Konkrétní nároky k počátečním cílům:

1. Pracovní produkt softwarová architektura musí splňovat nároky, které stanovuje Software Design Description (SDD DID), konkrétně se jedná o část CSCI architectural design a není-li zpracována jinde i o část CSCI-wide design decisions. (Pozn. CSCI, Computer Software Configuration Item představuje ty dekompoziční jednotky nadřazeného systému, které jsou realizovány softwarem tedy softwarovým produktem.) Návrh rozhraní musí odpovídat předpisu Interface Design Description (IDD DID). Návrh databáze, je-li v systému, musí odpovídat předpisu Database Design Description (DBDD DID). Vždy se jedná o DID ze standardu MIL-STD-498. Dále je vhodné se inspirovat předpisy tohoto pracovního produktu ve standardu NASA-STD-2100-91 (konkrétně se jedná o NASA-DID-P300 Architectural Design), standardu ESA PSS-05-0, NASA příručkách Manager's Handbook/ Recommended Approach. Dále je nutno znát vzor pro specifikaci návrhu od fy Construx (Software Design Specification) a tento respektovat. Návrh softwarové architektury musí obsahovat stanovení jednotlivých momentů na úrovni návrhu. Ten, kdo navrhuje nebo přezkoumává softwarovou architekturu, musí mít o této problematice obecné znalosti aspoň na úrovni odpovídající znalosti následujících textů: The 4+1 View Model of Architecture (Krutchen), An Introduction to Software Architecture (Garlan&Shaw), Software Architecture: An Executive Overview (Clements), On the Definition of Software System Architecture (Boehm). (Pozn.: dále je možné a vhodné se inspirovat seznamy kontrolních otázek – checklists – pro tento typ pracovního produktu; viz kapitola č. 3.)
2. Výstup práce, tj. příslušné pracovní produkty, z činností (návrh), detailní návrh a programování musí být průběžně sledován pomocí odborných přezkoumání. Jedním z cílů těchto odborných musí být zjišťování, zda nedochází k nerespektování navržené, resp. existující softwarové architektury a zda nedochází k nerespektování stanovených momentů na úrovni návrhu (např. že parametry se v té a té situaci předávají tak a tak). Tento konkrétní nárok musí být obsažen v používaných checklists. Konkrétní nároky na podstatnou praktiku odborná přezkoumání jsou níže v sekci č. 2.6.
3. Návrh softwarové architektury musí odpovídat těm zásadám dobrého návrhu, které mají pro tuto úroveň návrhu smysl. Samotné zásady dobrého návrhu jsou předmětem další podstatné praktiky a příslušných konkrétních nároků, viz sekce č. 2.3.

Přídavný nárok pro případ existence nadřazeného systému:

4. Je-li vyvíjený software součástí většího systému (tj. některé části systému jsou realizovány softwarem) potom je třeba, aby se tým odpovědný za software účastnil aktivit na úrovni vývoje nadřazeného systému. Konkrétně se jedná o aktivity specifikace systému a návrh systému. Výstupem z těchto aktivit musí být příslušné pracovní produkty, které odpovídají následujícím nárokům na tyto produkty: Operational Concept Description (DID OCD), System/ Subsystem Specification (DID SSS), System/ Subsystem Design Description (DID SSDD). Vždy se jedná o DID ze standardu MIL-STD-498. V případě existence nadřazeného systému pak jeho specifikace a návrh tvoří zásadní vstup pro klasicky chápanou činnost požadavky zákazníka na software a dále pro návrh softwarové architektury. Termín účastnit se znamená plně odpovídat pokud je systém tvořen výhradně softwarem.

Upozornění k interpretaci:

(a) Tzv. DIDs (Data Item Descriptions) obsažené v standardu MIL-STD-498 představují jedinečnou věc. Jsou to předepsané principiální nároky na důležité pracovní produkty včetně plánů. Tyto DIDs, tj. předpisy ani samotný standard MIL-STD-498 nenárokují konkrétní podobu pracovního produktu, nárokují pouze esenci. Tedy zda produkt softwarová architektura má podobu dokumentu nebo je obhospodařován v nástroji typu CASE, je z tohoto pohledu nepodstatné. Zda dokument má přesně strukturu takovou nebo jinou je nepodstatné. Zda pro znázornění návrhu je použita notace X či Y je nepodstatné. DIDs jsou zcela neutrální vzhledem k metodám vývoje a modelům postupu vývoje. Takto představují vskutku neocenitelný zdroj – představují totiž mimo jiné stanovení základních nároků na odpovídající činnosti. Jiná věc je, že z praktických důvodů je pro projekt (nebo pro daný kontext, třeba tým) třeba mít nějaké konkrétní závazné vzory pro dokumenty, pro pracovní produkty, atd. Tyto lze získat bezprostředním převzetím těch vzorů a DIDs, které jsou zde dávány k dispozici, popř. jejich přizpůsobení danému kontextu. Vynikající východisko pro vzory dokumentů je standard NASA-STD-2100-91 NASA Software Documentation Standard, který představuje zdroj vzorů pro dokumenty odpovídající pracovním produktům včetně plánů a různých formulářů plně pokrývající potřeby sw projektu. Dále je vhodné se inspirovat vzory a předpisy na dokumenty a pracovní produkty ve standardu ESA PSS-05-0, NASA příručkách Manager's Handbook/ Recommended Approach, materiálech SEPO a dalších zde uváděných zdrojích (viz kapitola č. 3). Tato poznámka je obecná a nebude už na jiných místech opakována.

(b) CSCI (Computer Software Configuration Item) představuje část systému realizovanou softwarovým produktem. Takovýchto částí může být v systému více.

(c) V případě, že existuje nadřazený systém, pak pro činnosti specifikace požadavků zákazníka a návrh softwarové architektury vzhledem k produktům specifikace systému a architektura systému platí analogicky konkrétní nárok č. 2 pro tuto podstatnou praktiku č. 2.

2.3 Návrh, detailní návrh a programování

Praktika číslo 3: Návrh, detailní návrh a programování
Zaměřeno na:
1. Základní zásady dobrého návrhu
2. Základní zásady programování v malém

Pro začátek chceme docílit:
1. Základní zásady dobrého návrhu: Je třeba docílit, aby činnosti návrh a detailní návrh byly prováděny v souladu se základními principy a zásadami dobrého návrhu.
2. Základní zásady programování v malém: Je třeba docílit, aby činnost programování byla prováděna v souladu se základními zásadami dobrého programování v malém.

Konkrétní nároky k počátečním cílům:
1. Při činnosti návrh a detailní návrh je nutno dodržovat principy a zásady určující, co to je z technického hlediska správný a dobrý návrh. Konkrétně se jedná o principy a zásady týkající se dekompozice/ modularity, způsobu volení abstrakcí, rozhraní a tzv. *information hiding*. Ve svém souhrnu jsou tyto principy a zásady stručně a dobře popsány a vysvětleny v následujících článcích: (i) On the Criteria to be Used in Decomposing Systems into Modules (D.L. Parnas, ACM Classic of the month, článek o tom jak provádět dekompozici; zavedení konceptu *information hiding*); (ii) Why You Should Use Routines ... Routinely (jak správně a všestranně používat koncept podprogramu); (iii) Keep It Simple (stručný přehled všech základních koncepčních zásad pro design a programování); (iv) Missing in Action: Information Hiding (volání po systematickém používání konceptu *information hiding*). (Články ii až iv jsou z Best Practices Column v IEEE Software.) Dále pracovní produkt detailní návrh musí splňovat náležitosti předepsané částí CSCI detailed design v předpisu Software Design Description (SDD DID) ze standardu MIL-STD-498. Co se konečné konkrétní podoby pracovního produktu detailní návrh týká, pak je třeba se dále inspirovat v předpisech, které jsou k dispozici v NASA Software Documentation Standard NASA-STD-2100-91 (konkrétně NASA-DID-P400 Detailed Design), standardu ESA PSS-05-0, NASA příručkách Manager's Handbook/ Recommended Approach. Dále je nutno znát vzor pro specifikaci návrhu od fy Construx (Software Design Specification) a tento respektovat. (Pozn.: dále je možné a vhodné se inspirovat seznamy kontrolních otázek – checklists – pro tento typ pracovního produktu; viz kapitola č. 3.)
2. Při činnosti programování je nutno dodržovat zásady dobrého programování v malém. Tyto jsou stručně uvedeny v příloze B normy 24/97. Dále, protože detailní návrh a programování se často překrývají, tak pro programování, tak, jak to má smysl, platí stejné zásady uvedené o bod výše. Konkrétně se jedná o zásady z článků (i) Why You Should Use Routines ... Routinely; (ii) Keep It Simple; (iii) Who Cares About Software Constructions? Dále je třeba stanovit a používat programovací standardy, tj. nárok na pracovní produkt zdrojový kód, v rozsahu nároků přílohy B normy 24/97; jako inspirace mohou sloužit programovací standardy pro C++ uvedené v sekci č. 3.3. Dále je nutno znát seznamy kontrolních otázek (checklists) pro oblast konstrukce od fy Construx a tyto respektovat. (Pozn.: dále je možné a vhodné se inspirovat seznamy kontrolních otázek – checklists – pro tento typ pracovního produktu; viz kapitola č. 3.)

Upozornění k interpretaci:
Výše uvedené nároky se pochopitelně snadněji realizují v mocnějších prostředích typu C++, Ada atd., které nabízejí konstrukty na úrovni jazyka podporující dobré zásady (návrhu), detailního návrhu a programování. Např. konstrukt třída v C++ se všemi náležitostmi nebo podobně package v jazyku Ada. Pozn. toto bylo do jazyků dodáno právě na základě principielních zjištění o návrhu v předcházejících desetiletích.

2.4 Testování

Praktika číslo 4: Testování
Zaměřeno na:
1. Testování na úrovni software unit, integrace a celého systému
2. Systematické plánování a provádění testování

Pro začátek chceme docílit:
1. Testování na úrovni software unit, integrace a celého systému: Je třeba docílit, aby se provádělo unit testování, integrační testování a testování na úrovni celého systému.
2. Systematické plánování a provádění testování: Je třeba docílit, aby činnost testování byla systematicky a včas plánována a systematicky prováděna a to včetně regresního testování.

Konkrétní nároky k počátečním cílům:

1. Unit testování, integrační testování a testování na úrovni systému je třeba chápat a z technického hlediska provádět, tak jak je stručně popsáno v sekci 5.3.2.3 Testing, části I a v sekci 4.2.4 Testing části II Standardu ESA PSS-05-0. Principielně to samé, ale jinými slovy je též velmi stručně popsáno v sekcích Software Implementation and Unit Testing, Software Integration and Testing, Software CI Qualification Testing v příručce Software Management Guidebook od NASA. Dále minimálně ten, kdo testování plánuje a je za něj celkově odpovědný, by měl znát minimum typu Little Book of Testing, Volume I: Overview and Best Practices a Little Book of Testing, Volume II: Implementation Technique. V těchto „malých knížkách" jsou stručně popsány principy, činnosti, postupy a vazby na okolí celkového procesu testování.
2. Testování na dané úrovni je třeba začít plánovat včas a vzhledem k příslušným vstupům. Schematicky to znamená: testování na úrovni systému (ať už se jedná o kvalifikační, akceptační či jiné) začít plánovat již v době, jakmile jsou k dispozici specifikované požadavky uživatele; integrační testování začít plánovat spolu s prováděním dekompozice systému tj. v době návrhu architektury; unit testování plánovat spolu s tvorbou detailního návrhu. *Co* se má *kdy* plánovat – mluvíme o procesu testování – je stručně popsáno v příslušných částech v sekci 4.4 Evolution of the SVVP Throughout the Life Cycle, část II ve standardu ESA PSS-05-0. Principielně to samé je popsáno v příslušné části příručky Software Assurance Guidebook, NASA-GB-A201. Výsledky plánování procesu testování musí být zaznamenány v plánu testování. Plán testování musí svým obsahem odpovídat specifikaci pro testování ve standardu ESA PSS-05-0, appendix C.10 (anebo podobném, např. NASA Software Documentation Standard). Dále, pro konkrétní formát a strukturu plánu testování je třeba se inspirovat příslušnými předpisy a vzory na tento typ pracovního produktu v standardu MIL-STD-498 (Software Test Plan DID, Software Test Description DID), v standardu NASA-STD-2100-91 (příslušná část NASA-DID-M400 Assurance Plan, NASA-DID-A200 Test Procedures), v NASA příručkách Manager's Handbook/ Recommended Approach. Testování je třeba podle plánů provádět a výsledky zaznamenávat. Zaznamenání ať má podobu zpráv z testování, které odpovídají předpisům na tento pracovní produkt v standardu MIL-STD-498 (Software Test Report DID), NASA-STD-2100-9 (NASA-DID-R009 Test Report). (Pozn.: pro to, co mají plány testování obsahovat, je možné a vhodné se inspirovat seznamy kontrolních otázek – checklists – pro tento typ pracovního produktu; viz kapitola č. 3.)

Přídavný nárok pro případ existence nadřazeného systému:

3. Je-li vyvíjený software součástí většího systému (tj. některé části systému jsou realizovány softwarem), potom je třeba, aby tým odpovědný za software participoval v aktivitách na úrovni vývoje systému. Konkrétně se jedná o aktivity integrační testování systému (tedy integrace jednotlivých částí systému a příslušné testování na této úrovni) a testování na úrovni celého systému. Termín participovat znamená účastnit se v případě, že systém obsahuje části realizované softwarovými produkty. Termín participovat znamená plně odpovídat pokud je systém tvořen výhradně softwarem.

Upozornění k interpretaci:

(a) Platí-li přídavný nárok č. 3, potom by ve formulaci nároků č. 1 a 2 mělo být radši použito termínů unit testování, integrace unit a testování, testování na úrovni jednotlivých částí systému realizovaných softwarem (v MIL-STD-498 označovaných jako CSCI, Computer Software Configuration Item). Takto je např. formulován standard MIL-STD-498. Byl však použit obvyklejší jednoduší přístup.

(b) Na úrovni systému lze provádět několik typů testů (např. zátěžový, mezní atd.). Vždy je však na straně dodavatele nutno provést tzv. kvalifikační test před předáním softwaru k akceptaci. Kvalifikační test v podstatě odpovídá akceptačnímu až na to, že je prováděn na straně dodavatele. Kvalifikační test je tedy prováděn vzhledem k specifikaci požadavků uživatele. Testování na úrovni systému je zde myšleno jak ve smyslu nároků č. 1 a 2, tak ve smyslu nároku č. 3.

(c) Plán testování, který je zde nárokován může mít obecně různý formát a strukturu a obecně může být v různém fyzickém vztahu vzhledem k plánu softwarového projektu. Nicméně je maximálně vhodné připravit na základě předpisů a vzorů, které jsou pro tento pracovní produkt dány k dispozici závazné vzory. Ukáže-li se to vhodné, pak vzorů může být několik pro různé typy projektů. O definitivní podobě a definitivním začlenění, fyzickém i logickém, plánu testování (v tomto kontextu jedno z dílčích plánů) do celkového plánu softwarového projektu nakonec rozhoduje vedení projektu (viz praktika a konkrétní nároky č. 8). Vynikající zdroj a inspirace pro celkové koncipování dokumentace softwarového projektu, které pokrývá plán sw projektu (tedy základní dokument projektu), všechny důležité dílčí plány (např. plán konfiguračního řízení) a všechny další důležité pracovní produkty (např. sw architektura) je NASA Software Documentation Standard NASA-STD-2100-91 (tento je i velmi detailní), standard ESA PSS-05-0 (konkrétně jde o přílohy B, C a E), standard MIL-STD-498 (konkrétně jde o čtveřici plánů: Software Development Plan (SDP DID), Software Test Plan (STP DID), Software Installation Plan (SIP DID), Software Transition Plan (STrP DID) spolu s ostatními předpisy – formou DIDs – zásadních pracovních produktů; s tím že je třeba poznamenat, že předpisy – formou DIDs – pro zásadní

pracovní produkty poskytnuté ve standardu MIL-STD-498 mají primární cíl vyžadovat principielní podstatu a nikoli bezprostředně působit jako vzor (template) příslušného dokumentu – ne vždy pracovní produkt nutně musí mít formu dokumentu) a do značné míry i dvě příručky Recommended Approach to Software Development (NASA, SEL-81-305) a Manager's Handbook for Software Development (NASA, SEL-84-101). Tato poznámka je obecná, týká se stejně tak plánu konfiguračního řízení, plánu zajišťování jakosti atd., a nebude již na jiných místech opakována.

(d) Po opravě nalezené chyby či po jakékoli změně na již otestované části systému by se mělo provádět tzv. regresní testování. Regresní testování by mělo zajistit, že provedené změny mají jen žádané účinky a žádné vedlejší efekty. Praxe ukázala, že velmi efektivní je automatizované opakování příslušných sad testů. Toto však nemusí být k dispozici. Proto je třeba k problému neustálého ručního přetestování přistupovat citlivě.

(e) Plány pro úroveň unit testování bývají neformální. Plány pro úroveň testování systému musí být formální. Na přípravě plánů mají spolupracovat autoři programů, ale testování na úrovni integrace a systému musí být nezávislé.

2.5 Konfigurační řízení

Praktika číslo 5: Konfigurační řízení
Zaměřeno na:
1. Činnost konfigurační řízení
2. Systematické plánování a provádění konfiguračního řízení

Pro začátek chceme docílit:
1. Činnost konfigurační řízení: Je třeba docílit, aby se prováděla činnost konfigurační řízení. Byť by to mělo být na základní úrovni a maximálně jednoduchým způsobem.
2. Systematické plánování a provádění konfiguračního řízení: Je třeba docílit, aby činnost konfigurační řízení byla systematicky a včas plánována a systematicky v souladu s plány prováděna.

Konkrétní nároky k počátečním cílům:
1. Činnost konfigurační řízení je třeba chápat a provádět, tak jak je stručně popsána v sekci 3.2 části II Standardu ESA PSS-05-0. Dále minimálně ten kdo konfigurační řízení plánuje a je za něj celkově odpovědný by měl znát minimum typu Little Book of Configuration Management, příp. Software Configuration Management Technologies and Applications (STSC Technology Report).
2. Věci, které nelze určit na začátku projektu anebo věci, s jejichž stanovením je výhodné počkat, musí být v každém případě naplánovány včas. *Co* se má *kdy* naplánovat – mluvíme o procesu konfiguračního řízení – je stručně popsáno v sekci č. 3.4 Evolution of the SCMP Throughout the Life Cycle, část II ve standardu ESA PSS-05-0. Dále činnost konfigurační řízení musí být stanovena a naplánována v plánu konfiguračního řízení. Plán konfiguračního řízení musí svým obsahem odpovídat modelovému plánu uvedeném v technické zprávě Configuration Management (CM) Plans: The Beginning to Your CM Solution ze Software Engineering Institute. Dále pro konkrétní formát a strukturu plánu konfiguračního řízení je vhodné se inspirovat příslušnými předpisy a vzory na tento typ pracovního produktu v standardu ESA PSS-05-0, v příručce MIL-HDBK-61, v materiálu Configuration Management Plan Outline (Software Productivity Center), v NASA dokumentačním standardu NASA-STD-2100-91.

Upozornění k interpretaci:

(a) Opravdu s rozvahou je třeba volit, *co* se *kdy* stane předmětem konfiguračního řízení. Moc jemná granularita položek, nad kterými operuje konfigurační řízení, a jejich moc brzké zařazení pod dozor konfiguračního řízení není dobré, může způsobit velkou režii a zbytečnou nic nepřinášející „byrokratizaci". Např. pro OO prostředí se třída, resp. objekt obecně považují za položky s dost jemnou granularitou. Jednoduše, tyto věci je třeba si velmi dobře rozmyslet a snažit se získat zdroj s odpovídajícími zkušenostmi pro danou konkrétní situaci.

(b) Plán konfiguračního řízení, který je zde nárokován, může mít obecně různý formát a strukturu a obecně může být v různém fyzickém vztahu vzhledem k plánu softwarového projektu. Nicméně je maximálně vhodné připravit na základě předpisů a vzorů, které jsou pro tento pracovní produkt dány k dispozici, závazné vzory. Ukáže-li se to vhodné, pak vzorů může být několik pro různé typy projektů. Dále platí vše, co bylo v této souvislosti řečeno v upozornění k interpretaci (c) u konkrétních nároků na testování (praktika č. 4).

2.6 Odborná přezkoumání

Praktika číslo 6: Odborná přezkoumání
Zaměřeno na:
1. Provádění odborných přezkoumání

2. Přezkoumávání zásadních pracovních produktů včetně plánů

Pro začátek chceme docílit:
1. Provádění odborných přezkoumání: Je třeba docílit, aby byla prováděna odborná přezkoumání, třeba byť základním a velmi jednoduchým způsobem. Je třeba stanovit postup odborných přezkoumání, tedy to, jakým konkrétním způsobem budou prováděna, a tento postup dodržovat.
2. Přezkoumávání zásadních pracovních produktů včetně plánů: Je třeba docílit, aby bylo včas a systematicky plánováno a systematicky prováděno odborné přezkoumání zásadních pracovních produktů, na kterých závisí úspěch projektu a další život vyvíjeného softwarového produktu.

Konkrétní nároky k počátečním cílům:
1. Činnost odborné přezkoumání je třeba provádět podle stanoveného postupu. Tento postup je třeba stanovit tak, aby na jednu stranu byl v daném kontextu vůbec realizovatelný (prostě nepřehánět to s formálností, počtem rolí atd.), ale zároveň aby na druhou stranu to pořád bylo smysluplné odborné přezkoumání, které efektivně hledá defekty a nesrovnalosti v pracovních produktech. (Pozn. samozřejmě se předpokládá i zajištění vyřešení identifikovaných defektů a nesrovnalostí.) Ti, kdo tento postup stanovují a jsou za odborná přezkoumání zodpovědní, musí znát standardní typy odborných přezkoumání (Walkthrough, Technical Review a Formal Inspections). Tyto standardní základní typy odborných přezkoumání jsou popsány v SEPO příručce Peer Review Process a formální inspekce jsou podrobně popsány v NASA příručce Software Formal Inspection Guidebook, popř. v dokumentu Formal Inspection Process od SEPO. Dále musí znát problémy způsobené snahou zavést nepřiměřeně vyspělý proces odborných přezkoumání do prostředí, které na to není připraveno. Tento problém je probírán v článcích Software Inspections & Technical Reviews: Transcending the Dogma, Formal Technical Reviews Across All Maturities a v článku Software Inspections: How to Diagnose Problems and Improve the Odds of Organizational Acceptance. Jedním z podkladů vstupujících do procesu přezkoumání jsou tzv. kontrolní seznamy (checklists), které sumarizují, co u daného pracovního produktu sledovat. Pro přezkoumávané pracovní produkty je třeba mít předem vypracované checklisty. Ty lze vypracovat na základě sad checklistů obsažených v textech Software Formal Inspections Guidebook od NASA a Formal Inspection Process od SEPO. Sadou checklists, které dává k dispozici firma Construx. Dále se lze inspirovat v materiálech Weiss and Kimbrough Inspections Materials firmy Motorola a materiálem An Abbreviated C++ Code Inspection Checklist. Dále, existuje-li, je třeba pro tvorbu checklistu využít předpis daného pracovního produktu ze standardu MIL-STD-498 (tedy vhodné DID). Dále kontrolní seznam (checklist) musí odpovídat konkrétnímu zvolenému předpisu či vzoru na daný pracovní produkt.
2. Přezkoumání je třeba včas a vhodně plánovat. *Co* se má *kdy* plánovat – mluvíme o procesu odborných přezkoumání – je stručně popsáno v příslušných částech v sekci 4.4 Evolution of the SVVP Throughout the Life Cycle, část II ve standardu ESA PSS-05-0. Principielně to samé je popsáno v NASA příručce Software Formal Inspection Guidebook a v příslušné části NASA příručky Software Assurance Guidebook. Pro začátek je minimální seznam pracovních produktů, které se mají odborně přezkoumávat, stanoven následovně: (a) specifikace požadavků zákazníka, (b) softwarová architektura; (c) detailní návrh; (d) zdrojový kód; (e) plán sw projektu; (f) plán testování; (g) plán konfiguračního řízení.

Upozornění k interpretaci:

(a) Věci spojené s procesem odborných přezkoumáním a stanovením jeho náležitostí samozřejmě musí být někde zaznamenány, tj. naplánovány. Může to být součást plánu validace a verifikace (tak to má standard ESA PSS-05-0, Software Validation and Verification Plan), může to být součástí plánu zajištění (jakosti) (tak to má standard NASA-STD-2100-91, NASA-DID-M400 Assurance Plan), může to být součástí samotného plánu vývoje softwaru (tak to má standard MIL-STD-498) atd. Pro konečné stanovení této záležitosti má poslední slovo činnost vedení projektu; viz upozornění k interpretaci (c) u konkrétních nároků na testování (praktika č. 4).

(b) Je vhodné stanovit více postupů/ typů přezkoumání, které se použijí podle toho, co se přezkoumává a jaká je situace. Vysvětlení: je vhodné disponovat přísnějším postupem pro nejkritičtější věci a situace a méně přísným postupem, který je méně časově náročnější.

(c) Uvedený minimální seznam pracovních produktů pro začátek je pochopitelně možné časem rozšiřovat anebo po dostatečných zkušenostech a pečlivé analýze modifikovat. Na druhou stranu nikdy by nemělo být rezignováno na kvalitní odborné přezkoumání specifikace požadavků zákazníka, softwarové architektury a plánu sw projektu.

(d) Inspirací pro „ideální“ seznam, co přezkoumávat (tj. jaké pracovní produkty) a vzhledem k čemu, poskytuje, přesněji řečeno nárokuje, standard MIL-STD-498 v příloze D v tabulce č. 6 Software products and associated evaluation criteria.

Poznámky:

(a) Standard MIL-STD-498 se úplně vyhýbá diskusím na téma inspekce, review atd. Jednoduše má koncept evaluace, což je definováno jako zjištění, zda produkt či aktivita splňuje nároky na ní kladené.

2.7 Zajištění naplánovaných a předepsaných věcí

Praktika číslo 7: Zajištění naplánovaných a předepsaných věcí
Zaměřeno na:

1. Provádění činnosti, která zajišťuje naplánované a předepsané věci
2. Systematické plánování a provádění zajištění naplánovaných a předepsaných věcí
3. Nezávislé provádění

Pro začátek chceme docílit:

1. Provádění činnosti, která zajišťuje naplánované a předepsané věci: Je třeba docílit, aby byl prováděn proces (činnost), třeba byť základním a velmi jednoduchým způsobem, zajišťující, že naplánované a předepsané činnosti se příslušně provádí. A dále, že naplánované a předepsané pracovní produkty vznikají a podstupují naplánované a předepsané kontroly (t.j. zjednodušeně řečeno testování anebo odborná přezkoumání).
2. Systematické plánování a provádění zajištění naplánovaných a předepsaných věcí: Je třeba docílit, aby bylo včas a systematicky plánováno a systematicky prováděno zajištění naplánovaných a předepsaných věcí. A toto se v každém případě týká zásadních činností (aktivit/ procesů) a zásadních pracovních produktů, které jsou kritické pro úspěch projektu a další život vyvíjeného softwarového produktu.
3. Nezávislé provádění: Je třeba docílit, aby činnost zajištění naplánovaných a předepsaných věcí byla prováděna nezávisle. Tzn. osobou/ osobami, které nezodpovídají za dané činnosti a produkty.

Konkrétní nároky k počátečním cílům:

1. Činnost zajištění naplánovaných a předepsaných věcí (dále už jen zajištění jakosti) je třeba chápat a provádět (včetně zajištění nápravy zjištěných nedostatků) tak, jak je velmi stručně ve svém „štíhlejším“ chápání popsána v sekci 5.2.8 Software Quality Assurance v NASA příručce Software Management Guidebook, NASA-GB-001-96; principielně to samé je požadováno v sekci 5.16 v standardu MIL-STD-498. Dále minimálně ten, kdo zajištění jakosti plánuje a je za něj celkově zodpovědný, musí znát a vhodně využívat kapitolu č. 5 Software Quality Assurance v části II ve standardu ESA PSS-05-0 a NASA příručku Software Assurance Guidebook, NASA-GB-A201. V případě, že se činnost přezkoumání používá jako prostředek pro činnost zajištění jakosti, pak je samozřejmě třeba respektovat nároky pro odborná přezkoumání (viz podstatná praktika a konkrétní nároky č. 6) (pozn. pro potřeby činnosti zajištění jakosti stačí velmi jednoduchá přezkoumání, ale musí být poctivá a musí mít k dispozici všechny potřebné podklady, které vznikají v průběhu projektu).
2. Činnost zajištění naplánovaných a předepsaných věcí, tedy zajištění jakosti je třeba včas a vhodně plánovat. *Co* se má *kdy* plánovat – mluvíme o procesu zajištění jakosti – je stručně popsáno v sekci 5.4 Evolution of the SQAP Throughout the Life Cycle, část II ve standardu ESA PSS-05-0. Principielně to samé je popsáno na příslušných místech v NASA příručce Software Assurance Guidebook, NASA-GB-A201. Dále činnost zajištění jakosti musí být stanovena a naplánována v plánu zajištění jakosti (SQA Plan, Assurance Plan atd.). Plán zajištění jakosti musí svým obsahem vycházet ze vzorů pro tento typ pracovního produktu ve standardu ESA PSS-05-0 (konkrétně Software Quality Assurance Plan) a standardu NASA-STD-2100-91 (konkrétně NASA-DID-M400 Assurance Plan). Tyto dva standardy lze též použít pro inspiraci co se týká konkrétního formátu a struktury. Úzkému chápání činnosti zajištění jakosti by odpovídalo zajišťovat činnosti a produkty nárokované tímto textem (konkrétně v kapitolách č. 1 a 2) plus co bude navíc obsaženo v konkrétním plánu sw projektu. Pro začátek toho může být dost. Proto pro začátek je minimální seznam činností a pracovních produktů, které by měly být zajišťovány následující: (a) specifikace požadavků zákazníka – činnost a souvisící produkt, (b) softwarová architektura – činnost a souvisící produkt, (c) detailní návrh – činnost a souvisící produkt, (d) zdrojový kód – činnost a souvisící produkt, (e) vedení projektu – činnost a další souvisící produkty (zejména plán sw projektu), (f) testování – činnost a souvisící produkt (tj. plán testování včetně záznamů o testování), (g) konfigurační řízení – činnost a souvisící produkty (hlavně plán konfiguračního řízení), (h) odborná přezkoumání – činnost a souvisící produkty (hlavně plány pro odborná přezkoumání a záznamy z vykonávání odborných přezkoumání). Pozn. seznam je tedy stejný jako u odborných přezkoumání s tím, že byl nutně rozšířen právě o činnost odborná přezkoumání a s tím související produkty.
3. Činnost zajištění naplánovaných a předepsaných věcí, tedy zajištění jakosti je třeba provádět nezávisle. To znamená, že osoby odpovědné za zjišťování, zda se provádí (činnosti), to co se naplánovalo a co je předepsáno, a zjišťující zda vznikne (pracovní produkty) to, co se naplánovalo a co je předepsáno a zda pracovní produkty podstoupí ty kontroly (testování anebo odborná přezkoumání), které se naplánovaly a co

jsou předepsány, že tyto osoby nejsou ty, co pracovní produkty vytvořily, nejsou ty, co prováděly činnosti a nejsou ty, co za to odpovídají. Toto nevylučuje účast takových osob na tomto zjišťování.

Upozornění k interpretaci:

(a) Tento konkrétní nárok v základní a jednoduché podobě nárokuje to, čemu se běžně říká SQA (Software Quality Assurance), tede česky zajištění jakosti. Dále viz celý odstavec nadepsaný upozornění v deklaraci této základní praktiky č. 7, tj. viz sekce č. 1.7.

(b) Plán zajištění jakosti, který je zde nárokován může mít obecně různý formát a strukturu a obecně může být v různém fyzickém vztahu vzhledem k plánu softwarového projektu (logicky je jeho součástí stejně jako všechny dílčí plány). Dále platí vše, co bylo v této souvislosti řečeno v upozornění k interpretaci (c) u konkrétních nároků na testování (praktika č. 4).

(c) Uvedený minimální seznam činností a pracovních produktů pro začátek je pochopitelně možné časem rozšiřovat nebo po dostatečných zkušenostech a pečlivé analýze modifikovat. Na druhou stranu nikdy by nemělo být rezignováno na dostatečné zajištění specifikace požadavků zákazníka – činnost a souvisící produkt, softwarové architektury – činnost a souvisící produkt, vedení projektu – činnost a souvisící produkty, odborná přezkoumání – činnost a souvisící produkty.

(d) Požadovaná nezávislost činnosti zajišťování jakosti je chápána, tak jak je popsána v sekci 5.16.3 v standardu MIL-STD-498.

2.8 Vedení softwarového projektu

Praktika číslo 8: Vedení softwarového projektu
Zaměřeno na:

1. Tvorba, používání a údržba plánu projektu
2. Předpisy/ vzory pro zásadní pracovní produkty
3. Nezúžený záběr plánování
4. „Základní“ metoda vedení sw projektu
5. Postupné a průběžné plánování
6. Vhodný a správný celkový postup vývoje softwaru, který zahrnuje vhodný(é) model(y) životního cyklu
7. Ošetření rizik
8. Odborné znalosti vedoucího projektu

Pro začátek chceme docílit:

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[1], *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 Metzger[1] 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 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 Nesim[2] vypadá následovně (použitá notace: [fáze/ proces] **à** (hlavní produkt fáze/ procesu)): [Získání požadavků a předběžná (high-level) analýza] **à** (Profil projektu) **à** [Identifikace subsystémů] **à** (Předběžné subsystémy) **à** [Analýza subsystémů] **à** (První obecný soubor tříd) **à** [Předběžná analýza znovupoužitelnosti] **à** (Pokyny pro znovupoužívání) **à** [Výběr nástrojů] **à** (Nástroje) **à** [Hrubé plánování projektu] **à** (Vazby mezi úkoly) **à** [Analýza zdrojů potřebných pro subsystémy] **à** (Náklady na subsystémy) **à** [Plánování projektu] **à** (Plánování) **à** [Hodnocení/ výběr zdrojů] **à** (Alokace zdrojů) **à** (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 zásada[3]: 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. 2 a 3 z těchto cílů pro začátek. Tématu údržby a tomu *co* a *kdy* má být stanoveno se také věnují body č. 4 a 5 z těchto cílů pro začátek. Tématu celkového postupu vývoje, zjednodušeně model životního cyklu, se také věnuje bod č. 6 z těchto cílů pro začátek.

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

[1] 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,)
[2] Jedná se o článek „Managing OO Projects Better" v *IEEE Software*, July/August 1998.
[3] 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".

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-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.)

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. 1 výše a nárokování standardních kvalitních vzorů/ předpisů pro zásadní pracovní produkty (typu sw architektura) v bodě č. 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;[4] 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.[5] Tento text může posloužit pro pomoc při definování softwarového procesu, který bude splňovat elementární nároky na dobrou praxi softwarového inženýrství. CMM pro úroveň 3 požaduje existenci standardního softwarového procesu organizace.)
4. „Základní" metoda vedení sw projektu:

[4] 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.)

[5] 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.)

- I) Je třeba docílit, aby ty činnosti, které bezprostředně a prakticky umožňují provedení, resp. vykonání[6] všeho, co se má udělat a tím realizaci projektu, byly dostatečně, vhodně a rutinně 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. (viz též sekce upozornění pro interpretaci níže). 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) 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 udržovat plán projektu; principielní důvody pro nutnost postupného a průběžného plánování a tím i údržby plánu projektu byly uvedeny výše v bodě č. 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.

[6] 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 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.

- 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.

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. 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. 1 a 4 výše). Principielní důvody, proč je třeba provádět postupné plánování, byly uvedeny v bodě č. 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[7] „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).
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. (Pozn. „korektní“ definice některých termínů viz sekce Upozornění k interpretaci.) 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

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

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.[8] Principielně o to samé, ale trochu jinak podávané, jde v diskusích[9] 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 vývoje[10] (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. 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 zmařit[11] 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 nároky 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 projektu[12]

[8] 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 (viz zdroje).

[9] 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, viz zdroje); „A Rational Development Process" v *CrossTalk*, July 1996 (taky jako Rational white paper, viz zdroje); článek „Using Win Win Spiral Model: A Case Study" od Boehma v *Computer*, July 1998 atd.

[10] Případ tohoto typu je např. popisován v článku „MIL-STD-498: What's New and Some Real Lessons Learned" v *CrossTalk*, March 1996 (viz zdroje).

[11] 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* (viz zdroje).

[12] 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, viz zdroje); „A New Process for Acquiring Software Architecture" ve stejné příručce jako minulý text,

a vyžaduje vzhledem k tomuto příslušným způsobem postupovat). Boehmovy nároky přijali v posledních verzích Rational Unified Process (viz zdroje).

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

Konkrétní nároky k počátečním cílům:

1. Plán softwarového projektu musí být vytvořen podle předem připraveného vzoru pro tento typ pracovního produktu. Použitý vzor plánu softwarového projektu pochopitelně může a má zohledňovat kontext a typ daného projektu (v prostředí daného týmu, organizace atd. může být připraveno několik vzorů vhodných pro různé typy projektů). Je třeba, aby vzor plánu projektu vycházel a svým logickým obsahem tedy záběrem odpovídal standardu NASA Software Documentation Standard, NASA-STD-2100-91 nebo předpisům pro plán projektu v standardu MIL-STD-498 (tj. Software Development Plan DID, Software Test Plan DID, Software Installation Plan DID, Software Transition Plan DID). (Pozn. Software Engineering Process Office dává k dispozici Software Development Plan (SDP) Template na základě příslušného předpisu v MIL-STD-498.) Pro konkrétní strukturování a fyzické členění je třeba se dále inspirovat vzory tohoto pracovního produktu ve standardu ESA PSS-05-0 (tj. Software Project Management Plan, Software Configuration Management Plan, Software Verification and Validation Plan, Software Quality Assurance Plan) a v NASA příručkách Manager's Handbook for Software Development a Recommended Approach to Software Development (tj. Software Development/ Management Plan, Test Plans, resp. Generalized Test Plan Format and Contents). Výsledný plán softwarového projektu může mít různé fyzické členění, tj. určité dílčí plány mohou být fyzicky samostatné dokumenty, např. plán konfiguračního řízení, plán testování, plán zajišťování jakosti atd. Fyzické členění ať zcela vyhovuje daným podmínkám. Je třeba, aby celkové koncipování dokumentace softwarového projektu (tedy plán sw projektu spolu s dokumentací důležitých pracovních produktů – nemusí mít nutně vždy formu klasického dokumentu, např. sw architektura může být v nástroji typu CASE) tvořilo konzistentní celek. Možnosti celkového koncipování dokumentace výborně ukazují standard NASA Software Documentation Standard (NASA-STD-2100-91), standard ESA PSS-05-0 (vzory plánů, produktů a formulářů jsou v přílohách B, C a E), předpisy pracovních produktů v standardu MIL-STD-498 (tj. Data Item Descriptions, tyto předpisy předepisují, co je principielní, nejsou to a ani nemají být vzory konvenčních dokumentů, ale tyto lze na základě nich připravit) a do značné míry vzory plánů, sw produktů, formulářů, checklistů atd. obsažené v NASA příručkách Manager's Handbook for Software Development (SEL-84-101) a Recommended Approach to Software Development (SEL-81-305). Je třeba plán projektu udržovat tak, jak je koncepčně popsáno v sekci 7.1.4 Maintaining the Software Plan v NASA příručce Software Management Guidebook (NASA-GB-001-96) (podrobněji viz konkrétní nárok č. 5 dále).
2. Je třeba, aby součástí stanovení technického aspektu činností, které se mají v průběhu projektu vykonávat, byly v plánu softwarového projektu obsaženy předpisy, resp. vzory pro zásadní pracovní produkty (o předpisu má smysl mluvit pokud forma pracovního produktu není konvenční dokument, o vzoru pokud ano). Předpisy, resp. vzory musí vycházet z předpisů, resp. vzorů uvedených v NASA Software Documentation Standard (NASA-STD-2100-91) v standardu MIL-STD-498, v standardu ESA PSS-05-0, v NASA příručkách Manager's Handbook for Software Development a Recommended Approach to Software Development atd. Seznam vzorů a předpisů k dispozici viz sekce č. 3.12. Kromě plánů je třeba mít předpis, resp. vzor minimálně pro následující pracovní produkty: specifikace požadavků zákazníka, sw architektura, detailní návrh, zdrojový kód. Dále pro zaznamenání údajů z testování, odborných přezkoumáních, CM, SQA, atd. Je třeba respektovat to, co bylo vzhledem k pracovním produktům, resp. předpisům požadováno konkrétními nároky (č. 1 až 7) výše.
3. Je třeba, aby činnost vedení sw projektu a tím i plánování měly dostatečně široký záběr, aby nebyly chápány úzce. Proto je třeba, aby činnost plánování (nedělitelná součást činnosti vedení sw projektu) měla takový záběr – tedy zabývala se těmi věcmi – který je popsán v NASA příručce Software Management Guidebook,

---

Addendum G; *On the Definition of Software System Architecture*, technická zpráva Center for Software Engineering (viz zdroje); *Architecture-Based Development*, SEI-99-TR-007 (viz zdroje).

NASA-GB-001-96. Dále je třeba, aby činnosti plánování a vedení sw projektu zajistily nároky popsané v tomto textu. Následuje upozornění na některé činnosti, které jsou pro zdar celého projektu velmi důležité a nebyly samostatně probírány v tomto textu: (i) Uvedení do provozu: Vše co souvisí s uvedením sw produktu do provozu, tj. instalace, přechod ze starého systému atd. Stručný a schematický popis činnosti je např. k dispozici v sekci 5.2.5 Preparing for Software Delivery v NASA příručce Software Management Guidebook a v kapitole The Transfer Phase v standardu ESA PSS-05-0. Vzory a předpisy příslušných pracovních produktů jsou k dispozici např. v NASA Software Documentation Standard (konkrétně Delivery and Operational Transition Plan NASA-DID-M700, Version Description NASA-DID-P500) ve standardu MIL-STD-498 (konkrétně Software Installation Plan SIP DID, Software Transition Plan STrP DID, Software Product Specification SPS DID, Software Version Description SVD DID) atd.

4. „Základní" metodu vedení je třeba minimálně chápat a provádět tak, jak je popsána v sekci č. 1.3 Concepts and Theory ve zprávě Report on Project Management and Software Cost Estimation Technologies (STSC-TR-012-Apr95). Dále „základní" metoda vedení musí být prováděna tak, aby celkový proces vedení odpovídal popisu v kapitole č. 2 Software Project Management ve standardu ESA PSS-05-0 (v části 2) a v kapitole č. 6 Finishing the Software Plan – Defining the Management Approach a kapitole č. 7 Running the Project v NASA příručce Software Management Guidebook (NASA-GB-001-96). „Základní" metoda vedení musí respektovat, lépe řečeno naplňovat, zvolený a popsaný model(y) životního cyklu pro projekt; v této souvislosti se musí dodržovat principy doporučení uvedené v příloze H Guidance on Ordering Deliverables a v sekci G.6.4 Scheduling the selected activities in each build (v příloze G) ve standardu MIL-STD-498. Je třeba, aby plánování pro potřeby „základní" metody vedení projektu, tedy iniciální tvorba a poté po celou dobu trvání projektu údržba (což může být zdetailnění, zpřesnění, oprava, změna atd.) WBS, sítě aktivit, odhadů a časového plánu vhodně odpovídalo praxi, kterou příručka The Program Manager's Guide to Software Acquisition Best Practices nárokuje v: (i) Principal Best Practices (konkrétně jde o praktiky č. 4 Metrics-based Scheduling and Management č. 5 Binary Quality Gates at the Inch-Pebble Level); (ii) Best Practices (konkrétně jde o praktiku Activity Planning v oblasti Planning); (iii) Breathalyzer Test (konkrétně jde o sekce č. 1 Do you have a current, credible activity network supported by a Work Breakdown Structure (WBS)? č. 2 Do you have a current, credible schedule and budget? č. 5 Do you know your schedule compression percentage? č. 6 What is the estimated size of your software deliverable? How was it derived?). Je třeba, aby sledování postupu a stavu projektu vhodně odpovídalo praxi, kterou příručka The Program Manager's Guide to Software Acquisition Best Practices nárokuje v: (i) Principal Best Practices (konkrétně jde o praktiku č. 6 Program-wide Visibility of Progress vs. Plan); (ii) Best Practices (konkrétně jde o celou oblast Program Visibility). Za účelem co konkrétně je vhodné sledovat, v konečném důsledku měřit, pro činnost vedení sw projektu je třeba se inspirovat v Candidate Management Indicators (Appendix F ve standardu MIL-STD-498), v Metrics and Key Management Aids (sekce 6 v NASA příručce Manager's Handbook for Software Development), v Project Control Panel (kapitola 2 v příručce The Program Manager's Guide to Software Acquisition Best Practices), v článku Defining and Understanding Software Measurement Data (SEI Featured Article), v tabulce 6-4 Required Activities and Related Measures (v NASA příručce Software Management Guidebook). Používaná metoda, jak získat konkrétní metriky z koncepčního záměru, je popsána v textu The Goal Question Metric Approach. Stručný popis, jak dosáhnout rozumného programu měření, je např. v textu Developing a Successful Metrics Program (anebo Developing an Effective Metrics Program). Je třeba, aby osoba odpovědná za měření, tj. jeho vymyšlení a zavedení, znala některý z textů popisujících tuto problematiku podrobněji, jako např. Practical Software Measurement: A Foundation for Objective Project Management. K hodnocení stavu projektu je dále třeba vhodně využívat k tomuto specielně vyvinuté „testy", resp. sady otázek. K dispozici jsou Breathalyzer Test (kapitola 2 v příručce The Program Manager's Guide to Software Acquisition Best Practices), Questions to Determine Project Status/ Health (sekce 1 v Software Management for Executives Guidebook). Jako úplný základ k sledování stavu projektu z pohledu Kolik toho už je hotovo? Kolik to stálo? Jak dlouho to trvalo? Jak to vypadá se vztahem k původním časovým plánům a rozpočtu? atp. je třeba používat mocný a jednoduchý přístup Earned Value. Tento přístup spojuje činnosti plánování, sledování a vyhodnocování. Stručně a jasně je popsán v článcích Earned Value Project Management: an Introduction (CrossTalk, July 1999), Gaining Confidence in Using Return on Investment and Earned Value (CrossTalk, April 1999), Earned Value Project Management: A Power Tool for Software Projects (CrossTalk, July 1998), Checkpoint Restart – Part 1 a Part 2 (SEI Featured Articles). Pozn. praxe popsaná v příručce The Program Manager's Guide to Software Acquisition Best Practices zajišťuje použití přístupu Earned Value (od toho, že Project Control Panel obsahuje příslušná měření, přes definování jasných kriterií hotovo/nehotové u detailních úloh ve WBS atd.). Výsledkem vyhodnocení zjištěných údajů a informací o postupu a stavu projektu můžou být nápravné akce, které se projeví vhodnou údržbou plánů, která může mít mnoho podob. Stručný popis nápravných akcí lze nalézt v Metrics and Key Management Aids (sekce 6 v NASA příručce Manager's Handbook for Software Development), v Troubleshooting and Problem Avoidance (sekce 2 v Software Management for Executives Guidebook). Dále, vodítkem pro to jak mají zhruba vypadat hodnoty získané

sledováním mohou být Quantitative Targets (kapitola 4 v příručce The Program Manager's Guide to Software Acquisition Best Practices).

5. Při vedení sw projektu je třeba plánování provádět průběžně a postupně. Celkově je třeba dodržovat schematicky řečené nároky v sekci 7.1.4 Maintaining the Software Plan, NASA příručka Software Management Guidebook (NASA-GB-001-96). Je třeba, aby přístup k postupnému upřesňování, dopracovávání a průběžné údržbě plánu softwarového projektu koncepčně vycházel z přístupu, který je popsán ve standardu ESA PSS-05-0 (v sekcích 2.4, 3.4, 4.4, 5.4 v části I standardu ESA je mimo jiné schematicky popsáno co má obsahovat plán sw projektu po „fázích" požadavky uživatele, požadavky na software, architektonický návrh, detailní návrh a produkce; v sekcích 2.4, 3.4, 4.4, 5,4 v části II standardu ESA je schematicky popsán vývoj plánu sw projektu skrze životní cyklus; pozn. termín „fáze" je třeba chápat s vědomím příslušného vysvětlení v poznámce (f) v sekci níže věnované upozorněním pro interpretaci). NASA příručka Recommended Approach to Software Development SEL-81-305 také obsahuje stručné popisy co má tým vedení projektu zajišťovat v závislosti na stavu rozpracovanosti vyvíjeného sw produktu. Dále je třeba respektovat případné další nároky *co* se má *kdy* plánovat, které byly popsány pro činnosti testování, konfigurační řízení, odborná přezkoumání a zajištění naplánovaných a předepsaných věcí výše (jde o konkrétní nároky pro podstatné praktiky č. 4, 5, 6 a 7).
6. V rámci plánování sw projektu je třeba včas zvolit a popsat model životního cyklu, který bude pro „základní" metodu vedení projektu koncepčním návodem jak postupovat. (Pozn. „včas" znamená jednak v době iniciálního plánování a v případě, že ve skutečnosti je potřeba více jednotlivých modelů životního cyklu, tak „včas" znamená ve chvíli dostatku informací pro rozhodnutí, nejpozději však předtím než je daný model životního cyklu potřeba.) Základní popis problematiky modelů životního cyklu (tj. modelů postupu vývoje) je třeba znát minimálně ze sekce 5.1 Selecting an Appropriate Life-Cycle Model v NASA příručce Software Management Guidebook NASA-GB-001-96 (obsahu též model životního cyklu pro údržbu), z přílohy G Guidance on Program Strategies, Tailoring, and Build Planning ve standardu MIL-STD-498 (pozn. termínem Program Strategy je označován celkový model životního cyklu na nejhrubší úrovni dekompozice, kde jednotlivé dekompoziční jednotky jsou části nadřazeného systému realizované softwarem). Jako základní a dobrý úvod do tématu, nač a proč je vůbec dobrý nějaký model životního cyklu, velmi doporučuji poznámky k přednášce The Software Process z kurzu Software Design (jedná se o kurz SYDE 221 Software Design na University of Waterloo v Kanadě). Dále je potřeba, aby osoby, které vybírají a definují modely životního cyklu, znaly minimálně následující texty týkající se principů této problematiky: Models of Software Evolution: Life Cycle and Process (Curriculum Module SEI-CM-10-1.0), článek Improving Software Economics in the Aerospace and Defense Industry, technickou zprávu On the Definition of Software System Architecture (USC/CSE-95-TR-500), technickou zprávu Anchoring the Software Process (USC/CSE-95-TR-507). Zvolený model životního cyklu je třeba vhodně použít jako koncepční návod pro časové plánování a „základní" metodu vedení projektu. Je třeba zabránit, aby rigidní či nevhodné časové plánování zmařilo záměry, pro které byl daný model životního cyklu vybrán. Standard MIL-STD-498 toto nárokuje v příloze H Guidance on Ordering Deliverables a v sekci G.6.4 Scheduling the selected activities in each build (v příloze G). Je třeba, aby celkový postup vývoje softwaru odpovídal nárokům, které jsou uvedené a vysvětlené v technické zprávě Anchoring the Software Process (USC/CSE-95-TR-507). Ukázka sladění principielních nároků stanovených v textu Anchoring the Software Process a flexibility umožněné vlastnostmi OO přístupu je vidět v textu Rational Unified Process (white paper fy Rational k poslední verzi jejího procesu vývoje). Tento text velmi doporučuji znát, v případě OO vývoje je nutno jej znát.
7. V rámci vedení sw projektu je třeba provádět činnost, která ošetřuje potenciální rizika (Risk Management). Tj. činnost, která identifikuje, analyzuje, předchází, monitoruje atd. možná rizika. Je třeba, aby tato činnost byla prováděna v tom smyslu jak je maximálně stručně popsáno v sekci 2.2.3 Risk Management v části II ve standardu ESA PSS-05-0 a v článku Continuous Risk Management at NASA (ze SATC, NASA). Ten, kdo činnost řízení rizik plánuje a je za ní odpovědný, by měl znát podrobnější popis této problematiky, např. Software Risk Management, Technical Report SEI-96-TR-012.
8. Je třeba, aby osoby, které výkonně vedou projekt vývoje softwaru, tj. jsou zodpovědné za jeho plánování, provádění a řízení, měly dostatečné odborné znalosti. Doporučené souhrnné materiály jako minimum pro začátek: Software Management Guidebook (NASA-GB-001-96), ESA Software Engineering Standards (ESA PSS-05-0), The Program Manager's Guide to Software Acquisition Best Practices, Manger's Handbook for Software Development (NASA, SEL-84-101, Rev. 1), Software Project Management (Curriculum Module SEI-CM-21-1.0). Dále k tématu vedení projektu obecně doporučuji text A Guide to the Project Management Body of Knowledge (Project Management Institute). Velmi doporučuji průběžně sledovat časopis CrossTalk (The Journal of Defense Software Engineering). CrossTalk vychází jednou za měsíc, je k dispozici elektronicky a zabývá se tématy bezprostředně souvisícími s vedením projektů. Dále, každý by se měl vzdělávat systematicky a dlouhodobě. K tomu může výborně posloužit studijní plán fy Construx pro samostudium a samozřejmě odborné zdroje uvedené v tomto textu. (Pozn. tento bod se týkal

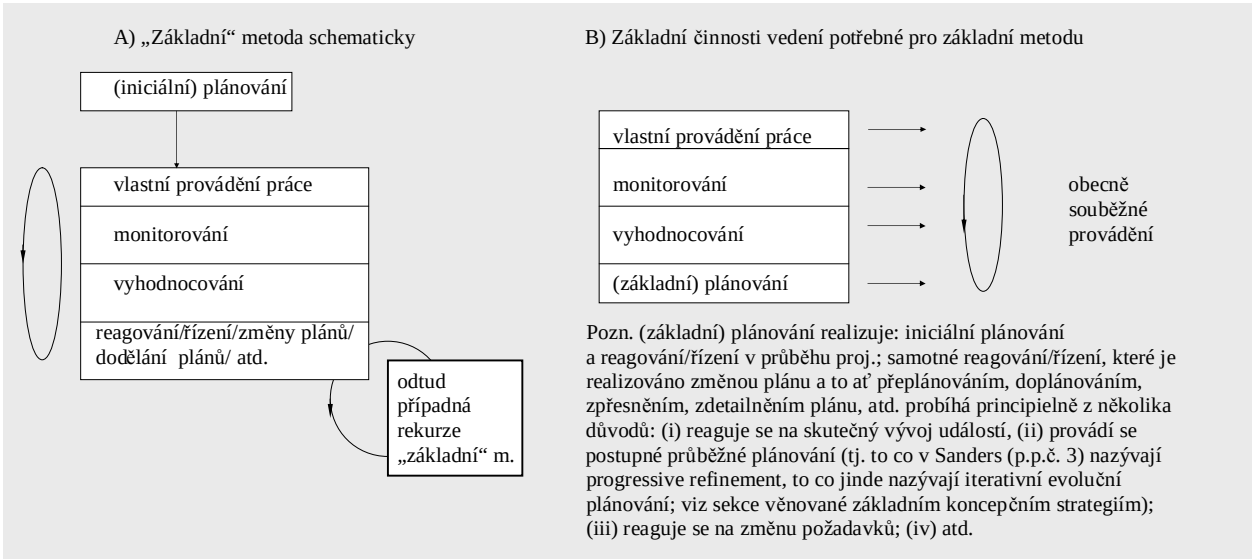
konkrétních nároků na odborné znalosti pouze vzhledem k vedení sw projektu, nikoli vzhledem k celé disciplině softwarové inženýrství.)

Upozornění pro interpretaci:

(a) Schematické znázornění „základní“ metody vedení a základních činností vedení je na následujícím obrázku 2.8-1. Dále viz bod b níže.

(b) Termín „základní“ metoda nebo elementární metoda vedení sw projektů, nebo jakkoli podobně jinak byl pro tento text záměrně zaveden (pozn. není to tedy termín typu modul, analýza, konfigurační řízení, odhadování, přezkoumání atd., tedy termín „základní metoda vedení sw projektu“ jako takový se v odborné literatuře na dané téma nevyskytuje a pokud ano, pak to nemá vědomou souvislost s tímto textem). Důležité je proč byl zaveden a co označuje. Zaveden byl v zásadě z organizačních důvodů, z důvodů co nejvhodněji tento text členit. Co označuje? Označuje následující skutečnosti, které z principu věci musí splňovat jakkoli prováděné vedení sw projektu – jakýkoli proces vedení projektu:

- je třeba získat/ získávat, zjistit atd. příslušné požadavky, omezující podmínky atd. všech kategorií na produkt a projekt;
- přitom je třeba provádět iniciální plánování a vytvořit plán projektu;
- je třeba provádět vlastní práci;
- postup projektu je třeba sledovat;
- postup projektu je třeba vyhodnocovat;
- na postup projektu je třeba reagovat příslušným a vhodným přeplánováním, doplánováním, atd.



Obrázek 2.8-1 Základní činnosti vedení a „základní“ metoda vedení

Takto pojatou „základní“ metodu – její konkrétní zápis, členění, včlenění, pojetí, interpretace, míra rozvedení do hloubky/ šířky atd. mohou být relativně velmi různorodé – lze skoro jistě najít v každém uceleném textu, který se zabývá vedením sw projektu, ať už jako norma, předpis, učebnice, přehled atd. Texty, ve kterých lze „základní“ metodu najít nebo vysledovat a to v nejrůznějších rolích a souvislostech jsou například: mezinárodní norma ISO 12207, technická zpráva STSC o vedení projektů, příručka NASA pro vedení projektů, nejlepší praktiky pro akvizici softwaru od SPMN, Capability Maturity Model atd. Přehled schematických zápisů několika variant základní metody obsahující jen relativně abstraktní činnosti vhodný pro účely této sekce je v tabulce 2.8-1.

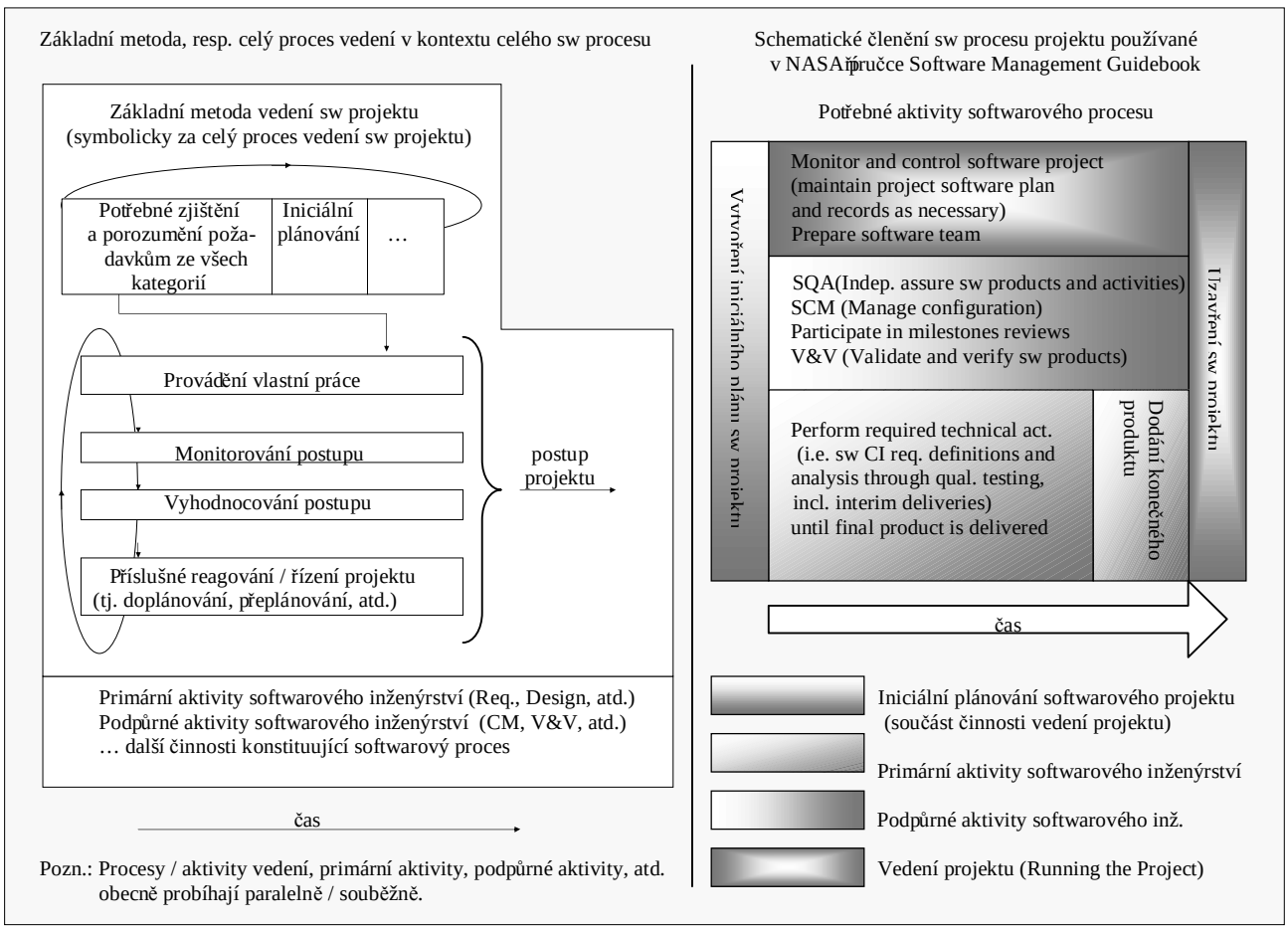
1. NASA Software Management Guide	**2. STSC Report on Project Management**
Step 1: Porozumění obsahu práce (začátek plánování projektu) Step 2: Definování technického přístupu Výběr vhodného modelu životního cyklu Výběr vhodných aktivit, metod a produktů Step 3: Definování přístupu k vedení (dokončení plánu projektu) Organizace, Odhadování, Časové plánování, … Step 4: Provádění projektu (vykonání sw plánu projektu) Monitorování, Řízení (Control), Údržba sw plánu, … Step 5: Uzavření projektu	… 1. Organizace projektu a vedení … 2. Tvorba plánu projektu 2.1 Definování požadavků na projekt 2.2 Tvorba časového plánu projektu (the Project Schedule) 2.3 Přiřazení zdrojů projektu … 3. Realizace plánu projektu … 3.3 Měření a zaznamenávání postupu 3.4 Provádění úprav plánu projektu … 4. Hlášení informací o projektu

Pozn.: tzv. The Five-Step Project Process; role: strukturuje celý text.	Pozn.: Concepts and Theory; role: sekce 1.3.

3. Software Acquisition Best Practices (SPMN)
 	
Program Control (Sw Management Program Control) vyžaduje:
- Plánování (Sw Management Planning) zahrnuje:
  - Definování cílů na produkt
  - Strukturování projektu
  - Přezkoumání plánu
  - Časové plánování (scheduling) projektu
  - Testování plánu
  - Vyčíslení nákladů (Costing the plan)
  - Časté revidování plánu podle změn okolností projektu
- Provádění aktuálního plánu
- Zahrnování změn do plánu a projektu
- Dosahování cílů týkajících se produktu, zdrojů, kvality
- Koordinování úsilí skupin a jednotlivců
- Zajišťování adekvátních kontrol kvality k podpoření vysoké kvality softwaru
 	
Pozn.: souhrn oblastí Program Control a Planning, do kterých jsou sdružovány příslušné „best“ practices.
	4. ISO 12207
	
	… 1. Iniciace a definice rozsahu
	… 2. Plánování
	  …
	  a) Časové plány
	  b) Odhady
	  c) Adekvátní zdroje
	  d) Alokace úkolů
	  …
	… 3. Provádění a řízení/vedení (Execution and Control)
	  …
	  3.2 Monitorování …
	  3.3 Prozkoumání, analyzování, vyřešení … změny plánů …
	  3.4 Hlášení postupu …
	… 4. Přezkoumání a zhodnocení
	… 5. Uzavření
	
	Pozn.: sekce 7.1; role: předepisující *generický* proces vedení (Management Process), který má být vhodně *instancován* pro konkrétní potřebu; např. pro proces vývoje, pro proces dodání, pro proces údržby, atd.

Tabulka 2.8-1 Schematické zápisy některých variant „základní“ metody, jak ji lze najít/vysledovat v literatuře

Na obrázku 2.8-2 je „základní“ metoda, jak byla charakterizována výčtem na začátku této sekce, znázorněna spolu s velmi schematickým, ale v principu pravdivým vztahem ke zbytku sw procesu projektu; v tomto vztahu „základní“ metoda symbolicky reprezentuje celý proces/činnost vedení sw projektu (software project management). Na obrázku je dále pro srovnání uveden obrázek, který je v NASA příručce Software Management Guidebook používán pro schematické znázornění softwarového procesu projektu.



Obrázek 2.8-2 „Základní“ metoda (+ schematické/symbolické znázornění kontextu)

Každý text, zdroj má více či méně jiný záměr, akcent, přístup, členění atd. k uchopení problematiky. Dále proces vedení sw projektu (symbolicky zastoupen základní metodou vedení sw projektu) z levé části obrázku 2.8-2 koresponduje s aktivitami plánování sw projektu + vedení sw projektu z pravé části obrázku. NASA příručka pro vedení projektu (Software Management Guidebook) představuje moderně koncipovaný text, který nepředjímá model SDLC, který zahrnuje moderní koncepty obsažené ve standardech MIL-STD-498 a ISO 12207, který předjímá průběžné postupné plánování jak jej zná standard ESA PSS-05-0 atd. V této NASA příručce lze základní metodu identifikovat ve struktuře celého textu (krok č. 1 – kapitola 4, krok č. 2 – kapitola 5 atd.; vysvětlení pro kroky č. 1 až 5 viz tabulka 2.8-1). Nicméně tato NASA příručka klade hlavní důraz na popis, co vše (jaké kategorie) se musí nejdříve zjistit za nejrůznější požadavky a omezení na projekt a poté, co vše se musí určit, stanovit atd., tj. určit/ definovat model životního cyklu, tj. naplánovat pro primární aktivity sw inženýrství (požadavky, analýza, architektura, detailní návrh, implementace, testování atd.), pro podpůrné aktivity SE (V&V, SQA, CM atd.), pro aktivity vedení sw projektu (WBS, odhady, harmonogram, měření pro vedení, monitorování, údržbu plánu sw projektu atd.). Samotnou bezprostřední realizací vedení sw projektu, tj. jak se zkonstruuje WBS, se moc nezabývá (nárokuje ať je), jak se odhaduje se moc nezabývá (nárokuje ať se odhaduje), jak se plán projektu postupně dodělává a předělává jak projekt žije se moc nezabývá (nárokuje ať se tak děje) atd. Z tohoto důvodu je NASA příručka Software Management Guidebook velmi vhodná pro účel zajištění nezúženého plánování. Naproti tomu co se týká samotné realizace základní metody vedení sw projektu je třeba tuto příručku určitě doplnit. STSC technická zpráva Report on Project Management zato obsahuje relativně dost podrobný popis týkající se WBS, tvorby harmonogramu, odhadování, jejich vztahů atd. Je psána na velmi nízké (nebo z určitého pohledu vysoké, záleží jak se to vezme) úrovni abstrakce, tj. to co se má udělat/ zabezpečit je „práce, která se má vykonat“ (tedy skoro celý námět, kterým se zabývá SMG/NASA), neuvažuje o existenci modelu SDLC atd. Toto základní, nízkoúrovňové a „mechanické“ vidění vedení, resp. „základní“ metodu vedení sw projektu je ve skutečnosti třeba různým způsobem obohacovat (základní strategie vedení sw projektu, tj. průběžné a postupné plánování, nezúžené plánování, role architektury, role SDLC, rekurse atd.), aby celkový proces / aktivita / činnost vedení softwarového projektu odpovídal typu a kontextu projektu.

(c) Ad nezúžený záběr plánování. Sanders (pozn. p.č. 3) v souvislosti s plánováním práce v rámci procesu vedení sw projektu mluví o: technickém plánování, plánování v souvislosti s kvalitou, plánování zdrojů.

(d) Postupné a průběžné plánování je dlouhodobý – skrze celé trvání projektu – koncepční záměr (koncepční strategie), který je v každodenním životě realizován „základní“ metodou vedení a údržbou plánu projektu tak, jak o nich bylo hovořeno výše.

(e) Vhodný a správný celkový postup vývoje spolu s použitými modely životního cyklu je dlouhodobý – skrze celé trvání projektu – koncepční záměr (koncepční strategie), který je v každodenním životě realizován „základní“ metodou vedení a údržbou plánu projektu tak, jak o nich bylo hovořeno výše.

(f) Následují definice některých termínů v souvislosti s životním cyklem a postupem vývoje používané v NASA příručce Software Management Guidebook a několik poznámek k těmto termínům a jejich interpretaci. Termíny: ***Životní cyklus softwaru*** (Software life cycle) je časový úsek, který začíná prvotními úvahami o softwarovém produktu a končí, když je software už natrvalo vyřazen z používání. Životní cyklus je typicky rozdělen do fází životního cyklu. ***Model životního cyklu*** (Life-cycle model) je rámec (framework), na který se mapují aktivity, metody, standardy, postupy, nástroje, produkty a služby (např. vodopád, spirálový). ***Fáze životního cyklu*** (Life-cycle phase) je nějaké rozdělení úsilí při vývoji softwaru do nepřekrývajících se časových úseků. Fáze životního cyklu jsou důležitými referenčními body pro vedoucího projektu. Ve fázi životního cyklu může být prováděno mnoho aktivit; aktivita může přesahovat více fází. ***Aktivita/ činnost*** (Aktivity) je část práce, která má dobře definovaná vstupní a výstupní kriteria. Činnosti mohou být obvykle rozděleny do samostatných kroků. Metoda (***Method***) je technika nebo přístup, potenciálně podpořená postupy a standardy, která stanovuje způsob provádění aktivit a dosažení žádaného výsledku. Poznámky: Zkratka SDLC znamená Software Development Life Cycle nebo také System Development Life Cycle (přičemž slovo system může být míněno jako software system nebo znamenat nadřazený systém, kde vyvíjený sw produkt je jen jednou z částí; o co konkrétně jde je třeba poznat z kontextu). Software Development Life Cycle je to samé jako Software Life Cycle. Pod pojmem fáze, budeme-li jej vůbec používat, se v žádném případě nesmí obecně provést ztotožnění jisté doby ve vývoji softwaru a určité aktivity softwarového inženýrství, např. analýza. Toto by bylo neskutečně omezující, v principu nepravdivé a skoro téměř vždy nerealizovatelné. Kořeny takovéhoto ztotožňování primárních aktivit softwarového inženýrství (tj. zjištění požadavků, analýza, návrh, implementace, testování, převedení do provozu) s jistými časovými úseky v době vývoje pochází ještě z dob, kdy modelům životního cyklu dominoval vodopád (waterfall) ve své rigidní podobě. Ve skutečnosti je většina činností sw inženýrství byť z různých důvodů v různé míře a za různým účelem potřeba po celou dobu životního cyklu. Dále různé části vyvíjeného sw produktu, zvláště u rozsáhlých anebo členitých sw produktů, mohou být v různém stavu rozpracovanosti. Principielně z těchto důvodů, když fa Rational ve svém Rational Unified Process[13] popisuje proces vývoje, tak používá dvě dimenze. Jedna je časová dimenze (organization along time; dříve nazývaná

[13] Na serveru fy Rational (viz zdroje) je vždy k dispozici tzv. white paper k poslední verzi jejich procesu vývoje, který se nyní jmenuje Rational Unified Process v. x.y.

Management‘s View) a druhá dimenze je členěná dle obsahu (organization along contetnt; dříve nazývaná Developer’s View). Časovou dimenzi člení na fáze, kterým ostentativně dali jména (typu Inception), která nelze zaměňovat se jmény primárních aktivit sw inženýrství, dříve to velmi zdůrazňovali. V posledních verzích Rational Unified Process se výslovně hlásí, co se základního členění této časové dimenze, tedy základní koncepce postupu vývoje týká, k přístupu popsaném v textu Anchoring the Software Process od Boehma (viz zdroje). Dimenze členěná dle obsahu reprezentuje činnosti softwarového inženýrství potřebné pro dobrou praxi sw inženýrství. Z výše uvedených důvodů je třeba si dávat pozor na interpretaci termínu fáze ve standardu ESA PSS-05-0.

(g) K potřebě či nepotřebě odborných znalostí uvádím snad dostatečně výmluvný výňatek ze Saturnina: ‘V určitém smyslu slova byl člověkem, který objevil celou řadu chemických pouček a pravidel nejrůznějšího druhu. Všechna tato pravidla už před ním objevili jiní, ale strýc o tom nic nevěděl, a nelze proto jeho zásluhy přehlížet.

Protože chemii vůbec nerozuměl, byly cesty jeho objevů posety trny a zkropeny potem, ale tím větší byla jeho radost ze získání zkušeností. Nebylo mu lze upřít sportovního ducha. *Podobal se člověku, který po zvládnutí malé násobilky prohlásil svým učitelům: „Dál už nic neříkejte. Nechci nic slyšet o tom, že pan Pythagoras, Eudoxus, Euklides, Archimédes a tak dále vymyslili to a to. Nepotřebuji týt z toho, co objevili jiní. Dejte mi papír, tužku a kružidlo a nechte mne na pokoji. Však já na to přijdu sám.“*

A strýček opravdu na leccos přišel. Tak například zjistil při pokusu, který měl velmi vzrušující průběh, že lít vodu do kyseliny je blbost, a vůbec mu nevadilo, že tento poznatek, korektněji vyjádřený, mohl získat z učebnice chemie pro nižší třídy škol středních, aniž by si při tom popálil prsty a zánovní vestu.

Chemie byla mu panenskou pevninou, roztočeným větrným zámkem, plným dveří, které se otvíraly tajemnými formulemi. Neznal názvosloví, ignoroval valenční koncovky a žasl, když mu ve zkumavkách a křivulích šuměly prudké reakce.’ (Jirotka, Z. *Saturnin.* Československý spisovatel, Praha, 1964.)

2.9 Mechanismus řešení problémů

Praktika číslo 9: Mechanismus řešení problémů
Zaměřeno na:
1. Evidence problémů
2. Řešení problémů

Pro začátek chceme docílit:
1. Evidence problémů: Je třeba docílit, aby problémy týkající se naplánovaných a předepsaných aktivit a naplánovaných a předepsaných pracovních produktů zjištěné jakýmkoli způsobem (tj. při odborných přezkoumáních, při zajišťování naplánovaných a předepsaných věcí (zajištění jakosti), při testování, při monitorování (jako součást činnosti vedení sw projektu), při společných přezkoumáních se zákazníkem atd.) byly oklasifikovány dle povahy, aby byla určena jejich závažnost a aby, byť jednoduchým a základním způsobem, byly evidovány.
2. Řešení problémů: Je třeba docílit, aby zjištěné a evidované problémy byly přijaty k vyřešení (někdo za daný zjištěný a evidovaný problém odpovídá), aby, byť jednoduchým a základním způsobem, byl sledován jejich stav, aby byly vyřešeny a aby záznamy o tomto byly archivovány.

Konkrétní nároky k počátečním cílům:
1. Je třeba stanovit způsob klasifikace druhu problému, závažnosti problému a způsob evidence zjištěných problémů. Pro klasifikaci problémů lze vycházet z Category and Priority Classifications for Problem Reporting, příloha C ve standardu MIL-STD-498. Pro evidenci problémů lze vycházet ze vzoru Software Problem Report (standard ESA PSS-05-0, Appendix E) a ze vzoru Discrepancy (NRCA) Report (standard NASA-STD-2100-99, NASA-DID-R004 Discrepancy (NRCA) Report; NRCA – nonconformance reporting and corrective action). Stanovený způsob klasifikace a evidence problémů je třeba při projektu dodržovat.
2. Je třeba stanovit obecný postup pro řešení problémů. Konkrétně jde o přijetí problému k vyřešení, určení kdo za vyřešení odpovídá, sledování stavu od přijmutí až po vyřešeni, uložení záznamu. Pro stanovení postupu je třeba vhodně vycházet z popisu nároku ve standardu MIL-STD-498 (5.17 Corrective action) anebo ve standardu ISO 12207 (6.8 Problem resolution process). Stanovený postup pro řešení problémů je třeba při projektu dodržovat.

Upozornění k interpretaci:

(a) V těchto konkrétních nárocích je nárokováno méně než pro řešení problémů předepisují standardy MIL-STD-498 a ISO 12207. Zde je nárokován opravdu minimální základ: evidence problému, sledování jeho stavu, vyřešení a archivaci záznamu o tomto. Standardy dále nárokují analýzu příčin, sledování trendů atd.

(b) Výše vyžadovaný obecný postup pro řešení problémů je nutno definovat v souladu s činnostmi testování, odborná přezkoumávání, zajišťování jakosti atd. prostě těmi, kde přirozeně dochází a má docházet

k odhalování nejrůznějších problémů. Problémy zjištěné při testování, odborných přezkoumáních, zajišťování jakosti atd. se musí řešit přirozeně pro tu kterou činnost a pro ten který problém. Vyžadovaný obecný postup pro řešení problémů nutně musí počítat s existujícími přirozenými mechanismy řešení nalezených problémů u testování, odborných přezkoumání atd. a musí být postaven na jejich vhodném referování a využívání (např. není vůbec myšleno, aby problémy zjištěné při obyčejném unit testování, dříve než je kód vůbec zařazen do konfiguračního řízení, byly administrovány nějakým rigidním mechanismem; na druhou stranu chyby, které se zjistí při kvalifikačním testování před dodáním sw produktu zákazníkovi k akceptaci musí být ošetřeny velmi disciplinovaně). V žádném případě není záměrem praktiky č. 9, aby se cokoli zdvojovalo. Při definici činnosti obecného postupu pro řešení problémů je nutno mít na mysli v podstatě jediné: zajistit, že identifikované problémy budou vyřešeny a „nezapadnou“.

3 Odborné zdroje pro jednotlivé činnosti/ oblasti softwarového inženýrství

V této kapitole jsou poskytnuty odborné materiály jak přímo podporující úvodní sadu konkrétních nároků, tak jsou zde poskytnuty další zdroje umožňující získat dostatek potřebných informací k mnohým aspektům (teorie, notace, produkty, metody, techniky, měření atd.) mnoha činností a oblastí softwarového inženýrství, tj. jakási „knihovnička softwarového inženýrství“. Prostě aby pro ty, kteří potřebují anebo mají zájem o odborné zdroje, bylo k dispozici dostatek studijního materiálu jak dále pokračovat v zlepšování praxe od úvodní sady konkrétních nároků.

Pokud materiál přímo podporuje úvodní sadu konkrétních nároků, tj. je v ní referován, pak má příslušné políčko v tabulce šedé pozadí (tmavější). Pokud je materiál velmi doporučen k přednostnímu prostudování, pak má políčko v tabulce šedé pozadí (světlejší). Dále jsou k dispozici pečlivě vybrané materiály pro další rozvíjení znalostí a tím pro podporu ustavení dobré praxe softwarového inženýrství při projektech.

U materiálů je uváděno, kde je lze získat, tj. URL adresa. Pokud je materiál obsažen v adresáři Odborné_zdroje, pak je toto uvedeno slovem disk.

K logice výběru odborných zdrojů. Odborné zdroje byly vybírány tak, aby pro každé téma byly tak, jak se to podařilo, dány k dispozici přístupné, stručné a prakticky orientované texty pokud možno vystihující opravdu ty nejpodstatnější věci. Chci-li co nejefektivněji podpořit, aby vybrané základní prvky dobré odborné praxe sw inženýrství se opravdu dostaly do reálné praxe v dané organizaci/ týmu, pak z praktických důvodů musí být odborná problematika zpočátku sdělena co nejstručněji. Příklad, činnost návrh je svět sám pro sebe, přiměřené této problematice by bylo poskytnout sadu monografií, technických zpráv a zásadních článků, to by ale stěží mohlo mít přímý dopad na praxi. Místo „tlusté knihy“ jsou tedy konkrétně poskytnuty články: Keep It Simple, Why You Should Use Routines Routinely, On the Criteria to be Used in Decomposing System into Modules, 4 + 1 View of Software Architecture atd. a předpisy, resp. vzory pracovního produktu návrh ze standardu armády, NASA atd. Takto na pár stranách je řečeno, co má splňovat výsledek a základní nároky na vlastní činnost od architektury po detailní design. Kromě stručných textů jsou často k dispozici dány i podrobnější materiály. Samozřejmě je dbáno na odbornou úroveň textů. Texty jsou až na výjimky k dispozici na Internetu anebo v adresáři Odborné_zdroje. Pouze výjimečně je k dispozici pouze papírová forma.

Ve sloupečku „Místo“ je uváděno, kde lze daný materiál získat. Lze-li materiál získat v některém z míst uvedených v příloze č. 2, pak je uvedeno jméno místa nebo zkratka, např. SEI jedná-li se o materiál ze Software Engineering Institute. Jinak je uváděna URL adresa. Pokud je materiál také k dispozici v adresáři Odborné_zdroje, pak je uvedeno slovo disk.

3.1 Požadavky na software

Co činnost/ téma konstituuje:

Dobrý obrázek si lze udělat prozkoumáním následujících zdrojů:

- IEEE CS & ACM projekt Guide to the Software Engineering Body of Knowledge (konkrétně viz obsahy Knowledge Area Descriptions)
- Technická zpráva A Software Engineering Body of Knowledge, v 1.0, SEI-99-TR-004 ze Software Engineering Institute
- Curriculum Modules ze Software Engineering Institute.

Toto platí i pro další činnosti/ témata a již to u nich nebude v této kapitole dále opakováno. Jak lze tyto zdroje získat viz příloha č. 2 a sekce č. 3.11. V těchto zdrojích se lze dočíst i různé možné definice těchto činností/ témat.

Zdroje:

Text	Místo
Zásady	
Writing Effective Natural Language Requirements Specification, článek v CrossTalk, February 1999 komentář: článek obsahuje zásady jak psát, tj. co má splňovat dokument obsahující specifikaci požadavků zákazníka. Tyto zásady lze bezprostředně aplikovat. (4 strany)	STSC/ CrossTalk
Making Requirements Management Work for You, článek v CrossTalk, April 1999 komentář: stručný základní přehled co vlastně představuje celý okruh problémů týkající se požadavků – Requirements Engineering (tj. jejich získání, analýza, specifikace, ověření, údržba, atd.) (4 strany)	STSC/ CrossTalk
Vzory/ předpisy pro pracovní produkt specifikace požadavků zákazníka	
Vzor dokumentu specifikace požadavků komentář: dokument obsahující specifikované požadavky je zásadní věc. Existuje řada vzorů. Přímo k dispozici z čeho lze hned čerpat je např.: standard MIL-STD-498 (DIDs), standard NASA-STD-2100-91 (DIDs), NASA příručky Recommended Approach to Software Development a Manager's Handbook for Software Development, standard ESA PSS-05-0, materiály SEPO pro úroveň L2, Process Impact (Software Requirements Specification Template v Process Goodies) atd. Je třeba nějaký vzor vybrat, přizpůsobit a začít používat. Vzor předepíše co má specifikace obsahovat a její strukturování. Zásady jak	viz jednotlivé uvedené materiály a příloha č. 2

toto psát viz výše. Doporučuji vyjít z a respektovat příslušný MIL-STD-498 DID. Pro různé kontexty v organizaci mohou být vyrobeny různé vzory. Vzor musí nárokovat specifikaci vnějších rozhraní, vzor pro specifikaci rozhraní je k dispozici v MIL-STD-498.	
Software Requirements Specification (SRS) Template komentář: vzor dokumentu specifikace požadavků na základě standardu MIL-STD-498; poskytováno v rámci podpory CMM L2. Velmi dobré pokrytí specifikace rozhraní.	SEPO; disk
Celkově	
STSC Requirements Engineering and Design Technology Report komentář: obsahuje základní přehled problematiky. Dále obsahuje přehled desítek CASE nástrojů vhodných pro zvládání požadavků (získání, formulace, uložení, údržba, vazby na další reprezentace vyvíjeného produktu) a návrhu.	STSC; disk
Software Requirements: A Pragmatic Approach, K. E. Wiegers komentář: z této knihy je k dispozici 5 kapitola a apendix: Chapter 1. The Essential Software Requirement, Chapter 2. Requirements from the Customer's Perspective, Chapter 9. Documenting the Requirements, Chapter 13 Setting Requirements Priorities, Chapter 18. Links in the Requirements Chain, Appendix: Current Requirements Practice Self-Assessment. Takto je k dispozici dobře čitelné prakticky orientované zpracování důležitých témat, např. dokumentace požadavků.	Process Impact; disk
Články k Software Requirements od Process Impact komentář: firma Process Impact dává k dispozici řadu stručných prakticky orientovaných článků k různým aspektům tématu požadavky. Seznam článků k dispozici (v závorce je původní místo publikace): - First Things First: Prioritizing Requirements (*Software Development*, vol. 7, no. 9) - Automating Requirements Management (*Software Development*, vol. 7, no. 7) - Writing Quality Requirements (*Software Development*, vol. 7, no. 5) - Listening to the Customer's Voice (*Software Development*, vol. 5, no. 3) - In Search of Excellent Requirements (*Journal of the Quality Assurance Institute*, vol. 9, no. 1)	Process Impact
SEI Curriculum Modules k tomuto tématu: - Software Requirements, Curriculum Module SEI-CM-19-1.2 - Formal Specification of Software, Curriculum Module SEI-CM-8-1.0 - Lecture Notes on Requirements Elicitation, SEI-93-EM-10 komentář: viz sekce č. 3.11.	SEI; disk
Software Engineering Process Office, SPAWAR System Center vybrané materiály k tomuto tématu: - SSC San Diego Requirements Management Policy - Requirements Management Guidebook - Software Requirements Specification (SRS) Template (MIL-STD-498) komentář: toto jsou materiály pro podporu klíčové oblasti Requirements Management v úrovni 2 (level 2) v rámci CMM (Capability Maturity Model). Dále komentář viz příloha č. 2.	SEPO; disk
Články v CrossTalk k tomuto tématu: Následuje seznam vybraných článků (nejsou to ani zdaleka všechny) k tomuto tématu publikovaných v CrossTalku: - Making Requirements Management Work for You (April 1999) - Writing Effective Natural Language Requirements Specifications (February 1999) - Validating Software Requirements (November 1999) - Object-Oriented Specification Documentation: Building Requirements Specifications from OO Models (June 1997)	STSC/ CrossTalk
Requirements, Testing, and Metrics, 15th Annual Pacific Northwest Software Quality Conference komentář: text se zabývá principielním vztahem požadavků, testováním na úrovni systému a co v této souvislosti sledovat a měřit.	SATC – NASA (stránka: Project Support & Outreach)
Doing Requirements Right the First Time! Software Technology Conference, 1998 komentář: text se zabývá definicí, verifikací a řízením požadavků (requirements management). Text je psán z kontextu velkých projektů v rámci NASA – díky tomu také ta palčivá potřeba, aby tato oblast byla maximálně zkonsolidovaná. Nicméně na principech to nic nemění. Text uvádí příklady použití nástrojů na obhospodařování požadavků.	SATC – NASA (stránka: Project Support & Outreach
Writing Effective Requirements Specification, Software Technology Conference, 1997 komentář: text zabývající se vhodným zápisem požadavků zákazníka. Text je napsán na základě analýzy desítek dokumentů se specifikací požadavků (konkrétně NASA System/Software Requirements Specifications (SRS) documents).	SATC – NASA (stránka: Project Support & Outreach)
Prostředek pro modelování	
Unified Modeling Language (UML) komentář: UML je standardní, mocný a zároveň jednoduchý vyjadřovací prostředek pro reprezentaci vyvíjeného softwarového systému na úrovních analýzy, návrhu a částečně i realizace. Podrobnější komentář: díky systematickému úsilí pánů Booche, Rumbaugha, Jacobsona a dalších z firmy Rational, díky organizaci OMG (Object Management Group) a díky úsilí firem, které se nakonec také přidaly (např. Hewlett-Packard, IBM, Oracle, PLATINUM Technology, SAP, Unisys Corporation atd.) vznikl standardní prostředek pro specifikaci, zobrazení, konstrukci a dokumentaci artefaktů (pracovních produktů), které reprezentují vyvíjený sw produkt. Kořeny UML jsou v notaci tří metodologií Booch, OMT (Rumbaugh), OOSE (Jacobson). Tito lidé od poloviny 90. let pracovali na sjednocení modelovacího jazyka (bylo několik předběžných verzí UML). Nakonec byl UML přijat OMG jako standard (poslední verze UML je OMG Unified Modeling Language Specification, Version 1.3, OMG dokument ad/99-06-08). UML je jednoduchý a zároveň mocný prostředek. UML dává k dispozici opravdu bohatou sadu vyjadřovacích prostředků, zjednodušeně diagramů, které umožňují dobře reprezentovat funkčnost, chování, komunikování a dekomposici (tj. čtyři vlastnosti, které potřebujeme	Rational (www.rational.com/uml) disk http://uml.shl.com; disk http://www.omg.org

specifikovat dle „A Survey of Structured and Object-Oriented Software Specification Methods and Techniques," In *ACM Computing Surveys*, Vol. 30, No. 4, December 1998, pp. 459-527) pro úrovně abstrakce analýzy, návrhu a realizace. O UML byla publikováno řada knih. (Výňatek z Předmluvy k UML v.1.3 „The Unified Modeling Language (UML) provides system architects working on object analysis and design with one consistent language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling. This specification represents the convergence of best practices in the object-technology industry. UML is the proper successor to the object modeling languages of three previously leading object-oriented methods (Booch, OMT, and OOSE). The UML is the union of these modeling languages and more, since it includes additional expressiveness to handle modeling problems that these methods did not fully address." UML je v počítačové komunitě široce přijímán jako standard (a toto je podpořeno faktem, že UML kromě notace obsahuje i definici sémantiky pro výrobce nástrojů atd.). Toto je velmi cenné, protože se snad zredukují neprincipielní problémy plynoucí z přemíry soupeřících notací.	

Tabulka 3-1: Zdroje pro téma požadavky na software

3.2 Návrh softwaru

Zdroje:

Text	Místo
Zásady, principy, detailní design	
Příslušné praktiky z Best Practices Column in IEEE Software komentář: viz sekce č. 3.11. Zejména se jedná o následující články: - Why You Should Use Routines Routinely; - Keep It Simple (skvělý maximálně stručný přehled základních koncepčních zásad pro design a programování); - Missing in Action: Information Hiding.	Construx Software
Why You Should Use Routines... Routinely, článek v IEEE Software, July/ August 1998 komentář: článek osahuje vysvětlenu základní ideu a důvody proč existuje, kdy a jak se má používat jeden z nejzákladnějších konceptů programování, kterým je procedura (jinak též rutina, funkce, podprogramu, subrutina atd.) (3 strany). Pozn. viz také poznámky k přednášce Programming in the small z kurzu SYDE 221 Software Design.	Construx Software
On the Criteria to be Used in Decomposing Systems into Modules, Comm. of the ACM, Dec. 1972 komentář: slavný článek od D.L. Parnase týkající se dekompozice systému do modulů. Zavádí a ilustruje koncept dnes zvaný *information hiding*, tj. o vnitřních věcech (data, reprezentace dat, algoritmy atd.) dané dekompoziční jednotky nemá zbytek systému vědět a ani dělat žádné předpoklady; jediné co se ví je jasně dané rozhraní. Otázkou, je jaká dekompozice charakterizuje dobrý design, co má být vnitřní záležitostí dekompozičních jednotek a co rozhraním. Těmto fundamentálním otázkám se článek věnuje a ilustruje je na příkladě. Článek byl psán v době, kdy k dobré praxi návrhu již patřila dekompozice systému do částí (modulů) s jasně definovaným rozhraním, tento článek jde dál a diskutuje problém jak dekomponovat a co mají jednotlivé části systému o sobě vědět. Nepochybně jeden z nejvýznamnějších článků k softwarovému inženýrství vůbec.	ACM Classic of the month
Poznámky k přednáškám z kurzu SYDE 221 Software Design k tomuto tématu: - Programming in the small - Modules - Abstract Data Types komentář viz sekce č. 3.10.	SYDE 221
Softwarová architektura	
The 4+1 View Model of Architecture, technická zpráva fy Rational, též IEEE Software, Nov 1995 komentář: dobrý, stručný, nezužující úvod do tématu softwarové architektury; z technického pohledu. Velmi cenné jako stručný dobrý úvod.	Rational
On the Definition of Software System Architecture, technická zpráva USC/CSE-95-TR-500 komentář: Technická zpráva týmu B. Boehma, která probírá softwarovou architekturu hlavně z pohledu její mnohostranné role v celém životním cyklu a vedení sw projektu. Velmi cenné svým zaměřením.	CSE
Softwarová architektura, norma 24/97, příloha B komentář: široce pojatý úvod k tématu sw architektura je k dispozici v příloze B normy 24/97. Je popisováno technické hledisko. Jsou uvedena některá příbuzná témata a je uvedeno dost literatury. Příloha B obsahuje vysvětlení co to je sw architektura a základní nároky, které mají být při projektu vzhledem k sw architektuře splněny.	Norma 24/97
Technické/ výzkumné zprávy ze SEI, CMU katedry počítačů, týmu Boehma komentář: na téma softwarová architektura lze najít dostatek odborných textů pokrývajících celé téma na následujících místech: - Software Engineering Institute, Carnegie Mellon University - Carnegie Mellon University, School of Computer Science (několik projektů týkajících se sw architektury, velmi mnoho článků a technických zpráv je k dispozici) - Center for Software Engineering, University of Southern California Pozn. literatury na toto téma je samozřejmě nepoměrně více, nicméně uvedená místa představují velmi dobré zdroje. Dobré a relativně velmi stručné přehledové úvodní texty jsou následující: - An Introduction to Software Architecture; CMU-CS-94-166 (více techničtější) - Software Architecture: An Executive Overview; SEI-96-TR-003 - Coming Attractions in Software Architecture; SEI-96-TR-008 Pozn. jako opravdu poctivý úvod do tématu se všemi nejen technickými souvislostmi, kde se nic důležitého nevynechá a přitom rozsah textu k prostudování je stále velmi rozumný (do 100 stran, velký font, řídké řádkování; při vynechání toho, co se zjevně překrývá, je rozsah ještě menší) velmi doporučuji	 SEI http://www.cs.cmu.edu CSE http://www.cs.cmu.edu; disk SEI SEI

následující kombinaci v následujícím pořadí: 1. The 4+1 View Model of Architecture 2. An Introduction to Software Architecture 3. Software Architecture: An Executive Overview 4. On the Definition of Software System Architecture (Pozn. tyto texty mají pomoci při dosahování jednoho konkrétního nároku na oblast návrhu, který zhruba zní: při projektu musí být kladen maximální důraz na vytvoření robustní sw architektury, která odpovídá určujícím požadavkům na produkt, tato architektura musí být po celý další život produktu respektována a neničena prací jednotlivých vývojářů při detailním designu a programování. To samé platí pro konkrétní rozhodnutí pro jednotlivé momenty návrhu, např. jak spolu komunikují moduly, jak se vyvolávají služby atd.)	Rational http://www.cs.cmu.edu; disk SEI CSE; disk
Design celkově	
STSC Requirements Engineering and Design Technology Report komentář: viz sekce č. 3.1 Požadavky na software.	STSC; disk
SEI Curriculum Modules k tomuto tématu: - Introduction to Software Design, Curriculum Module SEI-CM-2-2 - Software Design Methods for Real-Time Systems, Curriculum Module SEI-CM-22-1.0 - Concepts of Concurrent Programming, Curriculum Module SEI-CM-24 - Understanding Program Dependencies, Curriculum Module SEI-CM-26 komentář: viz sekce č. 3.11.	SEI; disk
Software Quality Metrics for Object-Oriented Environments, CrossTalk, April 1997 komentář: text se zabývá otázkou co v principu měřit pro určení jakosti (z technického hlediska) pracovních produktů návrh a zdrojový kód při objektově orientovaném přístupu.	SATC – NASA (stránka: Project Support & Outreach)
checklists	
Seznamy kontrolních otázek (checklists) od fy Construx komentář: firma Construx dává k tomuto tématu k dispozici následující seznamy kontrolních otázek: - Architecture - High-Level Design.	Construx
Prostředek pro modelování	
Unified Modeling Language (UML) komentář: viz tabulka č. 3-1.	Rational (www.rational.com/uml) disk

Tabulka 3-2: Zdroje pro téma návrh softwaru

3.3 Implementace softwaru

Zdroje:

Text	Místo
Zásady, principy	
Příslušné praktiky z Best Practices Column in IEEE Software komentář: viz sekce č. 3.11. Zejména se jedná o následující články: - Why You Should Use Routines, Routinely; - Keep It Simple (skvělý maximálně stručný přehled základních koncepčních zásad pro design a programování); - Who Cares About Software Constructions?	Construx Software
Why You Should Use Routines... Routinely; viz sekce 3.2 Návrh softwaru	Construx Software
Go To Statement Considered Harmful, Communications of the ACM, March 1968 komentář: slavný článek od E.W. Dijkstry o „škodlivosti“ příkazu go to.	ACM Classic of the month
Program Development by Stepwise Refinement, Communications of the ACM, April 1971 komentář: článek od N. Wirtha. Jeden ze základních článků o tom jak dobře navrhovat a psát programy. Přístup *shora dolů* je průběžně ilustrován na problému osmi královen.	ACM Classic of the month
Poznámky k přednáškám z kurzu SYDE 221 Software Design k tomuto tématu: - Programming in the small komentář viz sekce č. 3.10.	SYDE 221
Celkově	
Programovací standardy, norma 24/97, příloha B komentář: stručně popisuje zásady dobrého programování v malém včetně strukturovaného programování. Dostatek literatury v češtině.	Norma 24/97
Software Quality Metrics for Object-Oriented Environments, CrossTalk, April 1997 komentář: text se zabývá otázkou co v principu měřit pro určení jakosti (z technického hlediska) pracovních produktů návrh a zdrojový kód při objektově orientovaném přístupu.	SATC – NASA (stránka: Project Support & Outreach)
Programovací standardy	
An Abbreviated C++ Code Inspection Checklist komentář: tento text může sloužit jako velmi propracované, systematicky vytvořené a přitom ne moc rozsáhlé programovací standardy pro C++.	Formal Technical Review Archive
Seznamy kontrolních otázek (checklists) od fy Construx komentář: firma Construx dává k dispozici sadu checklists, která velmi podrobně pokrývá problematiku, kterou nazývají konstrukce, přičemž převážně jde o implementaci a detailní návrh. Tyto checklists lze kromě odborných přezkoumání z výhodou použít pro tvorbu programovacích standardů. Následuje seznam příslušných checklists: - Constructing a Routine - High-Quality Routines - High-Quality Modules	Construx

- Data Creation - Naming Data - General Consideration in Using Data - Fundamental Data - Organizing Straight-Line Code - Conditionals - Loops - Unusual Control Structures - Control-Structures Issues - Layout - Self-Documenting Code - Good Commenting Technique - Configuration Management - Debugging - Incremental Integration Strategy - Evolutionary Delivery - Code Changes	
Prostředek pro modelování	
Unified Modeling Language (UML) komentář: viz tabulka č. 3-1.	Rational (www.rational.com/uml) disk

Tabulka 3-3: Zdroje pro téma konstrukce/ implementace softwaru

3.4 Testování softwaru

Zdroje:

Text	Místo
Celkově	
Testing (sekce 5.3.2.3 část 1 a sekce 4.2.4 část 2, ESA PSS-05-0) komentář: tyto sekce v standardu ESA pro sw inženýrství obsahují základní schematické vysvětlení testování a základní nároky. Standard dále obsahuje popis, co se má a kdy v průběhu projektu vzhledem k této činnosti provádět.	Standard ESA PSS-05-0
Little Book of Testing, Volume I, Overview and Best Practices **Little Book of Testing, Volume II, Implementation Technique** komentář: popisuje principy, činnosti, postupy, náležitosti a vazby na okolí. Velmi cenné pro svoji stručnost a množství informací. Maximálně principielně a zároveň prakticky orientované. Pozn. všechny texty od SPMN jsou jedinečné, na jejich vzniku se podíleli lidé z armády, průmyslu, sw firem a univerzit; oponentura byla extensivní; koncentrují zkušenosti z bezpočtu sw projektů, které jsou většinou velký problém.	SPMN; disk
Software Test Planning and Management Guide komentář: podrobný popis jak postupovat při plánování a provádění testování tak, jak jej vyžaduje CMM L3 – tedy pořádně. Samotné techniky testování zde nejsou podrobně vysvětlovány. Velmi cenný materiál právě pro zaměření na celkový obrázek a souvislosti.	SEPO; disk
comp.software.testing Frequently Asked Questions (FAQ) komentář: FAQ diskusní skupiny na téma testování. Obsahuje mnoho odkazů na další zdroje.	Internet FAQ Consortium
STSC Software Test Technology Report komentář: celkový přehled problematiky testování a přehled problematiky podpory testování k tomu určenými nástroji.	STSC; disk
Developing a Testing Maturity Model: Part I & Part II; CrossTalk August 1996 & September 1996 komentář: tyto dva články v CrossTalku popisují jak zlepšit proces testování.	STSC/ CrossTalk
Testing Object-Oriented Systems: A Status Report; CrossTalk April 1995 komentář: přehled problematiky z pohledu objektově orientovaných systémů.	STSC/ CrossTalk
Články v CrossTalk na téma testování komentář: v CrossTalk vyšlo mnoho stručných článků k nejrůznějším otázkám a problémům testování.	STSC/ CrossTalk
Vybrané články – „White Papers“ firmy Quality Checked Software: - An Introduction to Software Testing - Software Testing and Software Development Lifecycles - Why Bother to Unit Test - Designing Unit Test Cases - Organizational Approaches for Unit Testing - Structural Coverage Metrics - Designing Testable Ada - Achieving Testability when using Ada Packaging and Data Hiding Methods - C++ - „It’s Testing, Jim, But Not As We Know It“ - AdaTEST, Cantata and DOD Mil Std 498 - AdaTEST, Cantata and ESA PSS-05-0, Issue 2 - AdaTEST, Cantata and the SEI Capability Maturity Model - Cantata and AdaTEST in ISO9001/BS5750/TickIT Software Quality Process komentář: viz příloha č. 2.	Quality Checked Software
Software Engineering Process Office, SPAWAR System Center vybrané materiály k tomuto tématu: Software Testing: - Software Test Planning and Management Guide	SEPO; disk

- Software Test Plan (STP) Template (MIL-STD-498) - Software Test Report (STR) Template (MIL-STD-498) komentář: toto jsou materiály pro podporu testování v rámci klíčové oblasti Software Product Engineering v úrovni 3 (level 3) v rámci CMM (Capability Maturity Model). Dále komentář viz příloha č. 2.	
Requirements, Testing, and Metrics, 15th Annual Pacific Northwest Software Quality Conference komentář: text se zabývá principielním vztahem požadavků, testováním na úrovni systému a co v této souvislosti sledovat a měřit.	SATC – NASA (stránka: Project Support & Outreach)
Testing – When Do I Stop? International Testing and Evaluation Conference 1994 komentář: text se zabývá otázkou, do kdy má smysl testovat.	SATC – NASA (stránka: Project Support & Outreach)
Unit testování	
Unit Analysis and Testing; Curriculum Module SEI-CM-9-2.0 komentář: přehled problematiky testování na úrovni softwarových unit.	SEI
„White Papers" firmy Quality Checked Software viz výše.	Quality Checked Software
Integrační testování	
Object-Oriented Integration Testing; Communications of the ACM September 1994 komentář: přehled problematiky integračního testování objektově orientovaných systémů.	disk
„White Papers" firmy Quality Checked Software viz výše.	Quality Checked Software

Tabulka 3-4: Zdroje pro téma testování softwaru

3.5 Údržba (a rozvoj) softwaru

Zdroje:

Text	Místo
A Case Study in Software Maintenance; Technical Report SEI-93-TR-8 komentář: přehled problémů typických pro proces údržby a návrh jejich řešení. Materiál vznikl systematickou analýzou mnoha projektů, kde se děje údržba softwaru.	SEI
Analysis of a Software Maintenance System: A Case Study; Technical Report SEI-92-TR-31 komentář: přehledová analýza, popis a komentář, jak funguje zkonsolidovaný proces údržby u jednoho středně velkého projektu. Tato zpráva se dále zabývá tím, jak bylo dosaženo dobrého procesu údržby.	SEI
SEI Curriculum Modules k tomuto tématu: - Understanding Program Dependencies, Curriculum Module SEI-CM-26 - Models of Software Evolution: Life Cycle and Process, Curriculum Module SEI-CM-10-1.0 komentář: viz sekce č. 3.11.	SEI; disk
Články v CrossTalk k tomuto tématu: Následuje seznam vybraných článků k tomuto tématu publikovaných v CrossTalku: - Measurement to Manage Software Maintenance (July 1997)	STSC/ CrossTalk
Vzory dokumentů	
Change-Control Plan komentář: vzor pro plán „Řízení změn", což je při údržbě rutina.	Construx (Document Templates); disk

Tabulka 3-5: Zdroje pro téma údržba a rozvoj softwaru

3.6 Konfigurační řízení softwaru

Zdroje:

Text	Místo
Celkově	
Software Configuration Management (část 2, kapitola 3, ESA PSS-05-0) komentář: tato kapitola standardu ESA pro sw inženýrství obsahuje základní vysvětlení konceptů CM a základní nároky. Dále obsahuje popis co se má a kdy v průběhu projektu vzhledem k této činnosti provádět.	Standard ESA PSS-05-0
Little Book of Configuration Management komentář: popíše co to je CM, proč je důležité, základní činnosti, vazby na okolí, způsob zavedení, „best" practices, základní pravidla, indikátory funkčnosti. Velmi cenné pro svoji stručnost a široký záběr.	SPMN; disk
Software Configuration Management Technology Report; September 1994 komentář: popisuje základní principy CM, náležitosti jeho zavádění do organizace a plánování při projektech. Report popisuje nástroje, které podporují konfigurační řízení.	disk (tento dokument byl nahrazen novější verzí)
Software Configuration Management Technologies and Applications; May 1999 komentář: nejnovější verze technologické zprávy (Technology Report) pro téma konfigurační řízení. Stručný (cca 50 stran) kompletní přehled problematiky s mnoha praktickými příklady, s komentovanou literaturou, se stručným přehledem nástrojů na trhu atd.	STSC; disk
NASA Software Configuration Management Guidebook; NASA CM Gdbk komentář: stručná příručka (a) vysvětlující základní definice a koncepty, (b) popisující vlastní činnost (proces) konfigurační řízení, (c) popisující, co se má kdy dělat v průběhu životního cyklu vyvíjeného softwaru (resp. jeho části, pokud různé části jsou v různém stavu rozpracovanosti), (d) popisující organizaci CM a vztahy ke zbytku projektu, (e) popisující přizpůsobení CM konkrétnímu projektu.	SATC – NASA; SWG – NASA; disk
Configuration Management Frequently Asked Questions (FAQs)	Internet FAQ Consortium

komentář: FAQs diskusní skupiny comp.software.config-mgmt na Usenetu. Kromě vlastní problematiky obsahují obsáhlý přehled nástrojů na podporu konfiguračního řízení.	
Configuration Management Guidance; MIL-HDBK-61 komentář: armádní příručka pro činnost konfigurační řízení. Obsahuje také vzory pro SCM plány, formuláře a kontrolní seznamy.	Standardy
Configuration Management Yellow Pages komentář: obrovský zdroj informací k problematice konfiguračního řízení. Viz Příloha č. 2.	viz příloha č. 2
Články v CrossTalk k tomuto tématu: Následuje seznam vybraných článků (nejsou to ani zdaleka všechny) k tomuto tématu publikovaných v CrossTalku: - Software Configuration Management: Don't Buy a Tool First (November 1997) - Achieving the Best Possible Configuration Management Solution (September 1996) - Software Configuration Management Terminology (January 1995) - Demystifying Software Configuration Management (May 1995) - Software Configuration Management by MIL-STD-498 (June 1996)	STSC/ CrossTalk
SEI Curriculum Modules k tomuto tématu: - Software Configuration Management, Curriculum Module SEI-CM-4-1.4 komentář: viz sekce č. 3.11.	SEI; disk
Software Engineering Process Office, SPAWAR System Center vybrané materiály k tomuto tématu: - SSC San Diego Software Configuration Management Policy - Software Configuration Management Process - Generic Software Configuration Management Plan komentář: toto jsou materiály pro podporu klíčové oblasti Software Configuration Management v úrovni 2 (level 2) v rámci CMM (Capability Maturity Model). Dále komentář viz příloha č. 2.	
Vzory plánu CM a formulářů	
Configuration Management (CM) Plans: The Beginning to Your CM Solution komentář: přehledová zpráva na toto téma. Obsahuje vyjádření k plánu konfiguračního řízení. Dle technologického zprávy STSC je toto vynikající zdroj pro SCM plány!	SEI; disk
Configuration Management Plan Outline komentář: vzor pro plán konfiguračního řízení s komentáři.	SPC; disk
Configuration Management Forms komentář: formuláře zamýšlené pro podporu procesu konfiguračního řízení, které dává k dispozici SEPO, SPAWAR Systems Center.	SEPO; disk

Tabulka 3-6: Zdroje pro téma konfigurační řízení softwaru

3.7 Validace a verifikace softwaru

Zdroje:

Text	Místo
Validace a verifikace celkově	
Software Verification and Validation (část 2, kapitola 4, ESA PSS-05-0) komentář: tato kapitola standardu ESA pro sw inženýrství obsahuje základní vysvětlení konceptů V&V a základní nároky. Dále obsahuje popis, co se má a kdy v průběhu projektu vzhledem k této činnosti provádět.	Standard ESA PSS-05-0
Software Product Validation and Verification (sekce 5.2.6 v příručce Software Management Guidebook) komentář: velmi stručný celkový popis problematiky a základní přehled konceptů. Dále přehled technik se stručným komentářem.	Software Management Guidebook, NASA-GB-001-96
SEI Curriculum Modules k tomuto tématu: - Introduction to Software Verification and Validation, Curriculum Module SEI-CM-13-1.1 - The Software Technical Review Process, Curriculum Module SEI-CM-3-1.5 komentář: viz sekce č. 3.11.	SEI; disk
Poznámky k přednáškám z kurzu SYDE 221 Software Design k tomuto tématu: - Software defects - Debugging - Design and code reviews komentář viz sekce č. 3.10.	SYDE 221
Odborná přezkoumání celkově	
Peer Review Process komentář: stručný přehledový text popisující odborná přezkoumání. Popisuje tři úrovně formálnosti odborných přezkoumání: Walkthrough, Technical review, Formal Inspection.	SEPO; disk
Software Formal Inspection Guidebook, NASA-GB-A302 komentář: příručka popisující postup a náležitosti formálních inspekcí. Tato příručka je vydána k stejnojmennému standardu (NASA-STD-2202-93), který je rovněž k dispozici v SATC – NASA. Příručka je velmi užitečná, kromě toho, že je k dispozici postup formálních inspekcí, tak obsahuje přes 20 kontrolních seznamů (checklist) týkající se základních pracovních produktů primárních aktivit sw inženýrství (tj. požadavky, architektura, detailní návrh, kód); také popisuje *co* a *kdy*.	SATC – NASA; disk
Software Inspections & Technical Reviews: Transcending the Dogma **Formal Technical Reviews, Across All Maturities** komentář: praktický přínos odborných přezkoumání spočívá v odhalení chyb včas tak, aby nepokračovaly dále v procesu vývoje při vynaložení relativně malých nákladů; příklad tím, že je	Institute for Zero Defect Software

přezkoumán návrh, tak se potenciálně zabrání velkým nákladům na opravu v době testování, atd. Smysl má přezkoumávat všechny pracovní produkty včetně plánů. Potíž spočívá v tom, jak tato přezkoumání koncipovat a provádět. Je zřejmé, že pokud se přezkoumává „halabala“, tak se náklady na přezkoumání vyplýtvají a nic se nezíská, jen znechucení. Je zřejmé, že pokud se do kontextu, kde není moc věcí zkonsolidovaných zavedou přísně formální přezkoumání, tak je to nepřiměřené a těžko prosaditelné. Empiricky se zjistilo, že velmi přísně a formálně prováděné inspekce jsou velmi efektivní. Také se však ukázalo, že pokud se zavádí tam, kde pro ně není půda, skončí to nedobře. Závěr: (i) odborná přezkoumání se musí dělat; (ii) způsob odborných přezkoumání musí být takový, aby v daném prostředí mohl být realizován efektivně. Výše uvedené dva články popisují, co je kdy přiměřené. Podobně zaměřený je i článek **Software Inspections: How to Diagnose Problems and Improve the Odds of Organizational Acceptance** z Crosstalk	STSC/ CrossTalk
Overcoming the NAH Syndrome for Inspection Deployment komentář: stručný článek, který popisuje způsob, konkrétně dva druhy experimentů, kterým lze relativně rychle demonstrovat velký přínos odborných přezkoumání, zde konkrétně inspekcí kódu. Takto lze překonat nedůvěru v tuto základní praktiku sw inženýrství a zavést ji snadněji do praxe.	Proceedings of International Conference on Software Engineering 1998, pp. 371 – 378; disk
Články v CrossTalk k tomuto tématu: Následuje seznam vybraných článků (nejsou to ani zdaleka všechny) k tomuto tématu publikovaných v CrossTalku: - Setting Up a Software Inspection Program (February 1997) - Software Inspections: How to Diagnose Problems and Improve the Odds of Organizational Acceptance (August 1997) - Software Inspections at Applicon (October 1994) - Optimizing Software Inspections (March 1998) - Defining Review Levels for Software Documentation (January 1996)	STSC /CrossTalk
Texty v Formal Technical Review Archive	Formal Technical Review Archive
Články k Software Technical Review od Process Impact komentář: firma Process Impact dává k dispozici několik stručných prakticky orientovaných článků k technickým přezkoumáním. Seznam článků k dispozici (v závorce je původní místo publikace): - The Seven Deadly Sins of Software Reviews (*Software Development*, vol. 6, no. 3) - Improving Quality through Software Inspections (*Software Development*, vol. 3, no. 4)	Process Impact
Software Engineering Process Office, SPAWAR System Center vybrané materiály k tomuto tématu: - SSC San Diego Peer Review Policy - Peer Review Process - Formal Inspection Process komentář: toto jsou materiály pro podporu klíčové oblasti Peer Reviews v úrovni 3 (level 3) v rámci CMM (Capability Maturity Model). Dále komentář viz příloha č. 2.	SEPO; disk
Odborná přezkoumání – Checklists (kontrolní seznamy) a formuláře	
Kontrolní seznamy (checklists) komentář: pro odborná přezkoumání, ať už jde o kterýkoli typ, jsou velkou pomůckou kontrolní seznamy, tj. seznamy věcí, na které je třeba se zaměřit (checklists). Dvě rozsáhlé sady kontrolních seznamů jsou k dispozici v Software Formal Inspection Guidebook od NASA a v Formal Inspection Process od SEPO (v rámci podpory CMM L 3). Další vynikající sadu kontrolních seznamů nabízí k dispozici firma Construx. Tyto kontrolní seznamy spolu s dalšími uvedenými v tomto textu mohou velmi dobře sloužit jako příklad pro vytvoření vlastních kontrolních seznamů pro všechny zásadní pracovní sw produkty a plány.	STSC – NASA; SEPO; Construx; disk
An Abbreviated C++ Code Inspection Checklist komentář: kontrolní seznam (checklist) vzhledem k programovému standardu pro jazyk C++ pro účely přezkoumání. Programový standard se snaží předcházet chybám, které mohou vzniknout díky nevhodnému používání jazyka C++ a vzhledem k jeho specifickým rysům. Tento text je vlastně stručný návod na formální přezkoumání zdrojového kódu, kontrolní seznam a programovací standardy pro C++. Text je proto referován i v sekci č. 3.3 Implementace softwaru. Jde o velmi dobrý a přitom stručný text, který čerpá z mnoha zkušeností.	Formal Technical Review Archive
Weiss and Kimbrough Inspection Materials komentář: sada materiálů podporujících inspekce používaná v jedné pobočce Motoroly v USA. Obsahuje stručný úvod do inspekcí a zdůraznění, čím se inspekce liší od přezkoumání; při inspekcích jde primárně o detekci chyb před tím, než pracovní produkt pokračuje dál; při přezkoumáních jde o detekci chyb s možností vymyslet řešení problému v průběhu práce na pracovním produktu; oboje inspekce i přezkoumání jsou typy odborných přezkoumání. Dále materiály obsahují jednoduché standardy pro: specifikaci návrhu, dokumenty podléhající inspekci, specifikaci požadavků, plán testování.	Formal Technical Review Archive
Formal Inspection Forms komentář: formuláře zamýšlené pro podporu procesu formálních inspekcí, které dává k dispozici SEPO, SPAWAR Systems Center.	SEPO; disk
Testování celkově	
Testování viz sekce č. 3.4	

Tabulka 3-7: Zdroje pro téma validace a verifikace softwaru

3.8 Zajištění jakosti Softwaru

Poznámka:

V jádru této činnosti jde vždy, ale nemusí to tím být nutně omezeno, o zajištění toho, co se naplánovalo v širokém slova smyslu (tj. určilo, předepsalo, naplánovalo atd.) pro činnosti a pracovní produkty. Příklady:

Provádí se činnost testování tak, jak se naplánovala? Má pracovní produkt sw architektura ty náležitosti, které předepisuje příslušný předpis (např. příslušný MIL-STD-498 DID)? Dělají se ty kontroly, které se naplánovaly a způsobem, kterým se naplánovaly? atd. Jaký je přesně záběr SQA (Software Quality Assurance), v tom se jednotlivé „školy" vzájemně liší. Jedni např. zahrnují kontrolu adekvátnosti V&V do SQA, jiní ne atd. (vždy to má určité příčiny). Dodejme, že SQA samozřejmě dohlíží i na V&V. Významným nástrojem SQA jsou nejrůznější typy přezkoumání (může se jim říkat různými slovy audit, certifikace, evaluace atd.). Pozn. přezkoumání jsou stejně tak i jedním z nástrojů V&V.

Zdroje:

Text	Místo
Software Assurance Guidebook, NASA-GB-A201 komentář: velmi dobrá příručka, která dává pěkný přehled, co tvoří tuto oblast sw inženýrství. Pozn. tato příručka patří do školy, která činnost Software Assurance pojímá široce, což z praktického hlediska je nepodstatné, protože jestli se přezkoumání návrhu řadí do assurance či V&V je celkem jedno, podstatné je, že se dělat musí (pozn. naplnění této sekce a sekce věnující se V&V odpovídá spíše užšímu přístupu použitém v MIL-STD-498, NASA příručce Software Management Guidebook, tj. SQA odpovídá za existenci naplánovaných činností, produktů a za provedení naplánovaných testů a přezkoumání (pokud se vše, co se má hodnotit předepíše jako v MIL, pak lze mít užší pohled na SQA, jinak se musí do SQA zahrnout hodnocení adekvátnosti naplánovaných přezkoumání atd.), pro tuto kapitolu jsou tyto diskuse nepodstatné, tato kapitola má zpřístupnit důležité a pokud možno praktické informace). Na této příručce je dále cenné, že říká *co* a *kdy*. Tato příručka je vydána k stejnojmennému standardu, který je rovněž k dispozici v SATC – NASA.	SATC – NASA; disk
Software Quality Assurance (část 2, kapitola 5, ESA PSS-05-0) komentář: tato kapitola standardu ESA pro sw inženýrství obsahuje základní vysvětlení konceptů SQA a základní nároky. Dále obsahuje popis co se má a kdy v průběhu projektu vzhledem k této činnosti provádět.	Standard ESA PSS-05-0
SEI Curriculum Modules k tomuto tématu: - Assurance of Software Quality, Curriculum Module SEI-CM-7-1.1 komentář: viz sekce č. 3.11.	SEI; disk
Software Engineering Process Office, SPAWAR System Center vybrané materiály k tomuto tématu: - SSC San Diego Software Quality Assurance Policy - Software Quality Assurance Process - Software Quality Assurance Plan komentář: toto jsou materiály pro podporu klíčové oblasti Software Quality Assurance v úrovni 2 (level 2) v rámci CMM (Capability Maturity Model). Dále komentář viz příloha č. 2.	SEPO; disk
Software QA and Testing Resource Center komentář: zdroj materiálů a odkazů k tématu SQA. Dále viz příloha č. 2.	viz příloha č. 2
Články v CrossTalk k tomuto tématu: Následuje seznam vybraných článků k tomuto tématu publikovaných v CrossTalku: - National Software Quality Experiment: A Lesson in Measurement: 1992 - 1997 (December 1998) - Effective Software Defect Tracking: Reducing Project Costs and Enhancing Quality (April 1999)	STSC /CrossTalk
Software Metrics and Reliability, 9th International Symposium on Software Reliability Engineering komentář: text pojednává o tom co principielně měřit v souvislosti se spolehlivostí softwaru.	SATC – NASA (stránka: Project Support & Outreach)
A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality, 8th Annual Software Technology Conference komentář: text se zabývá principielní otázkou co měřit a jak to využít pokud nás zajímá identifikování rizik projektu a hodnocení jakosti softwaru.	SATC – NASA (stránka: Project Support & Outreach)
checklists	
Seznamy kontrolních otázek (checklists) od fy Construx komentář: firma Construx dává k tomuto tématu k dispozici následující seznamy kontrolních otázek: - Quality-Assurance Program - Effective Inspections - Test Cases	Construx

Tabulka 3-8: Zdroje pro téma zajištění jakosti softwaru

3.9 Vedení softwarového projektu

Zdroje:

Text	Místo
Celkově	
Celkové pokrytí tématu komentář: následují texty, které různě detailně a s různým zaměřením a ve svém souhrnu nezúženě pojednávají činnost vedení sw projektu:	
- *ESA Software Engineering Standards, ESA PSS-05-0*	Standard ESA PSS-05-0; disk
- Guidelines for Successful Acquisition and Management of Software-Intensive Systems	STSC; disk
- *Software Management Guidebook, NASA-GB-001-96*	NASA – SWG; disk

- *Manager's Handbook for Software Development, Rev. 1, SEL-84-101* - Software Management for Executives Guidebook - Poznámky k vedení projektu I, část G, RSI Poznámky: (i) Komentář k textům je uveden v sekci č. 3.11. (ii) Tři kurzívou označené texty ve své kombinaci představují celkem stručný a přitom relativně úplný a prakticky orientovaný „návod" jak vést projekty a přitom na nic podstatného nezapomenout (SMG NASA celková moderní koncepce, ESA dobré popisy jednotlivých činností + rozpracování postupného plánování). (iii) MIL-STD-498 mimo jiné obsahuje mnoho užitečných konceptů pro vedení, jako celek má samozřejmě širší záměr stejně jako ESA PSS-05-0. (iv) Poslední uvedený text umožňuje vidět, jaký je záběr a souvislosti, když se na činnost vedení sw projektu díváme nezúženě; toto lze vidět už na jeho obsahu; rozpracované partie týkající se základních koncepčních strategií vedení rozebírají např. takové věci jako souvislost architektury, modelu SDLC/ postupu vývoje a základní metody vedení atd. Text obsahuje mnoho komentovaných zdrojů k nejrůznějším záležitostem činnosti vedení sw projektu.	NASA – SEL; disk SEPO – SPAWAR disk
SEI Curriculum Modules k tomuto tématu: - Software Project Management, Curriculum Module SEI-CM-21-1.0 - Models of Software Evolution: Life Cycle and Process, Curriculum Module SEI-CM-10-1.0 - Software Metrics, Curriculum Module SEI-CM-12-1.1 - Software Specification: A Framework, Curriculum Module SEI-CM-11-2.1 komentář: viz sekce č. 3.11.	SEI; disk
A Guide to the Project Management Body of Knowledge komentář: podrobný soustavný popis problematiky vedení projektů. Velmi cenný zdroj.	PMI; disk
Články k Software Management od Process Impact komentář: firma Process Impact dává k dispozici řadu stručných prakticky orientovaných článků k různým aspektům vedení sw projektu. Seznam článků k dispozici (v závorce je původní místo publikace): - Know Your Enemy: Software Risk Management (*Software Development*, vol. 6, no. 10) - A Project Management Primer (*Software Development*, vol. 6, no. 6) - Standing on Principles (*Journal of the Quality Assurance Institute*, vol. 11, no. 3) - Recognizing Achievements Great and Small (*American Programmer*, vol. 10, no. 5) - Creating a Software Engineering Culture (*Software Development*, vol. 2, no. 7) - Effective Quality Practices in a Small Software Group (*The Software QA Quarterly*, vol. 1, no. 2) - Implementing Software Engineering in a Small Software Group (*Computer Language*, vol. 10)	Process Impact
Články v CrossTalk k tomuto tématu: Následuje seznam vybraných článků (nejsou to ani zdaleka všechny) k tomuto tématu publikovaných v CrossTalku: - An Introduction to Function Point Analysis (November 1995) - Measurement – The Road Less Traveled (April 1996) - Requirements-Driven Management: A Planning Language (June 1997) - Metric-Based Scheduling and Management (July 1997) - Metrics: Problem Solved? (December 1997) - Earned Value Project Management: A Powerful Tool for Software Projects (July 1998) - Measurement 101 (August 1998) - Managing (the Size of) Your Projects: A Project Management Look at Function Points (February 1999) - Applying Management Reserve to Software Project Management (March 1999) - Gaining Confidence in Using Return on Investment and Earned Value (April 1999) - Rules a Program Manager Can Live By: Getting Back to the Basics (July 1998) - Earned Value Project Management... an Introduction: Once upon a time, a young man journeyed through project management (July 1999) - Something from Nothing: An Overview of the Project Management Body of Knowledge Model (July 1999) - Measurement with a Focus: Goal-Driven Software Measurement (September 1998)	STSC /CrossTalk
Software Engineering Process Office, SPAWAR System Center vybrané materiály k tomuto tématu: <u>Software Project Planning</u>: - SCC San Diego Software Project Planning Policy - Software Project Planning Process - Software Size, Cost, and Schedule Estimation Process - Software Development Plan (SDP) Template - Risk Management Process <u>Software Project Tracking and Oversight</u>: - Software Project Tracking and Oversight Process - Appendix A – Sample Software Measurement Plan - Attachment A – Sample Microsoft Project Plan (MS Project 98) - Attachment B – Sample Monthly Actual Costs Spreadsheet (MS Excel 97) - Attachment C – Sample Project Tracking Spreadsheet (MS Excel 97) - Attachment D – Sample Staff Hour Metrics Forms (MS Excel 97) - Attachment D – Sample Production Engineering Staff Hour Metrics Form (MS Excel 97) - Attachment E – Sample Status Data Collection Forms - Attachment F – Sample Planning Data Collection Forms - Attachment G – Sample Quarterly Review Requirements - Appendix B – An Earned Value Overview	SEPO; disk

- Software Management for Executives Guidebook komentář: toto jsou materiály pro podporu klíčových oblastí Software Project Planning a Software Project Tracking and Oversight v úrovni 2 (level 2) v rámci CMM (Capability Maturity Model). Dále komentář viz příloha č. 2.	
Koncepční strategie vzhledem k postupu vývoje Pozn.: toto téma je poměrně komplikované, ne moc časté, ale naprosto zásadní, protože se vlastně jedná o diskusi o modelech jak vést postup vývoje vlastního sw produktu. Uvedené věci jsou zde spíš pro ilustraci, o co jde. Pokud by o to byl zájem, pak viz text Poznámky k vedení projektu I, kde je tomu věnován značný prostor spolu s uvedením a komentováním odborných zdrojů.	
Anchoring the Software Process, technical report USC/CSE-95-TR-507; také IEEE Software, July 1996 komentář: tato technická zpráva od B. Boehma se vyjadřuje zásadním způsobem k modelu postupu vývoje (též modelu SDLC, model softwarového procesu atd.) Jsou palčivé otázky: Jak postupovat, když *waterfall* nefunguje? Když inkrementy, tak kdy, jaké a vzhledem k čemu? Jaký je vztah mezi architekturou, požadavky, plánem sw projektu, vedením? Jaké zásady platí univerzálně a co může být specifické jednotlivým projektům? atd. Tato zpráva se snaží říci co je invariantní pro každý dobrý postup vývoje sw produktu, co by měl splňovat každý proces vývoje. Zpráva nepředepisuje, žádný konkrétní model SDLC, jde však o konkrétní nároky, které vedením projektu a použitými modely SDLC je třeba splnit. Zpráva představuje jedinečné vyjádření ke koncepční strategii jak vést projekt a čemu mají globálně sloužit konkrétní použité modely SDLC; zpráva staví na důležitosti architektury softwarového produktu a na vyváženém přístupu k požadavkům. Ve zprávě jsou zavedeny tři milníky, které by měl respektovat každý postup vývoje: - Life Cycle Objectives (LCO) - Life Cycle Architecture (LCA) - Initial Operational Capability (IOC) Pozn. UML Process fy Rational převzal tyto zásady.	CSE
Improving Software Economics in the Aerospace and Defense Industry komentář: článek zásadním způsobem nabourává představu o modelu vývoje vodopád, velkých a nákladných milníků typu „detailní design hotov“, představy že všechny požadavky jsou stejně relevantní pro další vývoj atd. Stejně tak nepodporuje domněnku, pokud by ji někdo měl, že modely vývoje, které se módně označují přívlastky „inkrementální, evoluční, iterativní“ a jsou většinou spjaty s OO přístupem, by měly, představovat nějak nedisciplinovaný, vágně definovaný vše umožňující přístup. Zcela jasně deklaruje zásadní důležitost softwarové architektury při vývoji, pro model vývoje a pro vedení projektu. (Pozn. pružný vývoj vzhledem k architektuře je dnes velmi podpořen vlastní modularitou OO prostředků pro programování a design). Názory prezentované v tomto článku stojí bezprostředně u základů dnešního UML Process fy Rational. Tento článek byl autory/ editory mohutné příručky Guidelines for Successful ... dán do přílohy jako vzor jak si udělat v tolika rozličných věcech a často protichůdných nárocích při diskusi o vývoji softwaru jasný koncepční názor.	Guidelines for Successful Acquisition and Management of Software-Intensive Systems, Chapter 15, Addendum A; Rational Software
Architecture-Based Development, SEI-99-TR-007 komentář: tendence brát sw architekturu velmi vážně, zde vyúsťují v technickou zprávu, která celý proces vývoje pojímá v zorném úhlu architektury.	SEI; disk
Process Control for Error-Free Software: A Software Success Story, IEEE Software, May/ June 1999 komentář: článek popisuje úspěšně použitý přístup k postupu vývoje na jednom vojenském projektu. Kontextem je OO přístup, dále model SDLC ve velkém je evoluční a inkrementální. Většinou, když se mluví o modelech vývoje, které se snaží být flexibilní na rozdíl od jasného vodopádu, tak je obtíž překlenout mezeru mezi „děláme to postupně a iterativně a evolučně“ a otázkami „kolik musí být architektury, aby se mohlo začít postupně“, „co je předmětem inkrementu ve velkém a co v malém atd.“, „jak vypadá každodenní praxe jednotlivých vývojářů“ atd. Tento článek po konstatování, že používají evoluční a inkrementální model vývoje ukazuje, jakým procesem vývoje „v malém“ musí projít každá, tzv. položka vývoje (Development Item). Proces vývoje vzhledem k každé položce vývoje je definován konečným automatem (říkají State-Based Process) a přechody mezi jednotlivými stavy jistí přezkoumání nebo testování. Tedy velmi ukázněný proces, kterému Booch říká micro-process. Pozn. těmto koncepčním otázkám jak postupovat při vývoji (model SDLC) ve velkém i v malém, vztahu ke struktuře produktu a k vedení sw projektu se velmi věnuje text Poznámky k vedení projektu I, tam lze najít mnoho komentované literatury, výňatků atd. na toto téma. Článek je primárně o úspěšné aplikaci přezkoumání, inspekcí a měření u projektu s velmi vysokými nároky na kvalitu – žádné chyby – a na čas – velmi rychle. Zde by intuice říkala opustíme zdržující průběžná přezkoumání, průběžná testování, atd. To je ovšem zásadní chyba, ví se to (viz např. pokyny v NASA příručce Software Management Guidebook v tomto směru u činnosti V&V), ale nerespektuje.	IEEE Software; papír
The software process, poznámky k přednášce z kurzu SYDE 221 Software Design komentář: jednoduché a od základů jdoucí vysvětlení, co to je a na co je potřeba nějaký model životního cyklu.	SYDE 221
On the Definition of Software System Architecture, technická zpráva USC/CSE-95-TR-500 komentář: text se také zabývá mnoha rolemi, které sw architektura má v celém životním cyklu a vztahem architektury a vedení sw projektu.	CSE
Rational Unified Process, white paper fy Rational komentář: konkrétní příklad názoru na moderní postup vývoje. Chce flexibilitu, kterou dává OO prostředí, chce stavět na robustní sw architektuře, chce disciplinovaný vývoj atd. Názory na proces vývoje publikované firmou Rational už několik let postupně krystalizují a mění své názvy. Dnešní podoba Rational Unified Process bezprostředně vychází z tezí popsaných v článku „A Rational Development Process“ v *CrossTalk*, July 1996 (od Krutchena z fy Rational), v textu Improving Software Economics in the Aerospace and Defense Industry (od Royce z fy Rational; zdroj viz výše), z názorů Jacobsona (Objectory, kniha Object Oriented Software Engineering atd.; nyní fy Rational), z názorů Booche (např. kniha Object-Oriented Analysis and Design with Applications; fy Rational), z názorů	Rational STSC/ CrossTalk; Rational STSC; Rational

Rumbaugha (OMT, nyní fy Rational) atd. V posledních verzích Rational Unified Process je ukazováno jak dostát nárokům z textu Anchoring the Software Process. Každému doporučuji seznámit se s Rational Unified Process. Pro toho, kdo vyvíjí v OO prostředí, by to měla být nutnost. Kromě technických zpráv (resp. white papers) fy Rational jsou už k dispozici knihy. Firma Rational má technickou zprávu, kde ukazuje mapování mezi Rational Unified Process a nároky CMM do úrovně č. 3 včetně.	CSE Rational
Základní metoda: iniciální plánování -> provádění | monitorování | vyhodnocování | (do/ pře) plánování	
Software Project Management (standard ESA PSS-05-0, část 2, kapitola 2) komentář: tato kapitola standardu ESA pro softwarové inženýrství obsahuje velmi stručně popsanou činnost vedení sw projektu, základní principy a základní nároky na údržbu plánu softwarového projektu.	Standard ESA PSS-05-0
Finishing the Software Plan – Defining the Management Approach (kapitola 6 z NASA-GB-001-96) **Running the Project** (kapitola 7 z NASA-GB-001-96) komentář: tyto kapitoly obsahují velmi stručný popis a vzájemnou souhru základních činností, které jsou nedílnou součástí celkové činnosti vedení sw projektu.	NASA příručka Software Management Guidebook, NASA-GB-001-96
Organizing and Planning (sekce 2 z SEL-84-101, Rev. 1, NASA) **Cost Estimating, Scheduling, and Staffing** (sekce 3 z SEL-84-101, Rev. 1, NASA) **Metrics and Key Management Aids** (sekce 6 z SEL-84-101, Rev. 1, NASA) komentář: sekce 2 obsahuje schematický popis vedení projektu – popis „základní“ metody vedení: zorganizovat projekt, vytvořit plán projektu, vykonat plán projektu. Sekce 3 pak stručně popisuje odhadování, časového plánování a přidělování zdrojů. Sekce 6 pak stručně popisuje co měřit, co sledovat a jak na to reagovat – tedy jak projekt řídit. Přestože je tato příručka stručná, tak je konkrétní, což je velmi cenné a ilustrativní, na druhou stranu je třeba dát pozor s interpretací některých věcí, protože model životního cyklu je uvažován a la vodopád a konkrétní koeficienty u odhadování vychází z daného kontextu v NASA. Přesto je tato příručka velmi užitečná, doporučuje ji i vzdělávací program fy Construx.	NASA příručka Manager's Handbook for Software Development, Software Engineering Laboratory Series, SEL-84-101, Revision 1
Report on Project Management and Software Cost Estimation Technologies, STSC-TR-012-Apr95 komentář: obsahuje stručný popis základní metody vedení projektu (sekce č. 1.3 Concepts and Theory). Obsahuje stručný popis problematiky odhadování (Appendix A: Software Estimation Technology). Obsahuje rozsáhlý popis a klasifikaci konkrétních nástrojů podporujících činnost vedení projektu.	STSC; disk
Software Project Planning Process **Software Project Tracking and Oversight Process** komentář: popis procesu vedení projektu jak jej vyžaduje CMM L2. Stručný taxativní přehled „velkého“ obrázku, co představuje vedení sw projektu.	SEPO; disk
Téma: Měření	
Články k Software Metrics od firmy Process Impact komentář: firma Process Impact dává k dispozici řadu stručných prakticky orientovaných článků k měření. Seznam článků k dispozici (v závorce je původní místo publikace): - A Software Metrics Primer (*Software Development*, vol. 7, no. 7) - Metrics: Ten Traps to Avoid (*Software Development*, vol. 5, no. 10) - Lessons from Software Work Effort Metrics (*Software Development*, vol. 2, no. 10)	Process Impact
Practical Software Measurement: A Foundation for Objective Project Management **Goal Driven Software Measurement – A Guidebook, SEI-96-HB-002** **Software Measurement Guidebook, NASA-GB-001-94** komentář: podrobné příručky, kde je probráno vše, co souvisí s nutným měřením pro vedení sw projektů a produkcí kvalitního výsledného sw produktu. Obsahují: co měřit, jak zjistit co měřit, zavedení měření, definice metrik, definice procedur atd. Velmi cenné. Pro systematické naplánování a úspěšné zavedení měření v organizaci nepostradatelné.	PSM; disk SEI; disk SWG – NASA; disk
Establishing a Software Measurement Process; Technical Report SEI-93-TR-16 komentář: technická zpráva popisuje jak ustanovit a zavést měření jakožto integrální součást celkového procesu vývoje.	SEI
CrossTalk, June 1999 komentář: číslo zaměřené na měření. Další vybrané články v CrossTalk na toto téma: - Measurement – The Road Less Traveled (April 1996) - Metric-Based Scheduling and Management (July 1997) - Metrics: Problem Solved? (December 1997) - Measurement 101 (August 1998) - Measurement with a Focus: Goal-Driven Software Measurement (September 1998)	CrossTalk/ STSC
Developing a Successful Metrics Program, International Conference on Software Engineering 1997 komentář: text se zabývá jak zavést měření (pozn. text se konkrétně zabývá měřením za účelem hodnocení jakosti produktu, hodnocení rizik, pomoci při zkvalitňování procesu a produktu; nicméně princip postupu je aplikovatelný obecně; je popsáno použití Goal/Question/Metrics přístupu).	SATC – NASA (stránka: Project Support & Outreach)
Developing An Effective Metrics Program, European Space Agency Software Assurance Symposium 1996 komentář: text velmi srozumitelně diskutuje a na příkladu ilustruje jak vyvinout dostupný program měření, který vedení projektu pomůže monitorovat rizika projektu a hodnotit jakost produktu. Je použit Goal/Question/Metrics přístup. Ukázaný přístup je použitelný obecně.	SATC – NASA (stránka: Project Support & Outreach)
Developing a Successful Metrics Program, 8th Annual Software Technology Conference, 1996 komentář: podrobněji zpracované téma stejné jako u stejnojmenného článku výše.	SATC – NASA (stránka: Project Support & Outreach)
Applying and Interpreting Object Oriented Metrics, Software Technology Conference 1998 komentář: text se zabývá otázkou co principielně měřit při objektově orientovaném přístupu.	SATC – NASA (stránka: Project Support & Outreach)
Defining and Understanding Software Measurement Data (také publikován na konferenci) komentář: stručný článek popisující principy základního procesu měření, který je potřeba pro podporu vedení a řízení sw projektu. Schematicky popsána minimální sada metrik.	SEI (Featured Articles)

The Goal Question Metric Approach, Encyclopedia of Software Engineering, Wiley 1994 komentář: text od Basiliho *et al* popisující GQM přístup, který pomáhá identifikovat co měřit, známe-li obecně formulované cíle.	ESEG; disk
Téma: Odhadování (Estimating)	
Software Estimation Technology, (Appendix A v STSC-TR-012-Apr95) komentář: příloha A v STSC technologické zprávě věnované vedení projektu obsahuje stručný popis procesu odhadování, stručný přehled metod pro odhadování a stručný popis k podpůrným nástrojům a modelům.	Report on Project Management and Software Cost Estimation Technologies, STSC-TR-012-Apr95, STSC
COCOMO II Model Definition Manual komentář: následník slavného modelu pro odhadování COCOMO ze 70. let. Oba vznikly pod vedením Dr. Boehma. Tento model představuje jednu z možných systematických dobře dokumentovaných cest jak přistoupit k odhadování; odhadování je nutný základ pro plánování času a dalších zdrojů. COCOMO II zohledňuje posun v metodách a postupech vývoje, který nastal za uplynulá léta.	CSE; disk
Estimating With Objects – Part 1 – Part 11 komentář: série 11 stručných článků o odhadování v kontextu objektově orientovaného vývoje. Články jsou od Humphrey, jednoho z lidí, kteří stály u zrodu Software Engineering Institute, iniciátora CMM, autora Personal Software Process atd. (Pozn. články původně vycházely v on-line časopise Object Currents posléze Object Magazine Online.)	SEI (Featured Articles)
Téma: WBS (Work Breakdown Structure)	
Department of Defense Handbook: Work Breakdown Structure, MIL-HDBK-881 komentář: příručka instruující jak vytvářet strukturu dekompozice práce pro armádní projekty. Tato příručka je vhodná zvláště pro inspiraci nikoli pro bezprostřední aplikaci (kvůli typu projektů, např. software v rámci palubních systémů lodí, letadel atd.)	Standardy
Téma: monitorování a vyhodnocování postupu projektu	
Checkpoint Restart – Part 1 – Part 2 komentář: dva stručné články vysvětlující důvody pro a samotný koncept Earned Value – jednoduchý a přitom mocný způsob jak spolehlivě sledovat postup projektu.	SEI (Featured Articles)
Články na téma Earned Value a příbuzná v CrossTalk: - Earned Value Project Management: A Powerful Tool for Software Projects (July 1998) - Applying Management Reserve to Software Project Management (March 1999) - Gaining Confidence in Using Return on Investment and Earned Value (April 1999) - Earned Value Project Management... an Introduction: Once upon a time, a young man journeyed through project management (July 1999)	STSC/ CrossTalk
Software Project Tracking and Oversight komentář: dokumenty k tomuto tématu od Software Engineering Process Office viz výše v této tabulce. Koncept Earned Value je pěkně a stručně vysvětlen v Appendix B - An Earned Value Overview.	SEPO; disk
Metrics and Key Management Aids (sekce 6 z SEL-84-101, Rev. 1, NASA) komentář: popis co principielně měřit, jak to vyhodnocovat, co jsou principielní varující signály a jaké činit nápravné akce pro vedení a řízení sw projektu.	Manager's Handbook for Software Development, SEL-84-101, Rev. 1, NASA
Candidate Management Indicators, standard MIL-STD-498, Appendix F komentář: návrh 11 důležitých ukazatelů, které je vhodné sledovat (v konečném důsledku realizováno měřením) pro účely vedení projektu.	Standard MIL-STD-498
Project Control Panel (kapitola 2 z „Best Practices") **Breathalyser Test** (kapitola 3 z „Best Practices") **Quantitative Targets** (kapitola 4 z „Best Practices") komentář: kapitola 2 (Project Control Panel) obsahuje seznam věcí, které je pro vedení a operativní řízení projektu potřeba měřit. Kapitola 3 (Breathalyzer Test) obsahuje seznam otázek pro zjištění stavu projektu – otázky se ptají na zásadní věci vzhledem k vedení sw projektu. Kapitola 4 (Quantitative Targets) obsahuje chtěné hodnoty a varující hodnoty vzhledem k tomu co se má měřit. (Pozn. „Best practices" jsou velmi cenný zdroj, jdou k podstatě, jsou stručné konkrétní a cílené vzhledem k zvládnutí vedení projektu. Nicméně, jako všechno, si v určitých ohledech jistě vyžádají přizpůsobení konkrétní situaci a zkušenostem.)	The Program Manager's Guide to Software Acquisition Best Practices; SPMN
Questions to Determine Project Status/Health (sekce 1 ze Software Management for Executives Guidebook) komentář: seznam otázek pro určení stavu projektu.	Software Management for Executives Guidebook; SEPO - SPAWAR
„Best" praktiky	
Příslušné praktiky z Best Practices in IEEE Software komentář: viz sekce č. 3.11. Zejména se jedná o následující články: - Feasibility Studies; - Dealing With Problem Programmers; - The Art, Science, and Engineering of Software Development; - Achieving Leaner Software; - Tool Support for Project Tracking; - Gauging Software Readiness With Defect Tracking; - Software's Ten Essential; - Annualized Software Delivery; - Classic Mistakes; - Daily Build and Smoke Test; - How to Defend an Unpopular Schedule.	Construx Software
The Program Manager's Guide to Software Acquisition Best Practices komentář: viz sekce č. 3.11. Maximálně cenné, platí i pro vše další od SPMN.	SPMN; disk
Little Yellow Book of Software Management Questions komentář: obsahuje otázky pro každou „Principal Best Practice", které pomáhají určit a pochopit jak na tom s tím projekt/ tým je. Vazba na text „Best practices".	SPMN; disk

The Little Book of Bad Excuses komentář: obsahuje základní chybné výmluvy proč nedělat ty a ty nutné věci. Vazba na text „Best practices“.	SPMN; disk
Project Breathalyzer komentář: obsahuje test stavu projektu. Vazba na a také součást textu „Best Practices“.	SPMN; disk
The Condensed Guide To Software Acquisition Best Practices komentář: Stručný přehled textu „Best Practices“.	SPMN; disk
Industrial-Strength Management Strategies, článek v CrossTalk, August 1996 komentář: článek v CrossTalku vysvětluje ustavení Software Project Management Network (SPMN), dále důvod a genezi vzniku „Best practices“, tj. textu The Program Manager's Guide to Software Acquisition Best Practices. Pozn. tomuto tématu je dále věnované hlavní téma čísla October 1999 časopisu CrossTalk	STSC/ CrossTalk STSC/ CrossTalk
Téma: Zvládání rizik	
Software Risk Management; Technical Report SEI-96-TR-012 komentář: přehled problematiky zvládání nebo řízení rizik při softwarovém projektu.	SEI
Taxonomy-Based Risk Identification; Technical Report SEI-93-TR-6 komentář: popis konkrétní metody na identifikaci rizik. Tuto metodu doporučuje Boehm a vyžaduje NASA.	SEI
Seznamy kontrolních otázek (checklists) od fy Construx komentář: firma Construx dává k tomuto tématu k dispozici následující seznamy kontrolních otázek: - Most Common Schedule Risks - Complete List of Scheduled Risks	Construx
Software Metrics Program for Risk Assessment, (publikováno také na konferenci) komentář: popis programu měření pro podporu hodnocení stavu projektu, rizik a kvality produktu skrze celý životní cyklus.	SATC – NASA (stránka: Project Support & Outreach)
A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality, 8th Annual Software Technology Conference komentář: text se zabývá principielní otázkou co měřit a jak to využít pokud nás zajímá identifikování rizik projektu a hodnocení jakosti softwaru.	SATC – NASA (stránka: Project Support & Outreach)
Continuous Risk Management at NASA, Quality Week San Francisco, 1999 komentář: stručné vysvětlení základů činnosti Risk Management.	SATC – NASA (stránka: Project Support & Outreach)

Tabulka 3-9: Zdroje pro téma vedení softwarového projektu

3.10 Školení a vzdělávání

Zdroje:

Text	Místo
A Software Engineering Body of Knowledge, v 1.0, Technical Report SEI-99-TR-004 komentář: vynikající strukturovaný přehled, co vlastně tvoří softwarové inženýrství. Obsahuje reference na základní literaturu k jednotlivým tématům. Lze použít jako vodítko pro stanovení potřebných znalostí pro jednotlivé role při projektu, resp. pozice v organizaci. Lze použít pro koncipování vzdělávacích programů. Atd. Stručný komentář viz sekce č. 3.11.	SEI
Construx Professional Ladder komentář: přehledný systematický plán na postupné dlouhodobé profesní samovzdělávání v softwarovém inženýrství. Obsahuje knihy, články, standardy a příručky ke všem tématům sw inženýrství a pořadí v jakém je studovat.	Construx Software
Guidelines for Software Engineering Education Version 1.0, Technical Report SEI-99-TR-032 komentář: tento dokument vytvořila pracovní skupina se zaměřením na vzdělávání a školení softwarového inženýrství (Working Group on Software Engineering Education and Training) při Software Engineering Institute. Text obsahuje schematický přehled látky, co tvoří tělo znalostí softwarového inženýrství a vlastní návrh curricula. (Pozn. shodný text, ale mnohem úsporněji formátovaný lze také získat ze stránky „Software Engineering Education“.). Tento text, technická zpráva *A Software Engineering Body of Knowledge* (SEI-99-TR-004) atd. jsou stručné, kvalitní, neocenitelné zdroje pro všechny, kteří mají co do činění s procesem vývoje softwaru, jeho vedením, jeho definicí či změnou.	SEI; disk http://www.sei.cmu.edu/collaborating/ed/workgroup-ed.html http://faculty.db.erau.edu/hilburn/se-educ
SYDE 221 Software Design komentář: poznámky k přednáškám ke kurzu SYDE 221 Software Design na University of Waterloo. K dispozici jsou poznámky k následujícím tématům: 1. Introduction and Historical perspective 2. The software process 3. Programming in the small 4. Software defects 5. Debugging 6. Design and code reviews 7. Problem definition and requirements analysis 8. Modules 9. Abstract Data Types 10. Inheritance and Polymorphism	http://seurat.uwaterloo.ca/sd221/index.html
Training Guidelines: Creating a Training Plan for a Software Organization; SEI-95-TR-007 komentář: poměrně podrobný přehledový popis toho, co by kdo měl umět plus popis postupu jak vzdělávání v organizaci provádět. Technická zpráva obsahuje příklady jak ke vzdělávání přistupují v různých firmách (např. jaké pořádají kurzy atd.).	SEI

Tabulka 3-10.1: Zdroje pro téma školení/ vzdělávání

3.11 Materiály k softwarovému inženýrství jako celku

Následuje výběr materiálů, které se z různých důvodů a různým způsobem vyjadřují k velké části softwarového inženýrství. Cenné na nich je, že se vyjadřují k celku a dávají tak velmi potřebný celkový obrázek. Ze své podstaty jde o materiály velmi koncisní, protože se vyjadřují k ohromnému tématu. To platí i pro příručku vzdušných sil *Guidelines for Successful Acquisition and Management of Software-Intensive Systems*, přestože má cca 2000 stran.

Zdroje:

Text	Místo (viz příloha č. 2)
Vojenský standard MIL-STD-498 Software Development and Documentation komentář: vynikající přehled nároků na sw inženýrství při projektu; zvláště cenné DIDs (Data Item Description) předepisující nároky na pracovní softwarové produkty a některé zásadní plány, hlavně plán softwarového projektu; mnoho moderních konceptů zaručujících, že se vyžadují principielní věci ve svém principu; standard již neplatí protože armáda chce komerční standardy, nicméně jako zdroj vynikající. (Pozn. Článek ISO/IEC 12207 Software Lifecycle Processes, CrossTalk, August 1996 stručně vystihuje moderní koncepty standardu MIL-STD-498 (na pozadí dřívějšího standardu DOD-STD-2167A) a srovnává jej se standardem ISO 12207. V Crosstalku bylo publikováno mnoho dobrých článků o standardech, jejich charakteristikách, používání, srovnání a vzájemném mapování.)	Standardy (stránka pana Kolacki); DISA; disk CrossTalk, August 1996
ESA Software Engineering Standards, ESA PSS-05-0 Issue 2 komentář: standard evropské vesmírné agentury; i když uvádí, že je nezávislý na modelu SDLC a metodách vývoje, tak jde přece jen spíše o klasický přístup; velmi cenné je rozpracování konceptu postupného plánování, tedy co se v jednotlivých stavech rozpracovanosti vyvíjeného sw produktu má plánovat vzhledem k vedení projektu, CM, V&V a SQA. Obsahuje vzory pro pracovní produkty včetně plánů. Obsahuje velmi stručné popisy jednotlivých činností.	http://dxsting.cern.ch/sting/ ESA.txt disk
Standard ISO/IEC 12207 Software life cycle processes komentář: standard popisuje, které všechny činnosti (procesy) tvoří žádanou praxi softwarového inženýrství a její podporu na úrovni organizace. Dále standard v některých případech přímo nárokuje spřažení těchto činností, např. neřekne, jen že existuje činnost návrh a činnost přezkoumání, tento standard vysloveně jako jednu z podčinností návrhu nárokuje přezkoumání.	papír
NASA Software Documentation Standard, NASA-STD-2100-91 komentář: standard obsahuje předpisy/ vzory pro všechny podstatné plány a pracovní produkty, které mají být vytvářeny a udržovány během sw projektu. Jako všechny jiné předpisy pracovních produktů včetně plánů i tento standard je míněn logicky, tj. předepisuje, co je podstatné, nevynucuje nějakou konkrétní fyzickou podobu. Velmi cenné.	SATC - NASA; SWG – NASA; disk
SPICE (předloha standardu ISO/IEC 15504) komentář: pracovní verze standardu pro hodnocení sw procesu a podporu jeho zlepšování. Jeho část 2: A model for process management je vlastně popis co vše tvoří softwarové inženýrství.	ESI; disk
Capability Maturity Model (CMM), Technical Reports SEI-93-TR-25, SEI-93-TR-24 komentář: tento *de facto* standard předepisuje co mají splňovat jednotlivé oblasti softwarového inženýrství a pořadí jejich konsolidace; ve svém souhrnu pak nárokuje velmi vyspělou a konsolidovanou odbornou praxi softwarového inženýrství jak při projektu tak potřebný kontext v organizaci jako celku; jde o jeden ze základních textů v oblasti zkvalitňování sw procesu. Tento text lze použít jako normál pro zásadní nároky na jednotlivé činnosti softwarového inženýrství, jako jsou např. požadavky, plánování projektu, CM, odborná přezkoumání atd. (Pozn. Pokud se někdo vážně zabývá problematikou zkvalitňování softwarového procesu měl by kromě CMM také určitě znát přístup Software Engineering Laboratory popsaný v příručce **Software Process Improvement Guidebook**, SEL-95-102. Přístup SEL je založen na analýze současného stavu a řízené změně identifikovaných nejproblémovějších věcí v daném kontextu. SEL tento přístup používá přes ¼ století v rámci NASA a má s ním mnoho zkušeností. Dále, protože oba přístupy CMM a SEL/NASA jsou náročné a dlouhodobé a protože mnohde je tíživá potřeba udělat s procesem vývoje něco rychleji, tak vznikají iniciativy typu Software Project Management Network, které přicházejí s určitými sadami „nejlepších“ praktik; zde jde o příručku **The Program Manager's Guide to Software Acquisition Best Practices**. Tato iniciativa byla poměrně podrobně představena a vysvětlena v článku *Industrial-Strength Management Strategies* v časopise CrossTalk (pozn. z nejnovější doby je tématu „best practices“ věnováno říjnové číslo CrossTalk z roku 1999). Nakonec poznamenejme, že všechny tři zmíněné přístupy, tedy CMM, SEL/NASA a „best“ practices se vzájemně doplňují – v žádném případě nestojí proti sobě. Tyto uvedené přístupy představují relativně velkorysé podniky. Při zkvalitňování sw procesu lze samozřejmě smysluplně postupovat i skromněji. Praktické příklady, jak něco užitečného zavést nebo zlepšit, jsou popsány, např. v článcích *Overcoming the NAH Syndrome for Inspections Deployment* z Proceedings of ICSE'98, *Integrating Measurement with Improvement: An Action-Oriented Approach Experience Report* z Proceedings of ICSE'97 atd. Mnoho okomentovaných zdrojů k problematice zkvalitňování sw procesu je uvedeno v textu Poznámky k vedení projektu I v RSI.)	SEI SEL – NASA; disk - SPMN; disk - CrossTalk, August 1996 - CrossTalk, October 1999 - Proc. of the International Conference on Software Engineering 1997, 1998; disk RSI/ disk
Guidelines for Successful Acquisition and Management of Software-Intensive Systems komentář: rozsáhlá příručka (cca 2000 stran) je cenná svým opravdu širokým záběrem, obsahuje např. vazby na systémové inženýrství. Většina témat je aspoň trochu rozpracovaných na rozdíl např. od NASA příručky Software Management Guidebook, kde jsou v podstatě jen vyjmenované. Příručka je velmi cenná tím, že nic nezastírá ze složité reality a schválně obsahuje velmi rozdílné pojednání téhož. Obsahuje velmi dobré shrnující články v přílohách, např. k sw architektuře.	STSC; disk

Software Management Guidebook, NASA-GB-001-96 komentář: psáno z pohledu vedoucího projektu; přehled zásadních činností, technik a pracovních produktů; vysvětleno mnoho souvislostí; vazba na MIL—STD-498; moderní, stručné (cca 120 stran), jako základní přehled velmi dobré. Obsahuje velmi stručné popisy jednotlivých činností.	Software Working Group – NASA; disk
Recommended Approach to Software Development, Rev. 3, SEL-81-305 komentář: velmi pečlivě propracovaná a inspirativní příručka popisující vývoj softwaru; klasický přístup; celé koncipované pro model SDLC vodopád s inkrementy od detailního designu dále; mnoho vzorů pro plány a předpisy produktů; počítá z velkými milníky typu CDR (Critical Design Review) což je v modernějších standardech MIL-STD-498, ISO 12207 nahrazováno flexibilnějším konceptem JTR/JMR (Join Technical/ Management Review). Pokud se používá model vývoje „vodopád" s inkrementy až hodně „vzadu" lze čerpat mnoho bezprostředně, jinak nutno přizpůsobit. Obsahuje mnoho věcí *co*, *kdo*, *kdy*, v jaké souvislosti udělá – cenné.	SEL – NASA; disk
Manager's Handbook for Software Development, Rev. 1, SEL-84-101 komentář: tato příručka je velmi stručná, nicméně pokrývá všechny důležité oblasti vedení a doplňuje se s předešlou Recommended Approach. Koncepčně pro ní platí obdobný komentář.	SEL – NASA; disk
Curriculum Moduly komentář: přehledy důležitých oblastí softwarového inženýrství jako jsou design, vedení projektu, inspekce, SQA, paralelní programování atd. Těchto textů je cca 20 a původně měly sloužit jako podklad pro přípravu příslušných univerzitních kurzů. Stejně tak je lze použít pro představu, co konstituuje jednotlivé činnosti sw inženýrství, pro přehled základních konceptů, pro přehled literatury na dané téma.	SEI; disk
Software Management for Executives Guidebook komentář: příručka, která velmi schematicky (doslova „bodově"), ale systematicky a dost široce probírá oblast vedení sw projektu. Věnuje se následujícím okruhům: (i) otázky pro stanovení stavu projektu, (ii) řešení a předcházení vybraných problémů; (iii) stručné přehledy jednotlivých procesů/ činností sw inženýrství; (iv) kontrolní seznamy (checklist) pro jednotlivé fáze projektu; (v) přezkoumání a kontrolní seznamy; (vi) měření. Tato příručka podporuje úroveň 2 CMM, oblast Software Project Tracking and Oversight. Velmi cenné pro rychlý, schematický a dost úplný přehled a jako seznam na co nezapomenout.	SEPO; disk
A Software Engineering Body of Knowledge, v 1.0, Technical Report SEI-99-TR-004 komentář: zpráva obsahuje dekompozici a stručný popis toho, co tvoří disciplinu softwarové inženýrství a podpůrné discipliny. Znalosti jsou členěny do třech úrovní dekompozice: kategorie, oblast, jednotka (unit). Příkladem knowledge unit je návrh architektury. Zpráva obsahuje stručné popisy identifikovaných kategorií, oblastí a units. Zpráva obsahuje základní literaturu. Velmi cenné. Tato zpráva je součástí společného projektu ACM a IEEE CS s názvem Guide to the Software Engineering Body of Knowledge (viz příloha č. 2.). Příloha č. 3 obsahuje strukturu dekompozice softwarového inženýrství podle této zprávy až do úrovně knowledge units (jednotek).	SEI; disk
Guide to the Software Engineering Body of Knowledge komentář: viz příloha č. 2	viz příloha č. 2; disk
CrossTalk: The Journal of Defense Software Engineering komentář: časopis ministerstva obrany pro softwarové inženýrství (The Journal of Defense Software Engineering). Časopis se vyznačuje snahou pomoci dostat základní důležité elementy dobré praxe softwarového inženýrství do skutečné praxe. Obsahuje mnoho přehledových článků o důležitých věcech, mnoho zkušeností atd. Články jsou stručné a většinou obsahují reference na podrobnější zdroje. Crosstalk je měsíčník, je k dispozici on-line. Jednotlivá čísla bývají profilována určitým tématem, např. požadavky, systémové inženýrství, zajištění kvality, vedení projektů, „nejlepší praktiky" atd. K dispozici je on-line archiv několik let dozadu (k dispozici jsou všechna čísla od roku 1994 včetně) spolu s možností vyhledávání. Velmi cenné, je že časopis je celkově prakticky orientován, to ale nelze interpretovat tak, že je zjednodušující nebo se něčím „nepohodlným" nezabývá. Pro jednotlivé roky je k dispozici index článků řazený podle témat (témata v indexu za rok 1998 jsou: Ada, Acquisition Management, BackTALK, Capability Maturity Model, Configuration Management, COTS, Design, Education and Training, Inspections, Integrated CMM, Internet and Intranet, Lessons Learned, Measurement and Metrics, Policy and Management, Process Improvement, Project Management, Quality, Requirements, Software Certification, Standards, System Engineering, Testing, Tools, Use Cases, Visual Software Development, Year 2000).	STSC; http://stsc.hill.af.mil/CrossTalk/crostalk.html; disk (ročníky 1998 a 1999)
Best Practices Column in IEEE Software Magazine komentář: 16 krátkých článků (2 až 3 strany), které vycházely od ledna 1996 do července 1998 v IEEE Software. Články jsou věnovány nejzákladnějším principům a zásadám, hlavně z oblasti programování, návrhu a vedení projektu, které, když nejsou dodržovány, tak to vede ke zbytečným závažným problémům. Velmi cenné, protože stručné, prakticky orientované a týkající se oblastí s velkým dopadem. Velmi cenný zdroj, který má převážně pohled „ze zdola" – doplňuje se s „nejlepšími praktikami" uvedenými níže. Tento text se od ostatních v této tabulce liší tím, že se zaměřuje pouze na vybrané velmi důležité věci. Jednotlivé články: (1) Who Cares About Software Constructions? (2) Missing in Action: Information Hiding, (3) How to Defend an Unpopular Schedule, (4) Daily Build and Smoke Test, (5) Classic Mistakes, (6) Keep It Simple, (7) Annualized Product Delivery, (8) Software's Ten Essential, (9) Gauging Software Readiness With Defect Tracking, (10) The Programmer Writing, (11) Tool Support for Project Tracking, (12) Achieving Leaner Software, (13) The Art, Science, and Engineering of Software Development, (14) Dealing with Problem Programmers, (15) Feasibility Studies, (16) Why You Should Use Routines, Routinely.	Construx Software
The Program Manager's Guide to Software Acquisition Best Practices komentář: iniciativa vedoucích velkých projektů (hlavně pro americkou armádu) identifikovala velmi problémové oblasti a příslušné praktiky, kterými začít při konsolidaci praxe u velkých projektů. Velmi cenný zdroj, který má pohled „ze shora" – doplňuje se s „nejlepšími praktikami" uvedenými výše. Tento text se od ostatních v této tabulce liší tím, že se zaměřuje jen na něco, co z určitého pohledu je to nejdůležitější. Identifikované praktiky jsou následující: Principal Best Practices: (1) Formal Risk Management, (2) Agreement on Interfaces, (3) Formal Inspections, (4) Metrics-Based Scheduling and	SPMN; disk

Management, (5) Binary Quality Gates at the Inch-Pebble Level, (6) Program-wide Visibility of Progress vs. Plan, (7) Defect Tracking Against Quality Targets, (8) Configuration Management, (9) People-Aware Management Accountability; dále Best Practices v následujících oblastech vedení projektu: (1) Risk Management, (2) Planning, (3) Program Visibility, (4) Program Control, (5) Engineering Best Practices and Culture, (6) Process Improvement Best Practices, (7) Solicitation and Contracting. Pozn. slovo Program zde znamená projekt. Základní jádro tvoří Principal Best Practices spolu se sadou veličin, které měřit, sadou varovných indikátorů, sadou testů indikujících stav projektu, sadou nejhorších věcí, kterým předcházet. Podrobnější informace o tomto textu viz Poznámky k vedení projektu I v RSI.	
Internet FAQ Consortium komentář: viz příloha č. 2.	viz příloha č. 2
A Taxonomy of Software Development Methods, Communications of the ACM, November 1994 komentář: článek obsahuje klasifikaci metod používaných pro vývoj softwaru. Dále článek obsahuje historický přehled se jmény autorů a základními koncepty jak byly postupně formulovány (vybraná jména: Dijkstra, Wirth, Myers, Parnas, Hoare, DeMarco, Chen, Jackson, Meyer, Booch, Coad, Yourdon atd.; vybrané koncepty: *levels of abstractions*, *virtual machine*, *stepwise refinement*, *functional decomposition*, *modularization*, *structured design*, *coupling*, *cohesion*, *information hiding*, *structured programming*, *abstract data type*, *structured analysis* atd.). Článek je bohužel hůře čitelný pro lidi, kteří nejsou zvyklí na „akademické“ texty, přesto vřele doporučuji všem, kteří neprošli příslušným vzděláním a výše uvedená jména a koncepty jim nic moc neříkají anebo jim není jasný jejich smysl a vzájemné vztahy.	disk

Tabulka 3-11: Materiály týkající se softwarového inženýrství jako celku

Pozn. Pokud v komentáři bylo použita charakteristika moderní, pak to znamená, že text je nezávislý na modelu SDLC, metodách vývoje a nemíchá činnosti sw inženýrství a stav rozpracovanosti vyvíjeného produktu.

3.12 Vzory dokumentů, předpisy pracovních produktů, checklists atd. volně k dispozici

Předpisy pracovních produktů včetně plánů, tedy nárokování podstatných věcí, DIDs (Data Item Descriptions) ze standardu MIL-STD-498, dále vzory (templates) pracovních produktů včetně plánů ať už se nazývají též DIDs (v NASA Software Documentation Standards, NASA-STD-2100-91) nebo prostě vzory (většinou formou dokumentu) nebo jiné předpisy pro pracovní produkty včetně plánů obsažené ve standardu ESA PSS-05-0, v NASA příručkách Manager's Handbook for Software Development a Recommended Approach to Software Development, dokumentaci SEPO atd. představují velmi cenný zdroj – představují totiž základní principiální nároky na to, co má být produktem určité činnosti asociované s daným pracovním produktem, např. plánování s plánem sw projektu, návrh architektury se sw architekturou atd. Vzory, resp. předpisy pracovních produktů všech typů představují velkou pomoc pro plánování a provádění sw projektu. Stejně tak představují velkou pomoc pro specifikování softwarového procesu organizace. Následuje přehled vzorů dokumentů, předpisů pracovních produktů, seznamů kontrolních otázek (checklists) atd., které jsou volně k dispozici. Pro všechny dále uvedené zdroje bylo dříve v této kapitole uvedeno, kde je lze získat. Tato sekce má sloužit hlavně jako pomůcka pro rychlý přehled a orientaci. Komentáře k uvedeným zdrojům, předpisům, vzorům, checklists atd. jsou na jiných místech v textu. Následující přehled nemusí obsahovat úplně všechny předpisy, vzory atd. uvedené v textu.

3.12.1 Standard MIL-STD-498

MIL-STD-498 Data Item Descriptions		
(plány)		
Plán vývoje softwaru	(Software Development Plan, SDP)	
Plán testování softwaru	(Software Test Plan, STP)	
Plán instalace softwaru	(Software Installation Plan, SIP)	
Plán předání softwaru	(Software Transition Plan, STrP)	
(popisy a specifikace)		
Popis konceptu fungování	(Operational Concept Description, OCD)	
Specifikace systému/subsystému	(System/ Subsystem Specification, SSS)	
Specifikace požadavků na rozhraní	(Interface Requirements Specification, IRS)	
Popis návrhu systému/subsystému	(System/ Subsystem Design Description, SSDD)	
Popis návrhu rozhraní	(Interface Design Description, IDD)	
Specifikace požadavků na software	(Software Requirements Specification, SRS)	
Popis návrhu softwaru	(Software Design Description, SDD)	
Popis návrhu databáze	(Database Design Description, DBDD)	
Popis testování softwaru	(Software Test Description, STD)	
Zpráva o testování softwaru	(Software Test Report, STR)	*(report)*
Specifikace softwarového produktu	(Software Product Specification, SPS)	
Popis verse softwaru	(Software Version Description, SVD)	
(manuály)		
Manuál uživatele softwaru	(Software User Manual, SUM)	
Manuál vstupů/výstupů softwaru	(Software Input/Output manual, SIOM)	

Manuál pro operátora	(Software Center Operator Manual, SCOM)
Manuál pro provozování počítače	(Computer Operation Manual, COM)
Manuál pro programování počítače	(Computer Programming Manual, CPM)
Manuál pro podporu firmwaru	(Firmware Support Manual, FSM)

Tabulka 3-12.1: MIL-STD-498 Data Item Descriptions

Pro představu v tabulce 3-12.2 následuje ukázka z předpisu pro návrh architektury, tedy nároky na pracovní produkt softwarová architektura. Jedná se o přeložený výňatek ze Software Design Description DID, ze standardu MIL-STD-498. Konkrétně jde o sekci 10.2.4 CSCI architectural description (v ukázce bude číslování sekcí začínat vždy číslem 4 aby se předčíslí 10.2 neopakovalo, ve shodě s originálem). Poznámka k ukázce, v standardu MIL-STD-498 se části nadřazeného systému, které jsou realizované softwarem nazývají CSCI (Computer Software Configuration Item).

Software Design Description (SDD) DID, MIL-STD-498

...
4. Návrh architektury CSCI. Tato sekce bude rozdělena na následující paragrafy za účelem popsání návrhu architektury CSCI. Pokud část nebo celý návrh závisí na stavech nebo režimech systému, tak tato závislost má být indikována. Jestliže informace týkající se návrhu spadá do více než jednoho paragrafu, může být presentována jednou a v ostatních paragrafech referována. Konvence návrhu nutné k porozumění návrhu musí být presentovány nebo referovány.

4.1 Komponenty CSCI. Tento paragraf musí:

a. Identifikovat programové jednotky (software units), které tvoří CSCI. Každá programová jednotka musí mít přiřazen jednoznačný identifikátor. Poznámka: Programová jednotka je element v návrhu CSCI; například dekompoziční jednotka první úrovně dekompozice CSCI, komponenta této dekompoziční jednotky, třída, objekt, modul, funkce, rutina nebo databáze. Programové jednotky se mohou vyskytovat na různých úrovních hierarchie a mohou se skládat z dalších programových jednotek. Programová jednotka v návrhu může a nemusí mít 1:1 vztah k entitám v zdrojovém kódu či datech (rutinám, procedurám, databázím, souborům, atd.), které ji implementují nebo s soubory, které obsahují tyto entity. Databáze může být uvažována jak CSCI tak jako programová jednotka. SDD se může odkazovat na programové jednotky jakýmkoli jménem/jmény konsistentními s použitou metodologií návrhu.

b. Ukázat statický (takový jako „složeno s“) vztah/vztahy programových jednotek. Může být prezentováno mnoho vztahů v závislosti na vybrané metodologii návrhu (například v objektově orientovaném návrhu, tento paragraf může presentovat architektury CSCI ze struktur tříd a objektů a také procesů a modulů).

c. Říci účel každé programové jednotky a identifikovat požadavky na CSCI a rozhodnutí celkového návrhu CSCI (CSCI-wide design) přiřazené k této programové jednotce. (Alternativně, přiřazení požadavků může být poskytnuto v 6.a).

d. Identifikovat status/typ každé programové jednotky (takový jako nový vývoj, existující návrh či program k znovupoužití jak je, existující návrh či program k předělání, vyvinutí software pro znovupoužívání, software plánovaný pro build N, atd.) Pro existující návrh nebo program, popis musí poskytnout identifikující informace, jako jméno, verse, reference na dokumentaci, knihovna, atd.

...

4.2 Koncept provádění (Concept of execution). Tento paragraf popíše koncept provádění/běhu mezi programovými jednotkami. To bude zahrnovat diagramy a popis ukazující dynamické vztahy programových jednotek, což znamená, jak budou interagovat v průběhu fungování CSCI, zahrnující, je-li to vhodné, tok předávání řízení, tok dat, STD diagramy, diagramy ukazující časování, priority mezi jednotkami, obsluha přerušení, časové/sekvenční vztahy, obsluha výjimek, souběžný běh, dynamická alokace/dealokace, dynamické tvoření/mazání objektů, procesy, úlohy a další aspekty dynamického chování.

4.3 Návrh rozhraní. Tento paragraf bude rozdělen do následujících pod-paragrafů za účelem popisu charakteristik rozhraní programové jednotky. Bude zahrnovat jak rozhraní mezi programovými jednotkami, tak rozhraní s externími entitami jako jsou systémy, konfigurační položky a uživatelé. Jestliže část nebo vše z těchto informací je obsaženo v Interface Design Description, v sekci 5 SDD nebo jinde, tak tyto zdroje mohou být referovány.

4.3.1 Identifikace rozhraní a diagramy.
...
4.3.x (Unikátní identifikátor rozhraní v rámci projektu).
...

a. Priorita přiřazená rozhraní entitami, které spolu interagují

b. Typ rozhraní (jako přenos dat v reálném čase, ukládání a výběr dat, atd.) ...

c. Charakteristiky jednotlivých datových elementů, které interagující entita(y) poskytne, uloží, pošle, provede přístup, přijme, atd.: takové jako ...

d. Charakteristiky větších celků složených z datových elementů (záznamů, zpráv, souborů, polí, obrazovek, výpisů, atd.), které intergující entita(y) poskytne, uloží, ...

e. Charakteristiky metod komunikace, které interagující entity použijí pro rozhraní ...

f. Charakteristiky protokolů ...

...

Tabulka 3-12.2: Fragment ze Software Design Description (SDD) DID, MIL-STD-498

3.12.2 NASA-STD-2100-91 (NASA Software Documentation Standard)

Základní struktura standardu NASA- STD-2100-91 (vzory dokumentů)

(plán(y) vedení)	
Plán vedení	(NASA-DID-M000 Management Plan)
Plán akvizičních aktivit	(NASA-DID-M100 Acquisition Activities Plan)
Plán vývojových aktivit	(NASA-DID-M200 Development Activities Plan)
Plán školení	(NASA-DID-M210 Training Development Plan)

Plán Sustaining Engineering a provozních aktivit	(NASA-DID-M300 Sustaining Eng. and Oper. Act. Plan)
Zajišťovací plán	(NASA-DID-M400 Assurance Plan)
Plán řízení rizik	(NASA-DID-M500 Risk Management Plan)
Plán konfiguračního řízení	(NASA-DID-M600 Configuration Management Plan)
Plán dodávky a přechodu na nový systém	(NASA-DID-M700 Delivery and Oper. Transition Plan)
(specifikace produktu)	
Specifikace produktu	(NASA-DID-P000 Product Specification)
Koncept	(NASA-DID-P100 Concept)
Požadavky	(NASA-DID-P200 Requirements)
Návrh architektury	(NASA-DID-P300 Architectural Design)
Detailní návrh	(NASA-DID-P400 Detailed Design)
Podpůrný manuál firmware	(NASA-DID-P410 Firmware Support Manual)
Popis verze	(NASA-DID-P500 Version Description)
Uživatelská dokumentace	(NASA-DID-P600 User's Guide)
Manuál provozních postupů	(NASA-DID-P700 Operational Procedures Manual)
(zajišťovací a testovací postupy)	
Zajišťovací a testovací postupy	(NASA-DID-A000 Assurance and Test Procedures)
Zajišťovací postupy	(NASA-DID-A100 Assurance Procedures)
Testovací postupy	(NASA-DID-A200 Test Procedures)
(Management, Engineering, and Assurance Reports)	
Zprávy týkající se vedení, inženýrství a zajišťování	(NASA-DID-R000 Mngt., Eng., and Assurance Reports)
Zpráva o certifikaci	(NASA-DID-R001 Certification Report)
Zpráva o auditu	(NASA-DID-R002 Audit Report)
Zpráva o inspekci	(NASA-DID-R003 Inspection Report)
Zpráva o nesouladu	(NASA-DID-R004 Discrepancy (NRCA) Report)
Technický návrh změny	(NASA-DID-R005 Engineering Change Proposal)
Zpráva o získaných zkušenostech	(NASA-DID-R006 Lessons Learned Report)
Zpráva o stavu a výkonu	(NASA-DID-R007 Performance / Status Reports)
Zpráva o zajišťování	(NASA-DID-R008 Assurance Activity Report)
Zpráva o testování	(NASA-DID-R009 Test Report)
Požadavek o výjimku	(NASA-DID-R010 Waiver / Deviation Request)
Zpráva o přezkoumání	(NASA-DID-R011 Review Report)

Tabulka 3-12.3 Základní struktura standardu NASA-STD-2100-91 (vzory dokumentů)

3.12.3 NASA Manager's Handbook for Software Development

Vzory dokumentů v NASA příručce Manager's Handbook for Software Development	
(Plány)	
Plán vývoje/vedení (řízení) softwaru	(Software development/management plan)
Plány testování	(Test plans)
(Další dokumenty)	
Požadavky a funkční specifikace	(Requirements and functional specifications)
Koncept fungování	(Operations concept document)
Analýza požadavků	(Requirements analysis report)
Předběžný návrh	(Preliminary design report)
Detailní návrh	(Detailed design document)
Uživatelská příručka	(User's guide)
Popis systému	(System description)
Historie vývoje softwaru	(Software development history)
(Materiály k přezkoumáním)	
Přezkoumání požadavků na systém	(SRR Hardcopy Material, System Requirements Review)
Přezkoumání specifikace softwaru	(SSR Hardcopy Material, Software Specification Review)
Přezkoumání předběžného návrhu	(PDR Hardcopy Material, Preliminary Design Review)
Kritické přezkoumání návrhu	(CDR Hardcopy Material, Critical Design Review)
Přezkoumání připravenosti do provozu	(ORR Hardcopy Material, Operational Readiness Review)

Tabulka 3-12.4 Vzory dokumentů v NASA příručce Manager's Handbook for Software Development

Další NASA příručka Recommended Approach to Software Development obsahuje mírně odlišnou, ale v principu stejnou sadu vzorů dokumentů, která zde již není uváděna. Pozn. když se v těchto dvou NASA příručkách mluví o Software Specification Review (SSR), Critical Design Review (CDR) atd. tak jsou tím myšleny „velká" přezkoumání spojená s „velkými" milníky na koncích fáze či etapy ve vývoji typu vodopád. S těmito „velkými" přezkoumáními byla spojena řada formálních náležitostí, které dodavatel musel kontraktorovi dodat. Pokračování projektu v daném stupni rozpracovanosti bylo vázáno na úspěšné absolvování, např. CDR. Používání těchto „velkých" přezkoumání pochází z dřívějších dob z komunity MIL/ DoD a NASA. Protože s tímto přístupem byly velké problémy; *de facto* vynucoval velmi rigidní model postupu vývoje, byly vyžadovány věci, které se musely generovat jen pro účely těchto „velkých" přezkoumání atd., tak moderní standard MIL-STD-498 od tohoto konceptu zcela ustoupil a zavedl místo toho, pro kontraktora pro „vhled" do projektu, pružné Joint Technical/ Management Review. Dále, nevynucuje produkci věcí jen na „prezentaci" pro „velký" milník s „velkým" přezkoumáním, místo toho má koncept DIDs, které vyžadují principielní věci, ale

nevyžadují konkrétní formu pracovních produktů atd. Podobně tak tomu je ve standardu ISO 12207. (Pozn. Tento problém podrobněji viz např. MIL-STD-498 Overview and Tailoring Guidebook, článek ISO/IEC 12207 Software Lifecycle Processes v CrossTalk, August 1996).

3.12.4 Standard ESA PSS-05-0

Vzory dokumentů ve standardu ESA PSS-05-0 jsou poněkud schematické (tede ne tak podrobné) ve srovnání s ostatními zatím uvedenými. Pro svoji strukturu a způsob pokrytí potřeby dokumentovat sw projekt však mohou být v mnoha případech přínosné.

Vzory dokumentů ve standardu ESA PSS-05-0	
(Technické dokumenty)	
Požadavky uživatele	(User Requirements Document, URD)
Požadavky na software	(Software Requirements Document, SRD)
Architektonický návrh	(Architectural Design Document, ADD)
Detailní návrh	(Detailed Design Document, DDD)
Manuál uživatele softwaru	(Software User Manual, SUM)
Předání softwaru	(Software Transfer Document, STD)
Historie projektu	(Project History Document, PHD)
(Plány)	
Plán vedení softwarového projektu	(Software Project Management Plan, SPMP)
Plán konfiguračního řízení softwaru	(Software Configuration Management Plan, SCMP)
Plán verifikace a validace softwaru	(Software Verification and Validation Plan, SVVP)
Plán zajišťování kvality softwaru	(Software Quality Assurance Plan, SQAP)
(Zprávy/hlášení a formuláře)	
Záznam o změně dokumentu	(Document Change Record, DCR)
Status dokumentu	(Document Status Sheet, DSS)
Nesoulad k přezkoumání	(Review Item Discrepancy, RID)
Požadavek na změnu softwaru	(Software Change Request, SCR)
Zpráva o modifikaci softwaru	(Software Modification Error, SMR)
Zpráva o problému softwaru	(Software Problem Report, SPR)
Poznámka k releasu softwaru	(Software Release Note, SRN)

Tabulka 3-12.5 Vzory dokumentů ve standardu ESA PSS-05-0

3.12.5 Checklists

Checklists v Formal Inspection Process, Version 2.2, SEPO – SPAWAR Systems Center San Diego

Z komentáře v textu: Checklists jsou určeny pro členy inspekčních týmů. Checklists (1) poskytují pokrytí širokého spektra pracovních produktů, (2) jsou navrženy tak, aby pomohly inspektorům při detekci a klasifikaci potenciálních defektů v pracovních produktech a (3) mohou být přizpůsobeny potřebám jednotlivých projektů a typům pracovních produktů.

- Focus Area Checklist for Software Development Plan
- Focus Area Checklist for System Requirements
- Focus Area Checklist for Software Requirements
- Focus Area Checklist for Software Preliminary Design
- Focus Area Checklist for Software Detailed Design
- Focus Area Checklist for Fortran Source Code
- Focus Area Checklist for C Source Code
- Focus Area Checklist for Ada Source Code
- Focus Area Checklist for Test Plan
- Focus Area Checklist for Test Cases and Procedures
- Focus Area Checklist for Software User Documentation

Tabulka 3-12.6 Checklists v Formal Inspection Process

Checklists v Software Formal Inspections Guidebook, NASA-GB-A302

Následující checklists jsou poskytnuty pro ilustraci a jako vodítko pro přípravu přizpůsobených checklists pro každý typ inspekce u daného projektu. Seznamy kontrolních otázek, které mají v závorce JPL pochází z Jet Propulsion Laboratory Software Product Assurance. Checklists mají být vnímány jako výchozí bod při zkoumání. Inspektor by měl zajistit, že uvedené položky na seznamu jsou v pořádku, ale také by měl použít úsudek a zkušenost pro hledání dalších možných nedostatků. Také mohou být vhodné některé otázky z předchozích inspekcí. Jak organizace nebo projekt získává zkušenosti v provádění formálních inspekcí, tak checklists by se měly rozšiřovat a modifikovat; měly by být vytvářeny revize a přídavky.

Sample Checklists:

- Architecture Design Checklist
- Detailed Design Checklist
- Code Inspection Checklist C
- I0 – Architecture Design Checklist (JPL)
- I1 – Detailed Design Checklist (JPL)

- I2 – Code Inspection Checklist C (JPL)
- I2 – Code Inspection Checklist FORTRAN (JPL)
- IT1 – Test Plan Checklist (JPL)
- IT2 – Test Procedure and Function Checklist (JPL)
- R0 – Functional Design Checklist (JPL)
- R1 – Software Requirements Checklist (JPL)
- SY – Functional Requirements Checklist (JPL)

Tabulka 3-12.7 Checklists v Software Formal Inspections Guidebook

Software Development Checklists firmy Construx	
Management	- Loops
- Most Common Schedule Risks	- Unusual Control Structures
- Complete List of Schedule Risk	- Control-Structures Issues
Requirements	- Layout
- Requirements	- Self-Documenting Code
Design	- Good Commenting Technique
- Architecture	- Configuration Management
- High-Level Design	- Debugging
Construction	- Incremental Integration Strategy
- Constructing a Routine	- Evolutionary Delivery
- High-Quality Routines	- Code Changes
- High-Quality Modules	Quality Assurance
- Data Creation	- Quality-Assurance Program
- Naming Data	- Effective Inspections
- General Consideration in Using Data	- Test Cases
- Fundamental Data	Outsourcing
- Organizing Straight-Line Code	- Vendor Selection
- Conditionals	- Vendor Contract Considerations

Tabulka 3-12.8 Software Development Checklists firmy Construx

Další checklists

Process Goodies fy Process Impact

Requirements Engineering
- Requirements Change Impact Analysis Checklist and Worksheet
- Inspection Checklist for Software Requirements Specification
- Inspection Checklist for Use Case Document

Recommended Approach to Software Development, SEL-81-101, NASA
- Checklist for a Unit Design Inspection
- Sample Checklist for Code Inspection

Tabulka 3-12.9 Další checklists

3.12.6 Další

Dále jsou uvedeny některé další zdroje obsahující zajímavé vzory nebo předpisy pro pracovní produkty:
- SPICE: Part 5, Annex D Work product characteristics table obsahuje stručnou charakteristiku 109 typů pracovních produktů včetně plánů, zpráv atd. (Pozn. přílohy B, resp. C obsahují mapování mezi procesy (SPICE identifikuje 35 procesů), resp. praktikami (SPICE identifikuje cca 200 praktik). a výše zmíněnými pracovními produkty.)
- Software Engineering Process Office: Nabízí řadu vzorů pro různé dokumenty, některé viz tabulka č. 3-9. Dokument Software Management for Executives Guidebook obsahuje mnoho seznamů kontrolních otázek (checklists).
- Software Productivity Center: Nabízí vzor plánu konfiguračního řízení a některé další věci. (Pozn. jinak mají celou sadu vzorů dokumentů pokrývající potřeby sw projektu, ale není zadarmo.)
- Construx Software: Nabízí vzory některých pracovních produktů. (Pozn. v době psaní tohoto textu jich bylo 9, ale má jich být postupně více. Vzor pro Software Design Specification je vynikající.)
- Process Impact: Nabízí vzory některých pracovních produktů.
- Configuration Management (CM) Plans: The Beginning to Your CM Solution: Tato technická zpráva ze Software Engineering Institute obsahuje vzor pro plán konfiguračního řízení.

4 Příloha č. 1: Počáteční předpis softwarového procesu v tabulce

Podstatná praktika – *znění*	Konkrétní nárok – *konkrétní nároky k počátečním cílům*	Zdroje (viz kapitola č. 3)
Číslo 1: Požadavky *Je nutno co nejlépe zjišťovat, formulovat, zaznamenávat a udržovat požadavky uživatele anebo zákazníka. Tyto se týkají (i) požadovaného software, příp. systému a software a (ii) způsobu, průběhu, časování a dalších náležitostí softwarového projektu a dodávky vyvíjeného softwarového produktu.*	1. Vzor pro dokument „Specifikace požadavků zákazníka“ musí být přímo odvozen z příslušného předpisu (tzv. DID, Data Item Description) pro tento pracovní produkt v standardu MIL-STD-498 (konkrétně se jedná o Software Requirements Specification (SRS DID), příp. System/ Subsystem Specification (SSS DID)) nebo v standardu NASA-STD-2100-91 (konkrétně se jedná o NASA-DID-P200 Requirements). Pro inspiraci, co se týká struktury a obsahu, mohou dále sloužit předpisy obsažené v standardu ESA PSS-05-0, NASA příručkách Manager's Handbook/ Recommended Approach, materiálech SEPO (mají vzor (template) na základě příslušného DID z MIL-STD-498) a v knize Software Requirements: A Pragmatic Approach. Část vzoru týkající se externích rozhraní musí odpovídat nárokům Interface Requirements Specification (IRS DID) ze standardu MIL-STD-498. (Pozn.: dále je možné a vhodné se inspirovat seznamy kontrolních otázek – checklists – pro tento typ pracovního produktu; viz kapitola č. 3.) 2. Psaní, formulace a vnitřní struktura zápisu jednotlivých požadavků musí odpovídat zásadám, které jsou popsány a vysvětleny v článku „Writing Effective Natural Language Requirements Specifications“ uveřejněném v časopisu CrossTalk. Dále je třeba znát základní přehled problematiky požadavků popsaný v článku „Making Requirements Management for You“ z časopisu CrossTalk. Přídavný nárok pro případ existence nadřazeného systému: 3. Znění shodné s tímto nárokem u softwarové architektury v následující sekci č. 2.2.	- MIL-STD-498 - NASA-STD-2100-91 - ESA PSS-05-0 - Recommended Approach to Software Development, NASA - Manager's Handbook for Software Development, NASA - Software Requirements Specification (SRS) Template, SEPO - Software Rquirements: A Pragmatic Approach, Process Impact - CrossTalk, February 1999 - CrossTalk, April 1999 - MIL-STD-498
Číslo 2: Softwarová architektura *Je nutno vědomě nakládat s architekturou, zjednodušeně řečeno strukturou, vyvíjeného softwarového produktu. Při (novém) vývoji to znamená architekturu navrhnout, dokumentovat a ověřit, že splňuje nároky na softwarový produkt, resp. softwarový produkt a nadřazený systém. Při (novém) vývoji to dále znamená stanovit jak se určité momenty na úrovni návrhu - např. komunikace mezi subsystémy, moduly, vyvolání služeb, atd. - budou konkrétně realizovat a toto dokumentovat. Pro detailní design a programování to po celý život daného sw produktu, tj. při jeho vývoji, rozvoji či údržbě, znamená navrženou nebo už existující architekturu striktně respektovat a striktně respektovat stanovené momenty na úrovni návrhu. Dále při návrhu architektury je třeba dodržovat základní zásady dobrého návrhu (blíže viz podstatná praktika č. 3).*	1. Pracovní produkt softwarová architektura musí splňovat nároky, které stanovuje Software Design Description (SDD DID), konkrétně se jedná o část CSCI architectural design a není-li zpracována jinde i o část CSCI-wide design decisions. (Pozn. CSCI, Computer Software Configuration Item představuje ty dekompoziční jednotky nadřazeného systému, které jsou realizovány softwarem tedy softwarovým produktem.) Návrh rozhraní musí odpovídat předpisu Interface Design Description (IDD DID). Návrh databáze, je-li v systému, musí odpovídat předpisu Database Design Description (DBDD DID). Vždy se jedná o DID ze standardu MIL-STD-498. Dále je vhodné se inspirovat předpisy tohoto pracovního produktu ve standardu NASA-STD-2100-91 (konkrétně se jedná o NASA-DID-P300 Architectural Design), standardu ESA PSS-05-0, NASA příručkách Manager's Handbook/ Recommended Approach. Dále je nutno znát vzor pro specifikaci návrhu od fy Construx (Software Design Specification) a tento respektovat. Návrh softwarové architektury musí obsahovat stanovení jednotlivých momentů na úrovni návrhu. Ten, kdo navrhuje nebo přezkoumává softwarovou architekturu, musí mít o této problematice obecné znalosti aspoň na úrovni odpovídající znalosti následujících textů: The 4+1 View Model of Architecture (Krutchen), An Introduction to Software Architecture (Garlan&Shaw), Software Architecture: An Executive Overview (Clements), On the Definition of Software System Architecture (Boehm). (Pozn.: dále je možné a vhodné se inspirovat seznamy kontrolních otázek – checklists – pro tento typ pracovního produktu; viz kapitola č. 3.) 2. Výstup práce, tj. příslušné pracovní produkty, z činností (návrh), detailní návrh a programování musí být průběžně sledován pomocí odborných přezkoumání. Jedním z cílů těchto odborných musí být zjišťování, zda nedochází k nerespektování navržené, resp. existující softwarové architektury a zda nedochází k nerespektování stanovených momentů na úrovni návrhu (např. že parametry se v té a té situaci předávají tak a tak). Tento konkrétní nárok musí být obsažen v používaných checklists. Konkrétní nároky na podstatnou praktiku odborná přezkoumání jsou níže v sekci č. 2.6. 3. Návrh softwarové architektury musí odpovídat těm zásadám dobrého návrhu, které mají pro tuto úroveň návrhu smysl. Samotné zásady dobrého návrhu jsou předmětem další podstatné praktiky a příslušných konkrétních nároků, viz sekce č. 2.3. Přídavný nárok pro případ existence nadřazeného systému: 4. Je-li vyvíjený software součástí většího systému (tj. některé části systému jsou realizovány softwarem) potom je třeba, aby se tým odpovědný za software účastnil aktivit na úrovni vývoje nadřazeného systému. Konkrétně se jedná o aktivity specifikace systému a návrh systému. Výstupem z těchto aktivit musí být příslušné pracovní produkty, které odpovídají následujícím nárokům na tyto produkty: Operational Concept Description (DID OCD), System/ Subsystem Specification (DID SSS), System/ Subsystem Design Description (DID SSDD). Vždy se jedná o DID ze standardu MIL-STD-498. V případě existence nadřazeného systému pak jeho specifikace a návrh tvoří zásadní vstup pro klasicky chápanou	- MIL-STD-498 - NASA-STD-2100-91 - ESA PSS-05-0 - Recommended Approach to Software Development, NASA - Manager's Handbook for Software Development, NASA - Construx - Rational - http://www.cs.cmu.edu; disk - SEI - CSE; disk - viz podstatná praktika č. 6 - viz podstatná praktika č. 3 - MIL-STD-498

		činnost požadavky zákazníka na software a dále pro návrh softwarové architektury. Termín účastnit se znamená plně odpovídat pokud je systém tvořen výhradně softwarem.	
Číslo 3: Návrh, detailní návrh a programování *Při návrhu, detailním návrhu a programování je nutno dodržovat základní zásady dobrého návrhu a programování v malém. Základní zásady návrhu jsou modularita, vhodně volené abstrakce, jasně definovaná rozhraní,* information hiding *(daná dekompoziční jednotka zbytku systému nabízí jen vhodně volené a vhodně abstraktní rozhraní, vše ostatní je její vnitřní věc, o které nikdo nic neví a nepředpokládá, tzn. algoritmy, data, reprezentace dat atd. jsou lokální; pokud není možno fyzicky, tak logicky). Pokud je nutné použití globálních dat, tak přístup k nim musí být pouze pomocí striktně definovaných služeb nebo striktně definovaným způsobem (to pro případ nutných ohledů na výkon) a nijak jinak! Základní zásady programování v malém jsou správné používání konceptu podprogramů (lokalita dat atd.), strukturované programování, zdrojový kód splňující náležitosti ohledně jmenných konvencí, vzhledu, počtu vnoření, čitelnosti atd. (prostě minimální programovací standardy).*	1.	Při činnosti návrh a detailní návrh je nutno dodržovat principy a zásady určující, co to je z technického hlediska správný a dobrý návrh. Konkrétně se jedná o principy a zásady týkající se dekompozice/ modularity, způsobu volení abstrakcí, rozhraní a tzv. *information hiding*. Ve svém souhrnu jsou tyto principy a zásady stručně a dobře popsány a vysvětleny v následujících článcích: (i) On the Criteria to be Used in Decomposing Systems into Modules (D.L. Parnas, ACM Classic of the month, článek o tom jak provádět dekompozici; zavedení konceptu *information hiding*); (ii) Why You Should Use Routines ... Routinely (jak správně a všestranně používat koncept podprogramu); (iii) Keep It Simple (stručný přehled všech základních koncepčních zásad pro design a programování); (iv) Missing in Action: Information Hiding (volání po systematickém používání konceptu *information hiding*). (Články ii až iv jsou z Best Practices Column v IEEE Software.) Dále pracovní produkt detailní návrh musí splňovat náležitosti předepsané částí CSCI detailed design v předpisu Software Design Description (SDD DID) ze standardu MIL-STD-498. Co se konečné konkrétní podoby pracovního produktu detailní návrh týká, pak je třeba se dále inspirovat v předpisech, které jsou k dispozici v NASA Software Documentation Standard NASA-STD-2100-91 (konkrétně NASA-DID-P400 Detailed Design), standardu ESA PSS-05-0, NASA příručkách Manager's Handbook/ Recommended Approach. Dále je nutno znát vzor pro specifikaci návrhu od fy Construx (Software Design Specification) a tento respektovat. (Pozn.: dále je možné a vhodné se inspirovat seznamy kontrolních otázek – checklists – pro tento typ pracovního produktu; viz kapitola č. 3.)	- ACM Classic of the month - Construx Software - Construx Software - Construx Software - MIL-STD-498 - NASA-STD-2100-91 - ESA PSS-05-0 - Recommended Approach to Software Development, NASA - Manager's Handbook for Software Development, NASA - Construx
	2.	Při činnosti programování je nutno dodržovat zásady dobrého programování v malém. Tyto jsou stručně uvedeny v příloze B normy 24/97. Dále, protože detailní návrh a programování se často překrývají, tak pro programování, tak, jak to má smysl, platí stejné zásady uvedené o bod výše. Konkrétně se jedná o zásady z článků (i) Why You Should Use Routines ... Routinely; (ii) Keep It Simple; (iii) Who Cares About Software Constructions? Dále je třeba stanovit a používat programovací standardy, tj. nárok na pracovní produkt zdrojový kód, v rozsahu nároků přílohy B normy 24/97; jako inspirace mohou sloužit programovací standardy pro C++ uvedené v sekci č. 3.3. Dále je nutno znát seznamy kontrolních otázek (checklists) pro oblast konstrukce od fy Construx a tyto respektovat. (Pozn.: dále je možné a vhodné se inspirovat seznamy kontrolních otázek – checklists – pro tento typ pracovního produktu; viz kapitola č. 3.)	- norma 24/97 - Construx Software - Construx Software - Construx Software - Formal Technical Review Archive - Construx
Číslo 4: Testování *Je nutno provádět testování na úrovni programové jednotky (software unit - samostatně přeložitelná část programu/ jednotka zadání práce pro kódování), na úrovni integrování (celý sw produkt je postupně skládán z jednotlivých dekompozičních jednotek), na úrovni celého sw produktu (zde je základem kvalifikační testování, které musí být provedeno na straně dodavatele před předáním k akceptačnímu testování na straně zákazníka, dále existuje mnoho různě zaměřených typů testování na úrovni celého systému). Testování softwarového produktu je nutno systematicky postupně a průběžně plánovat a ve správný čas systematicky postupně a průběžně provádět. Schematicky řečeno to znamená: (i) spolu s psaním specifikace požadavků začít plánovat kvalifikační testování na straně dodavatele a začít formulovat akceptační kriteria pro převzetí vyvíjeného sw produktu nebo jeho části, (ii) spolu s dekompozicí vyvíjeného sw produktu, tj. v době návrhu architektury, příp. v době jemnější dekompozice, začít plánovat integrační testování (koncepční podklady musí dodat ti, kteří provádějí dekompozici a určují náležitosti vztahu a komunikace jednotlivých částí sw produktu – platí obecně pro všechny úrovně dekompozice, které se ještě nepovažují za unit ve slova smyslu unit testování), (iii) spolu s detailním návrhem jednotlivých programových jednotek (software unit) je třeba stanovit, jak se mají units testovat, (iv) před daným typem testování musí být jasně stanoveno, co se má testovat (testovací případy) a jak se to má testovat (testovací postupy), (v) vlastní proces testování je dokumentován a prováděn následovně: (a) po naprogramování zadané práce programátorem je prováděno testování na úrovni unit (může být provedeno stejným programátorem), (b) při skládání sw produktu z jednotlivých částí je*	1.	Unit testování, integrační testování a testování na úrovni systému je třeba chápat a z technického hlediska provádět, tak jak je stručně popsáno v sekci 5.3.2.3 Testing, části I a v sekci 4.2.4 Testing části II Standardu ESA PSS-05-0. Principielně to samé, ale jinými slovy je též velmi stručně popsáno v sekcích Software Implementation and Unit Testing, Software Integration and Testing, Software CI Qualification Testing v příručce Software Management Guidebook od NASA. Dále minimálně ten, kdo testování plánuje a je za něj celkově odpovědný, by měl znát minimum typu Little Book of Testing, Volume I: Overview and Best Practices a Little Book of Testing, Volume II: Implementation Technique. V těchto „malých knížkách" jsou stručně popsány principy, činnosti, postupy a vazby na okolí celkového procesu testování.	- ESA PSS-05-0 - Software Management Guide-book, NASA-GB-001-96 - SPMN - SPMN
	2.	Testování na dané úrovni je třeba začít plánovat včas a vzhledem k příslušným vstupům. Schematicky to znamená: testování na úrovni systému (ať už se jedná o kvalifikační, akceptační či jiné) začít plánovat již v době, jakmile jsou k dispozici specifikované požadavky uživatele; integrační testování začít plánovat spolu s prováděním dekompozice systému tj. v době návrhu architektury; unit testování plánovat spolu s tvorbou detailního návrhu. *Co* se má *kdy* plánovat – mluvíme o procesu testování – je stručně popsáno v příslušných částech v sekci 4.4 Evolution of the SVVP Throughout the Life Cycle, část II ve standardu ESA PSS-05-0. Principielně to samé je popsáno v příslušné části příručky Software Assurance Guidebook, NASA-GB-A201. Výsledky plánování procesu testování musí být zaznamenány v plánu testování. Plán testování musí svým obsahem odpovídat specifikaci pro testování ve standardu ESA PSS-05-0, appendix C.10 (anebo podobném, např. NASA Software Documentation Standard). Dále, pro konkrétní formát a strukturu plánu testování je třeba se inspirovat příslušnými předpisy a vzory na tento typ pracovního produktu v standardu MIL-STD-498 (Software Test Plan DID, Software Test Description DID), v standardu NASA-STD-2100-91 (příslušná část NASA-DID-M400 Assurance Plan, NASA-DID-A200 Test Procedures), v NASA příručkách Manager's Handbook/ Recommended Approach. Testování je třeba podle plánů provádět a výsledky zaznamenávat. Zaznamenání ať má podobu zpráv z testování, které odpovídají předpisům na tento pracovní produkt v standardu MIL-STD-498 (Software Test Report DID), NASA-STD-2100-9 (NASA-DID-R009 Test Report). (Pozn.: pro to, co mají plány testování obsahovat, je možné a vhodné se inspirovat seznamy kontrolních otázek – checklists – pro tento typ pracovního	- ESA PSS-05-0 - NASA-GB-A201 - ESA PSS-05-0 - NASA-STD-2100-91 - MIL-STD-498 - NASA-STD-2100-91 - Recommended Approach to Software Development, NASA - Manager's Handbook for Software Development, NASA - MIL-STD-498 - NASA-STD-2100-91

prováděno testování na úrovni integrace, které vrcholí testováním na úrovni celého sw produktu, (c) před předáním sw produktu je prováděno kvalifikační testování. Jedná-li se o rozvoj nebo údržbu existujícího softwarového produktu, pak vše výše uvedené platí pro části přímo zasažené a související s pracemi konanými rozvojem či údržbou.	produktu; viz kapitola č. 3.) Přídavný nárok pro případ existence nadřazeného systému: 3. Je-li vyvíjený software součástí většího systému (tj. některé části systému jsou realizovány softwarem), potom je třeba, aby tým odpovědný za software participoval v aktivitách na úrovni vývoje systému. Konkrétně se jedná o aktivity integrační testování systému (tedy integrace jednotlivých částí systému a příslušné testování na této úrovni) a testování na úrovni celého systému. Termín participovat znamená účastnit se v případě, že systém obsahuje části realizované softwarovými produkty. Termín participovat znamená plně odpovídat pokud je systém tvořen výhradně softwarem.	- MIL-STD-498
Číslo 5: Konfigurační řízení *Je nutno provádět konfigurační řízení (alespoň v minimální podobě).*	1. Činnost konfigurační řízení je třeba chápat a provádět, tak jak je stručně popsána v sekci 3.2 části II Standardu ESA PSS-05-0. Dále minimálně ten kdo konfigurační řízení plánuje a je za něj celkově odpovědný by měl znát minimum typu Little Book of Configuration Management, příp. Software Configuration Management Technologies and Applications (STSC Technology Report). 2. Věci, které nelze určit na začátku projektu anebo věci, s jejichž stanovením je výhodné počkat, musí být v každém případě naplánovány včas. *Co* se má *kdy* naplánovat – mluvíme o procesu konfiguračního řízení – je stručně popsáno v sekci č. 3.4 Evolution of the SCMP Throughout the Life Cycle, část II ve standardu ESA PSS-05-0. Dále činnost konfigurační řízení musí být stanovena a naplánována v plánu konfiguračního řízení. Plán konfiguračního řízení musí svým obsahem odpovídat modelovému plánu uvedeném v technické zprávě Configuration Management (CM) Plans: The Beginning to Your CM Solution ze Software Engineering Institute. Dále pro konkrétní formát a strukturu plánu konfiguračního řízení je vhodné se inspirovat příslušnými předpisy a vzory na tento typ pracovního produktu v standardu ESA PSS-05-0, v příručce MIL-HDBK-61, v materiálu Configuration Management Plan Outline (Software Productivity Center), v NASA dokumentačním standardu NASA-STD-2100-91.	- ESA PSS-05-0 - SPMN - STSC – NASA - ESA PSS-05-0 - SEI - ESA PSS-05-0 - MIL-HDBK-61 - SPC - NASA-STD-2100-91
Číslo 6: Odborná přezkoumání *Pro určitou minimální množinu softwarových pracovních produktů včetně plánů je nutno provádět odborná přezkoumání a následné nápravné akce. Tato přezkoumání a nápravné akce je třeba provést nejpozději předtím, než na základě daného sw pracovního produktu je prováděno podstatné množství práce anebo jsou činěna zásadní rozhodnutí (např. na základě relevantních požadavků zákazníka se navrhuje architektura, na základě plánu testování se testuje, na základě plánu sw projektu probíhá celý projekt, na základě architektury a detailního designu pracují programátoři, odladěný a otestovaný unit kód se integruje, atd.). (Pozn. softwarovým pracovním produktem se myslí cokoli, co vzniká v průběhu projektu vývoje softwaru, tj. specifikace požadavků zákazníka, návrh sw architektury, detailní design, zdrojový kód, plán testování, naměřené údaje, plán sw projektu, plán konfiguračního řízení, hlášení problému atd., lze říkat jen pracovní produkt; je vidět, že mezi pracovní produkty patří i plány, nicméně pro jistotu bude někdy použita formulace sw pracovní produkty včetně plánů). Odborným přezkoumáním pro účely těchto podstatných praktik pro začátek budeme rozumět přezkoumání sw pracovního produktu osobou nebo osobami, které mají dostatečné odborné znalosti za účelem zjištění, zda sw pracovní produkt neobsahuje chyby a problémy (tj. že odpovídá na něj kladeným kriteriím, která jsou několikerého druhu: (a) vyplývají z podstaty věci, tj. kód realizuje detailní design, integrační testování odpovídá dekompozici atd.; (b) vyplývají z obecných nároků na daný pracovní produkt, tj. předpis pracovního produktu, standard pracovního produktu, zásady pro provádění příslušné činnosti produkující daný pracovní produkt atd.; (c) další). V rámci odborných přezkoumání může být případně i doporučení řešení problému.*	1. Činnost odborné přezkoumání je třeba provádět podle stanoveného postupu. Tento postup je třeba stanovit tak, aby na jednu stranu byl v daném kontextu vůbec realizovatelný (prostě nepřehánět to s formálností, počtem rolí atd.), ale zároveň aby na druhou stranu to pořád bylo smysluplné odborné přezkoumání, které efektivně hledá defekty a nesrovnalosti v pracovních produktech. (Pozn. samozřejmě se předpokládá i zajištění vyřešení identifikovaných defektů a nesrovnalostí.) Ti, kdo tento postup stanovují a jsou za odborná přezkoumání zodpovědní, musí znát standardní typy odborných přezkoumání (Walkthrough, Technical Review a Formal Inspections). Tyto standardní základní typy odborných přezkoumání jsou popsány v SEPO příručce Peer Review Process a formální inspekce jsou podrobně popsány v NASA příručce Software Formal Inspection Guidebook, popř. v dokumentu Formal Inspection Process od SEPO. Dále musí znát problémy způsobené snahou zavést nepřiměřeně vyspělý proces odborných přezkoumání do prostředí, které na to není připraveno. Tento problém je probírán v článcích Software Inspections & Technical Reviews: Transcending the Dogma, Formal Technical Reviews Across All Maturities a v článku Software Inspections: How to Diagnose Problems and Improve the Odds of Organizational Acceptance. Jedním z podkladů vstupujících do procesu přezkoumání jsou tzv. kontrolní seznamy (checklists), které sumarizují, co u daného pracovního produktu sledovat. Pro přezkoumávané pracovní produkty je třeba mít předem vypracované checklisty. Ty lze vypracovat na základě sad checklistů obsažených v textech Software Formal Inspections Guidebook od NASA a Formal Inspection Process od SEPO. Sadou checklists, které dává k dispozici firma Construx. Dále se lze inspirovat v materiálech Weiss and Kimbrough Inspections Materials firmy Motorola a materiálem An Abbreviated C++ Code Inspection Checklist. Dále, existuje-li, je třeba pro tvorbu checklistu využít předpis daného pracovního produktu ze standardu MIL-STD-498 (tedy vhodné DID). Dále kontrolní seznam (checklist) musí odpovídat konkrétnímu zvolenému předpisu či vzoru na daný pracovní produkt. 2. Přezkoumání je třeba včas a vhodně plánovat. *Co* se má *kdy* plánovat – mluvíme o procesu odborných přezkoumání – je stručně popsáno v příslušných částech v sekci 4.4 Evolution of the SVVP Throughout the Life Cycle, část II ve standardu ESA PSS-05-0. Principielně to samé je popsáno v NASA příručce Software Formal Inspection Guidebook a v příslušné části NASA příručky Software Assurance Guidebook. Pro začátek je minimální seznam pracovních produktů, které se mají odborně přezkoumávat, stanoven následovně: (a) specifikace požadavků zákazníka, (b) softwarová architektura; (c) detailní návrh; (d) zdrojový kód; (e) plán sw projektu; (f) plán testování; (g) plán konfiguračního řízení.	- SEPO - NASA-GB-A302 - SEPO - Institute for Zero Defect Software - CrossTalk, August 1997 - NASA-GB-A302 - SEPO -Construx - Formal Technical Review Archive - MIL-STD-498 - ESA PSS-05-0 - NASA-GB-A302 - NASA-GB-A201
Číslo 7: Zajištění naplánovaných a předepsaných věcí *Při projektu je nutno průběžně zajišťovat, že každá činnost vyžadovaná plánem projektu anebo jiným způsobem, je prováděna v souladu s plánem*	1. Činnost zajištění naplánovaných a předepsaných věcí (dále už jen zajištění jakosti) je třeba chápat a provádět (včetně zajištění nápravy zjištěných nedostatků) tak, jak je velmi stručně ve svém „štíhlejším“ chápání popsána v sekci 5.2.8 Software Quality Assurance v NASA příručce Software Management Guidebook, NASA-GB-001-96; principielně to samé je požadováno v sekci 5.16 v standardu MIL-STD-498. Dále minimálně ten, kdo zajištění jakosti plánuje a je za	- Software Management Guidebook, NASA-GB-001-96 - MIL-STD-498

projektu a dalšími platnými nároky. Při projektu je nutno průběžně zajišťovat, že každý softwarový pracovní produkt vyžadovaný plánem projektu anebo jiným způsobem existuje a podstoupil testování, přezkoumání (zhodnocení/ evaluaci) a nápravné akce vyžadované plánem projektu a dalšími platnými nároky. Dalšími platnými nároky se rozumí nároky obsažené v předpisu softwarového procesu (nyní tento počáteční předpis, který se časem může rozšiřovat a prohlubovat), nároky požadované v kontraktu se zákazníkem a příp. další nároky. Zhodnocením/ evaluací se rozumí proces rozhodování zda činnost nebo pracovní produkt splňují na ně kladená kriteria (pozn. takto se vyhýbáme diskusím o rozdílech, vhodnosti atd. týkajících se různých typů přezkoumání, inspekcí atd.)		něj celkově zodpovědný, musí znát a vhodně využívat kapitolu č. 5 Software Quality Assurance v části II ve standardu ESA PSS-05-0 a NASA příručku Software Assurance Guidebook, NASA-GB-A201. V případě, že se činnost přezkoumání používá jako prostředek pro činnost zajištění jakosti, pak je samozřejmě třeba respektovat nároky pro odborná přezkoumání (viz podstatná praktika a konkrétní nároky č. 6) (pozn. pro potřeby činnosti zajištění jakosti stačí velmi jednoduchá přezkoumání, ale musí být poctivá a musí mít k dispozici všechny potřebné podklady, které vznikají v průběhu projektu).	- ESA PSS-05-0 - NASA-GB-A201 - praktika č. 6
	2.	Činnost zajištění naplánovaných a předepsaných věcí, tedy zajištění jakosti je třeba včas a vhodně plánovat. *Co* se má *kdy* plánovat – mluvíme o procesu zajištění jakosti – je stručně popsáno v sekci 5.4 Evolution of the SQAP Throughout the Life Cycle, část II ve standardu ESA PSS-05-0. Principielně to samé je popsáno na příslušných místech v NASA příručce Software Assurance Guidebook, NASA-GB-A201. Dále činnost zajištění jakosti musí být stanovena a naplánována v plánu zajištění jakosti (SQA Plan, Assurance Plan atd.). Plán zajištění jakosti musí svým obsahem vycházet ze vzorů pro tento typ pracovního produktu ve standardu ESA PSS-05-0 (konkrétně Software Quality Assurance Plan) a standardu NASA-STD-2100-91 (konkrétně NASA-DID-M400 Assurance Plan). Tyto dva standardy lze též použít pro inspiraci co se týká konkrétního formátu a struktury. Úzkému chápání činnosti zajištění jakosti by odpovídalo zajišťovat činnosti a produkty nárokované tímto textem (konkrétně v kapitolách č. 1 a 2) plus co bude navíc obsaženo v konkrétním plánu sw projektu. Pro začátek toho může být dost. Proto pro začátek je minimální seznam činností a pracovních produktů, které by měly být zajišťovány následující: (a) specifikace požadavků zákazníka – činnost a související produkt, (b) softwarová architektura – činnost a související produkt, (c) detailní návrh – činnost a související produkt, (d) zdrojový kód – činnost a související produkt, (e) vedení projektu – činnost a další související produkty (zejména plán sw projektu), (f) testování – činnost a související produkt (tj. plán testování včetně záznamů o testování), (g) konfigurační řízení – činnost a související produkty (hlavně plán konfiguračního řízení), (h) odborná přezkoumání – činnost a související produkty (hlavně plány pro odborná přezkoumání a záznamy z vykonávání odborných přezkoumání). Pozn. seznam je tedy stejný jako u odborných přezkoumání s tím, že byl nutně rozšířen právě o činnost odborná přezkoumání a s tím související produkty.	- ESA PSS-05-0 - NASA-GB-A201 - ESA PSS-05-0 - NASA-STD-2100-91
	3.	Činnost zajištění naplánovaných a předepsaných věcí, tedy zajištění jakosti je třeba provádět nezávisle. To znamená, že osoby odpovědné za zjišťování, zda se provádí (činnosti), to co se naplánovalo a co je předepsáno, a zjišťující zda vznikne (pracovní produkty) to, co se naplánovalo a co je předepsáno a zda pracovní produkty podstoupí ty kontroly (testování anebo odborná přezkoumání), které se naplánovaly a co jsou předepsány, že tyto osoby nejsou ty, co pracovní produkty vytvořily, nejsou ty, co prováděly činnosti a nejsou ty, co za to odpovídají. Toto nevylučuje účast takových osob na tomto zjišťování.	
Číslo 8: Vedení softwarového projektu *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 doděláváním 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*	1.	Plán softwarového projektu musí být vytvořen podle předem připraveného vzoru pro tento typ pracovního produktu. Použitý vzor plánu softwarového projektu pochopitelně může a má zohledňovat kontext a typ daného projektu (v prostředí daného týmu, organizace atd. může být připraveno několik vzorů vhodných pro různé typy projektů). Je třeba, aby vzor plánu projektu vycházel a svým logickým obsahem tedy záběrem odpovídal standardu NASA Software Documentation Standard, NASA-STD-2100-91 nebo předpisům pro plán projektu v standardu MIL-STD-498 (tj. Software Development Plan DID, Software Test Plan DID, Software Installation Plan DID, Software Transition Plan DID). (Pozn. Software Engineering Process Office dává k dispozici Software Development Plan (SDP) Template na základě příslušného předpisu v MIL-STD-498.) Pro konkrétní strukturování a fyzické členění je třeba se dále inspirovat vzory tohoto pracovního produktu ve standardu ESA PSS-05-0 (tj. Software Project Management Plan, Software Configuration Management Plan, Software Verification and Validation Plan, Software Quality Assurance Plan) a v NASA příručkách Manager's Handbook for Software Development a Recommended Approach to Software Development (tj. Software Development/ Management Plan, Test Plans, resp. Generalized Test Plan Format and Contents). Výsledný plán softwarového projektu může mít různé fyzické členění, tj. určité dílčí plány mohou být fyzicky samostatné dokumenty, např. plán konfiguračního řízení, plán testování, plán zajišťování jakosti atd. Fyzické členění ať zcela vyhovuje daným podmínkám. Je třeba, aby celkové koncipování dokumentace softwarového projektu (tedy plán sw projektu spolu s dokumentací důležitých pracovních produktů – nemusí mít nutně vždy formu klasického dokumentu, např. sw architektura může být v nástroji typu CASE) tvořilo konzistentní celek. Možnosti celkového koncipování dokumentace výborně ukazují standard NASA Software Documentation Standard (NASA-STD-2100-91), standard ESA PSS-05-0 (vzory plánů, produktů a formulářů jsou v přílohách B, C a E), předpisy pracovních produktů v standardu MIL-STD-498 (tj. Data Item Descriptions, tyto předpisy předepisují,	- NASA-STD-2100-91 - MIL-STD-498 - SEPO (Software Engineering Process Office) - ESA PSS-05-0 - Manager's Handbook for Software Development, SEL-84-101, NASA - Recommended Approach to Software Development, SEL-81-305, NASA - NASA-STD-2100-91 - ESA PSS-05-0 - MIL-STD-498

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.*		co je principielní, nejsou to a ani nemají být vzory konvenčních dokumentů, ale tyto lze na základě nich připravit) a do značné míry vzory plánů, sw produktů, formulářů, checklistů atd. obsažené v NASA příručkách Manager's Handbook for Software Development (SEL-84-101) a Recommended Approach to Software Development (SEL-81-305). Je třeba plán projektu udržovat tak, jak je koncepčně popsáno v sekci 7.1.4 Maintaining the Software Plan v NASA příručce Software Management Guidebook (NASA-GB-001-96) (podrobněji viz konkrétní nárok č. 5 dále).	- Manager's Handbook for Software Development, SEL-84-101, NASA - Recommended Approach to Software Development, SEL-81-305, NASA - Software Management Guidebook, NASA-GB-001-96
	2.	Je třeba, aby součástí stanovení technického aspektu činností, které se mají v průběhu projektu vykonávat, byly v plánu softwarového projektu obsaženy předpisy, resp. vzory pro zásadní pracovní produkty (o předpisu má smysl mluvit pokud forma pracovního produktu není konvenční dokument, o vzoru pokud ano). Předpisy, resp. vzory musí vycházet z předpisů, resp. vzorů uvedených v NASA Software Documentation Standard (NASA-STD-2100-91) v standardu MIL-STD-498, v standardu ESA PSS-05-0, v NASA příručkách Manager's Handbook for Software Development a Recommended Approach to Software Development atd. Seznam vzorů a předpisů k dispozici viz sekce č. 3.12. Kromě plánů je třeba mít předpis, resp. vzor minimálně pro následující pracovní produkty: specifikace požadavků zákazníka, sw architektura, detailní návrh, zdrojový kód. Dále pro zaznamenání údajů z testování, odborných přezkoumáních, CM, SQA, atd. Je třeba respektovat to, co bylo vzhledem k pracovním produktům, resp. předpisům požadováno konkrétními nároky (č. 1 až 7) výše.	- NASA-STD-2100-91 - MIL-STD-498 - ESA PSS-05-0 - Manager's Handbook for Software Development, SEL-84-101, NASA - Recommended Approach to Software Development, SEL-81-305, NASA
	3.	Je třeba, aby činnost vedení sw projektu a tím i plánování měly dostatečně široký záběr, aby nebyly chápány úzce. Proto je třeba, aby činnost plánování (nedělitelná součást činnosti vedení sw projektu) měla takový záběr – tedy zabývala se těmi věcmi – který je popsán v NASA příručce Software Management Guidebook, NASA-GB-001-96. Dále je třeba, aby činnosti plánování a vedení sw projektu zajistily nároky popsané v tomto textu. Následuje upozornění na některé činnosti, které jsou pro zdar celého projektu velmi důležité a nebyly samostatně probírány v tomto textu: (i) Uvedení do provozu: Vše co souvisí s uvedením sw produktu do provozu, tj. instalace, přechod ze starého systému atd. Stručný a schematický popis činnosti je např. k dispozici v sekci 5.2.5 Preparing for Software Delivery v NASA příručce Software Management Guidebook a v kapitole The Transfer Phase v standardu ESA PSS-05-0. Vzory a předpisy příslušných pracovních produktů jsou k dispozici např. v NASA Software Documentation Standard (konkrétně Delivery and Operational Transition Plan NASA-DID-M700, Version Description NASA-DID-P500) ve standardu MIL-STD-498 (konkrétně Software Installation Plan SIP DID, Software Transition Plan STrP DID, Software Product Specification SPS DID, Software Version Description SVD DID) atd.	- Software Management Guidebook, NASA-GB-001-96 - tento text - Software Management Guidebook, NASA-GB-001-96 - ESA PSS-05-0 - NASA-STD-2100-91 - MIL-STD-498
	4.	„Základní" metodu vedení je třeba minimálně chápat a provádět tak, jak je popsána v sekci č. 1.3 Concepts and Theory ve zprávě Report on Project Management and Software Cost Estimation Technologies (STSC-TR-012-Apr95). Dále „základní" metoda vedení musí být prováděna tak, aby celkový proces vedení odpovídal popisu v kapitole č. 2 Software Project Management ve standardu ESA PSS-05-0 (v části 2) a v kapitole č. 6 Finishing the Software Plan – Defining the Management Approach a kapitole č. 7 Running the Project v NASA příručce Software Management Guidebook (NASA-GB-001-96). „Základní" metoda vedení musí respektovat, lépe řečeno naplňovat, zvolený a popsaný model(y) životního cyklu pro projekt; v této souvislosti se musí dodržovat principy doporučení uvedené v příloze H Guidance on Ordering Deliverables a v sekci G.6.4 Scheduling the selected activities in each build (v příloze G) ve standardu MIL-STD-498. Je třeba, aby plánování pro potřeby „základní" metody vedení projektu, tedy iniciální tvorba a poté po celou dobu trvání projektu údržba (což může být zdetailnění, zpřesnění, oprava, změna atd.) WBS, sítě aktivit, odhadů a časového plánu vhodně odpovídalo praxi, kterou příručka The Program Manager's Guide to Software Acquisition Best Practices nárokuje v: (i) Principal Best Practices (konkrétně jde o praktiky č. 4 Metrics-based Scheduling and Management č. 5 Binary Quality Gates at the Inch-Pebble Level); (ii) Best Practices (konkrétně jde o praktiku Activity Planning v oblasti Planning); (iii) Breathalyzer Test (konkrétně jde o sekce č. 1 Do you have a current, credible activity network supported by a Work Breakdown Structure (WBS)? č. 2 Do you have a current, credible schedule and budget? č. 5 Do you know your schedule compression percentage? č. 6 What is the estimated size of your software deliverable? How was it derived?). Je třeba, aby sledování postupu a stavu projektu vhodně odpovídalo praxi, kterou příručka The Program Manager's Guide to Software Acquisition Best Practices nárokuje v: (i) Principal Best Practices (konkrétně jde o praktiku č. 6 Program-wide Visibility of Progress vs. Plan); (ii) Best Practices (konkrétně jde o celou oblast Program Visibility). Za účelem co konkrétně je vhodné sledovat, v konečném důsledku měřit, pro činnost vedení sw projektu je třeba se inspirovat v Candidate Management Indicators (Appendix F ve standardu MIL-STD-498), v Metrics and Key Management Aids (sekce 6 v NASA příručce Manager's Handbook for Software Development), v Project	- Report on Project Management and Software Cost Estimation Technologies, STSC-TR-012-Apr95 - ESA PSS-05-0 - Software Management Guidebook, NASA-GB-001-96 - MIL-STD-498 - SPMN (Software Program Managers Network) - SPMN - MIL-STD-498 - Managers's Handbook for Software Development, SEL-84-101, NASA

		Control Panel (kapitola 2 v příručce The Program Manager's Guide to Software Acquisition Best Practices), v článku Defining and Understanding Software Measurement Data (SEI Featured Article), v tabulce 6-4 Required Activities and Related Measures (v NASA příručce Software Management Guidebook). Používaná metoda, jak získat konkrétní metriky z koncepčního záměru, je popsána v textu The Goal Question Metric Approach. Stručný popis, jak dosáhnout rozumného programu měření, je např. v textu Developing a Successful Metrics Program (anebo Developing an Effective Metrics Program). Je třeba, aby osoba odpovědná za měření, tj. jeho vymyšlení a zavedení, znala některý z textů popisujících tuto problematiku podrobněji, jako např. Practical Software Measurement: A Foundation for Objective Project Management. K hodnocení stavu projektu je dále třeba vhodně využívat k tomuto specielně vyvinuté „testy", resp. sady otázek. K dispozici jsou Breathalyzer Test (kapitola 2 v příručce The Program Manager's Guide to Software Acquisition Best Practices), Questions to Determine Project Status/ Health (sekce 1 v Software Management for Executives Guidebook). Jako úplný základ k sledování stavu projektu z pohledu Kolik toho už je hotovo? Kolik to stálo? Jak dlouho to trvalo? Jak to vypadá se vztahem k původním časovým plánům a rozpočtu? atp. je třeba používat mocný a jednoduchý přístup Earned Value. Tento přístup spojuje činnosti plánování, sledování a vyhodnocování. Stručně a jasně je popsán v článcích Earned Value Project Management: an Introduction (CrossTalk, July 1999), Gaining Confidence in Using Return on Investment and Earned Value (CrossTalk, April 1999), Earned Value Project Management: A Power Tool for Software Projects (CrossTalk, July 1998), Checkpoint Restart – Part 1 a Part 2 (SEI Featured Articles). Pozn. praxe popsaná v příručce The Program Manager's Guide to Software Acquisition Best Practices zajišťuje použití přístupu Earned Value (od toho, že Project Control Panel obsahuje příslušná měření, přes definování jasných kriterií hotovo/nehotové u detailních úloh ve WBS atd.). Výsledkem vyhodnocení zjištěných údajů a informací o postupu a stavu projektu můžou být nápravné akce, které se projeví vhodnou údržbou plánů, která může mít mnoho podob. Stručný popis nápravných akcí lze nalézt v Metrics and Key Management Aids (sekce 6 v NASA příručce Manager's Handbook for Software Development), v Troubleshooting and Problem Avoidance (sekce 2 v Software Management for Executives Guidebook). Dále, vodítkem pro to jak mají zhruba vypadat hodnoty získané sledováním mohou být Quantitative Targets (kapitola 4 v příručce The Program Manager's Guide to Software Acquisition Best Practices).	- SPMN - SEI (Software Engineering Institute) - Software Management Guidebook, NASA-GB-001-96 - ESEG (Experimental Software Engineering Group) - SATC – NASA - SATC (Software Assurance Technology Center) – NASA - PSM (Practical Software Measurement) - SPMN - SEPO - CrossTalk, July 1999 - CrossTalk, April 1999 - CrossTalk, July 1998 - SEI - SPMN - Managers's Handbook for Software Development, SEL-84-101, NASA - SEPO - SPMN
	5.	Při vedení sw projektu je třeba plánování provádět průběžně a postupně. Celkově je třeba dodržovat schematicky řečené nároky v sekci 7.1.4 Maintaining the Software Plan, NASA příručka Software Management Guidebook (NASA-GB-001-96). Je třeba, aby přístup k postupnému upřesňování, dopracovávání a průběžné údržbě plánu softwarového projektu koncepčně vycházel z přístupu, který je popsán ve standardu ESA PSS-05-0 (v sekcích 2.4, 3.4, 4.4, 5.4 v části I standardu ESA je mimo jiné schematicky popsáno co má obsahovat plán sw projektu po „fázích" požadavky uživatele, požadavky na software, architektonický návrh, detailní návrh a produkce; v sekcích 2.4, 3.4, 4.4, 5,4 v části II standardu ESA je schematicky popsán vývoj plánu sw projektu skrze životní cyklus; pozn. termín „fáze" je třeba chápat s vědomím příslušného vysvětlení v poznámce (f) v sekci níže věnované upozorněním pro interpretaci). NASA příručka Recommended Approach to Software Development SEL-81-305 také obsahuje stručné popisy co má tým vedení projektu zajišťovat v závislosti na stavu rozpracovanosti vyvíjeného sw produktu. Dále je třeba respektovat případné další nároky *co* se má *kdy* plánovat, které byly popsány pro činnosti testování, konfigurační řízení, odborná přezkoumání a zajištění naplánovaných a předepsaných věcí výše (jde o konkrétní nároky pro podstatné praktiky č. 4, 5, 6 a 7).	- Software Management Guidebook, NASA-GB-001-96 - ESA PSS-05-0 - Recommended Approach to Software Development, SEL-81-305, NASA
	6.	V rámci plánování sw projektu je třeba včas zvolit a popsat model životního cyklu, který bude pro „základní" metodu vedení projektu koncepčním návodem jak postupovat. (Pozn. „včas" znamená jednak v době iniciálního plánování a v případě, že ve skutečnosti je potřeba více jednotlivých modelů životního cyklu, tak „včas" znamená ve chvíli dostatku informací pro rozhodnutí, nejpozději však předtím než je daný model životního cyklu potřeba.) Základní popis problematiky modelů životního cyklu (tj. modelů postupu vývoje) je třeba znát minimálně ze sekce 5.1 Selecting an Appropriate Life-Cycle Model v NASA příručce Software Management Guidebook NASA-GB-001-96 (obsahu též model životního cyklu pro údržbu), z přílohy G Guidance on Program Strategies, Tailoring, and Build Planning ve standardu MIL-STD-498 (pozn. termínem Program Strategy je označován celkový model životního cyklu na nejhrubší úrovni dekompozice, kde jednotlivé dekompoziční jednotky jsou části nadřazeného systému realizované softwarem). Jako základní a dobrý úvod do tématu, nač a proč je vůbec dobrý nějaký model životního cyklu, velmi doporučuji poznámky k přednášce The Software Process z kurzu Software Design (jedná se o kurz SYDE 221 Software Design na University of Waterloo v Kanadě). Dále je potřeba, aby osoby, které vybírají a definují modely životního cyklu,	- Software Management Guidebook, NASA-GB-001-96 - MIL-STD-498 - SYDE 221 Software Design

		znaly minimálně následující texty týkající se principů této problematiky: Models of Software Evolution: Life Cycle and Process (Curriculum Module SEI-CM-10-1.0), článek Improving Software Economics in the Aerospace and Defense Industry, technickou zprávu On the Definition of Software System Architecture (USC/CSE-95-TR-500), technickou zprávu Anchoring the Software Process (USC/CSE-95-TR-507). Zvolený model životního cyklu je třeba vhodně použít jako koncepční návod pro časové plánování a „základní" metodu vedení projektu. Je třeba zabránit, aby rigidní či nevhodné časové plánování zmařilo záměry, pro které byl daný model životního cyklu vybrán. Standard MIL-STD-498 toto nárokuje v příloze H Guidance on Ordering Deliverables a v sekci G.6.4 Scheduling the selected activities in each build (v příloze G). Je třeba, aby celkový postup vývoje softwaru odpovídal nárokům, které jsou uvedené a vysvětlené v technické zprávě Anchoring the Software Process (USC/CSE-95-TR-507). Ukázka sladění principielních nároků stanovených v textu Anchoring the Software Process a flexibility umožněné vlastnostmi OO přístupu je vidět v textu Rational Unified Process (white paper fy Rational k poslední verzi jejího procesu vývoje). Tento text velmi doporučuji znát, v případě OO vývoje je nutno jej znát.	- SEI - STSC (Software Technology Support Center); Rational - CSE (Center for Software Engineering) - CSE - MIL-STD-498 - CSE - Rational
	7.	V rámci vedení sw projektu je třeba provádět činnost, která ošetřuje potenciální rizika (Risk Management). Tj. činnost, která identifikuje, analyzuje, předchází, monitoruje atd. možná rizika. Je třeba, aby tato činnost byla prováděna v tom smyslu jak je maximálně stručně popsáno v sekci 2.2.3 Risk Management v části II ve standardu ESA PSS-05-0 a v článku Continuous Risk Management at NASA (ze SATC, NASA). Ten, kdo činnost řízení rizik plánuje a je za ní odpovědný, by měl znát podrobnější popis této problematiky, např. Software Risk Management, Technical Report SEI-96-TR-012.	- ESA PSS-05-0 - SATC – NASA - SEI
	8.	Je třeba, aby osoby, které výkonně vedou projekt vývoje softwaru, tj. jsou zodpovědné za jeho plánování, provádění a řízení, měly dostatečné odborné znalosti. Doporučené souhrnné materiály jako minimum pro začátek: Software Management Guidebook (NASA-GB-001-96), ESA Software Engineering Standards (ESA PSS-05-0), The Program Manager's Guide to Software Acquisition Best Practices, Manger's Handbook for Software Development (NASA, SEL-84-101, Rev. 1), Software Project Management (Curriculum Module SEI-CM-21-1.0). Dále k tématu vedení projektu obecně doporučuji text A Guide to the Project Management Body of Knowledge (Project Management Institute). Velmi doporučuji průběžně sledovat časopis CrossTalk (The Journal of Defense Software Engineering). CrossTalk vychází jednou za měsíc, je k dispozici elektronicky a zabývá se tématy bezprostředně souvisícími s vedením projektů. Dále, každý by se měl vzdělávat systematicky a dlouhodobě. K tomu může výborně posloužit studijní plán fy Construx pro samostudium a samozřejmě odborné zdroje uvedené v tomto textu. (Pozn. tento bod se týkal konkrétních nároků na odborné znalosti pouze vzhledem k vedení sw projektu, nikoli vzhledem k celé disciplině softwarové inženýrství.)	- NASA-GB-001-96 - ESA PSS-05-0 - SPMN - SEL-84-101 (Software Engineering Laboratory). NASA - SEI - PMI (Project Management Institute) - CrossTalk (STSC – Software Technology Support Center) - Construx
Číslo 9: Mechanismus řešení problémů *Při projektu je nutno mít ustanoven fungující mechanismus řešení problémů. Problémy se týkají vyžadovaných činností a vyžadovaných pracovních produktů. Problém může být identifikován při testování, při odborných přezkoumáváních, při zajišťování naplánovaných a předepsaných věcí, při společných přezkoumáních se zákazníkem atd. Identifikované problémy musí být evidovány. Musí být zahájeno jejich řešení. A musí být vyřešeny.*	1.	Je třeba stanovit způsob klasifikace druhu problému, závažnosti problému a způsob evidence zjištěných problémů. Pro klasifikaci problémů lze vycházet z Category and Priority Classifications for Problem Reporting, příloha C ve standardu MIL-STD-498. Pro evidenci problémů lze vycházet ze vzoru Software Problem Report (standard ESA PSS-05-0, Appendix E) a ze vzoru Discrepancy (NRCA) Report (standard NASA-STD-2100-99, NASA-DID-R004 Discrepancy (NRCA) Report; NRCA – nonconformance reporting and corrective action). Stanovený způsob klasifikace a evidence problémů je třeba při projektu dodržovat.	- MIL-STD-498 - ESA PSS-05-0 - NASA-STD-2100-91
	2.	Je třeba stanovit obecný postup pro řešení problémů. Konkrétně jde o přijetí problému k vyřešení, určení kdo za vyřešení odpovídá, sledování stavu od přijmutí až po vyřešeni, uložení záznamu. Pro stanovení postupu je třeba vhodně vycházet z popisu nároku ve standardu MIL-STD-498 (5.17 Corrective action) anebo ve standardu ISO 12207 (6.8 Problem resolution process). Stanovený postup pro řešení problémů je třeba při projektu dodržovat.	- MIL-STD-498 - ISO 12207

Tabulka 4-1: Shrnutí počátečního předpisu softwarového procesu

5 Příloha č. 2: URL adresy zdrojů odborných textů

Místo + komentář	URL adresa
Standardy komentář: místo, odkud lze získat zásadní standardy (DoD, MIL, NASA, ISO, IEEE, ...); obsahuje referenci na on-line podobu anebo kde jej lze objednat. (Pozn. článek Software Standards: Their Evolution and Current State v CrossTalk, December 1999 stručně popisuje 30. letou historii a současný stav standardů týkajících se vývoje softwaru v americké armádě.)	http://members.home.net/kolacki/critical_standards.htm STSC/ CrossTalk
Defense Information Systems Agency – Center for Information Technology Standards komentář: místo, kde lze získat standardy, specifikace, příručky, atd. vydané americkou armádou. Konkrétní místo: - Standards Document Library – DISA Center for Standards Další místo, přes která lze získat armádní standardy, příručky atd.: - STINET – DoD Index of Specifications & Standards (DODISS)	http://www.itsi.disa.mil http://www-library.itsi.disa.mil http://www.dtic.mil/stinet/htgi/dodiss
Computer Science Bibliography komentář: obsahuje obrovské množství referencí na práce všeho druhu (knihy, články, výzkumné zprávy, Ph.D. Theses, konference, atd.) z computer science.	http://sunsite.ust.hk/dblp/db
Construx Software komentář: server fy Construx Software obsahuje spoustu užitečných věcí, např. články z Best Practices Column jež vycházely v IEEE Software; plán četby odborné literatury pro odborný růst softwarových inženýrů atd. Následují vybrané velmi zajímavé a užitečné věci: - **Software Development Checklists**: vynikající sada seznamů kontrolních otázek (checklists) týkajících se následujících činností: Risk Management, Requirements, Design, Construction, Quality Assurance, Outsourcing. Činnost implementace (konstrukce) je pokryta nebývale bohatě (více než 15 checklists pokrývající mnoho aspektů zdrojového kódu programů). Kontrolní seznamy (checklists) se velmi hodí pro přezkoumání a vůbec udržení vědomí co je dobrá praxe v dané oblasti. - **Document Templates**: je k dispozici několik vzorů důležitých dokumentů (v době psaní tohoto textu bylo k dispozici 9 vzorů nebo předpisů na dokumenty s tím, že výhledově jich má být k dispozici více). K dispozici jsou vzory např. pro Software Design Specification, Software Development Plan, Change-Control Plan atd. Vzor pro Software Design Specification je excelentní. (Pozn. Tato stránka obsahuje odkazy na další místa, která obsahují vzorové dokumenty tohoto typu.) - **Survival Guide Website**: mnoho přídavných informací (kontrolní seznamy, atd.) ke knize *Software Project Survival Guide* (Steve Mc Connell, Microsoft Press, 1998), které i bez této knihy mohou být užitečné. - odkazy na další místa se zdroji k softwarovému inženýrství (**Links to Software Development Resources**) - **16 článků z Best Practices Column**, které vycházely v IEEE Software - plán četby odborné literatury (**Construx Professional Ladder**)	http://www.construx.com home -> Software Development Checklists home -> Document Templates home -> Survival Guide Website home -> Software Engineering Tools -> Links to Other Resources http://www.construx.com/stevemcc/ieeesoftware/bp.htm http://www.construx.com/ladder/home.htm
Process Impact komentář: server fy Process Impact obsahuje mnoho krátkých prakticky orientovaných článků na zásadní témata: vedení projektu, požadavky, přezkoumání, měření, zlepšování procesu; obsahuje mustry pro plány a specifikace produktů (tzv. process goodies); obsahuje několik užitečných kapitol z knihy Software Requirements: A Pragmatic Approach od K. Wiegers, atd.	http://www.processimpact.com
ACM Classic of the month komentář: články od legend computer science a softwarového inženýrství, které kdysi přicházely s koncepty, které jsou dnes základní výbavou naší disciplíny, např. Parnasův článek o kriteriích dekompozice sw systémů do modulů (o konceptu, kterému dnes říkáme *information hiding*), Dijkstrův o go to, atd.	http://www.acm.org/turing/clasics
Software Engineering Laboratory (SEL) – NASA komentář: SEL NASA dává k dispozici velmi mnoho kvalitních manuálů, příruček, atd. Tato laboratoř již několik desítek let usiluje o dobrou odbornou praxi softwarového inženýrství v NASA. Podloženo obrovskými zkušenostmi. Tato laboratoř je společné pracoviště NASA a University of Maryland (Basili).	http://sel.gsfc.nasa.gov
Software Assurance Technology Center (SATC) – NASA komentář: SATC se v rámci NASA zabývá věcmi typu CM, SQA, review, měření, risk management atd. Nabízí velmi kvalitní texty na daná témata.	http://satc.gsfc.nasa.gov
Software Working Group – NASA komentář: jedno místo odkud lze získat nejdůležitější materiály (standardy, příručky, manuály, články, atd.) produkované jednotlivými ústavy typu SEL, SATC, atd.	http://www.ivv.nasa.gov/SWG
Independent Verification & Validation Facility (IVV) – NASA komentář: toto středisko nabízí mnoho textů na široce pojaté téma validace a verifikace.	http://www.ivv.nasa.gov

Experimental Software Engineering Group, University of Maryland (ESEG) komentář: skupina Dr. Basiliho. Tato skupina velmi úzce spolupracuje se Software Engineering Laboratory v NASA, viz výše. Skupina se mimo jiné zabývá analyzováním dosavadních zkušeností a řízeným experimentováním v procesu vývoje. K tomuto je třeba kvalitní a cílený proces měření. To vše za účelem dosáhnout dobré praxe softwarového inženýrství a zkvalitnění softwarového procesu. V této souvislosti aplikují koncepty Quality Improvement Paradigm (QIP), Experience Factory (EF) a Goal/Question/Metric approach (GQM). Zdroj mnoha zajímavých a cenných textů. (Pozn. k těsné spolupráci se SEL NASA, např. přístup k zkvalitňování softwarového procesu vyvinutý a dlouhá léta používaný v SEL NASA je prodchnut výše uvedenými koncepty jako je analýza dosavadních zkušeností, návrh změn v procesu, vyzkoušení navržených změn atd.)	http://www.cs.umd.edu/projects/SoftEng/ESEG
Software Engineering Institute (SEI) komentář: institut softwarového inženýrství při Carnegie Mellon University je jedno z nejelitnějších míst intenzivně se zabývající softwarovým inženýrstvím. Jedná se většinou o velkoryse pojaté projekty týkající se věcí typu zkvalitňování sw procesu, sw architektura, měření, atd. SEI dává k dispozici bezpočet technických zpráv, příruček, článků, výukových materiálů, výsledků práce zájmových skupin, atd. V SEI vznikl např. CMM.	http://www.sei.cmu.edu
Software Engineering Process Office, Space and Naval Warfare System Center - (SEPO – SPAWAR) komentář: SEPO nabízí celou řadu materiálů, které mají podporovat úroveň 3 CMM při projektech v SPAWAR System Center (SEPO je totiž za to zodpovědná). K dispozici jsou zpracování témat, předpisy postupů, vzory plánů a předpisů pracovních produktů, odkazy na místa s dalšími zdroji na Internetu atd. Tyto materiály jsou k dispozici pro mnoho základních činností (tedy pro ty, co korespondují s úrovněmi č. 2 a 3 CMM, plus některé další) typu požadavky, konfigurační řízení, vedení projektu, SQA, přezkoumání atd. Jedná se o velmi cenný zdroj informací. Následuje seznam pro nás zajímavých činností/ oblastí pro, které jsou k dispozici výše zmíněné materiály, zdroje, odkazy atd. (na serveru je k dispozici daleko více věcí): - Requirements Management (L2) - Software Project Planning (L2) - Risk Management (l2) - Software Project Tracking and Oversight (L2) - Software Subcontract Management (L2) - Software Quality Assurance (L2) - Software Configuration Management (L2) - Software Testing (L3) - Peer reviews (L3)	http://sepo.nosc.mil
European Software Institute (ESI) komentář: zdroj cenných informací; např. tento institut zaštiťoval tvorbu SPICE (Software Process Improvement and Capability dEtermination), což je podklad pro normu ISO/IEC 15504.	http://www.esi.es
Project Management Institute (PMI) komentář: dává k dispozici A Guide to the Project Management Body of Knowledge (PMBOK); IEEE přejímá PMBOK do svých standardů pro softwarové inženýrství.	http://www.pmi.org
Practical Software Measurement (PSM) komentář: americká armáda, konkrétně Office of the Under Secretary of Defense for Acquisition and Technology, Joint Logistics Commanders, Joint Group on Systems Engineering dávají k dispozici podrobnou prakticky orientovanou příručku Practical Software Measurement – A Foundation for Objective Project Management; jedná se o skvělý text pro vedení projektů a měření.	http://www.psmsc.com
Software Program Managers Network (SPMN) komentář: tato skupina v rámci americké armády se zaměřuje na identifikaci nejproblémovějších věcí při vedení středních a velkých projektů, tedy na co si dát pozor a provádět pořádně; orientováno velmi prakticky; zorganizovali vypracování vynikající příručky The Program Manager's Guide to Software Acquisition Best Practices.	http://www.spmn.com
Software Technology Support Center (STSC) komentář: toto centrum americké armády (konkrétně vzdušných sil) dává k dispozici mnoho věcí. Jednak to jsou tzv. Technology Reports k určitému tématu, kde kromě základního přehledu dané problematiky jsou popsány podpůrné technologie (míněno nástroje), které podporují jednotlivé činnosti sw inženýrství, jako např. vedení projektů, konfigurační řízení, atd. Dále STSC vydává časopis CrossTalk, což je časopis ministerstva obrany pro softwarové inženýrství, velmi prakticky orientovaný na základní problémy vedení velkých projektů, dále v časopise vychází různé přehledové články týkající se sw inženýrství (archiv mnoho let dozadu); velmi cenné.	http://stsc.hill.af.mil http://stsc.hill.af.mil/CrossTalk

Guide to the Software Engineering Body of Knowledge (Guide to SWEBOK) komentář: široká iniciativa zastřešená dvěma hlavními světovými počítačovými profesními organizacemi ACM a IEEE Computer Society, která usiluje přesně a systematicky popsat, co konstituuje disciplinu softwarové inženýrství a jaké jsou související disciplíny (systémové inženýrství, computer science, atd.) a co z nich je potřeba. Iniciativa má dlouhodobý plán, co kdy má být hotovo (Straw Man Version, Stone Man Version, Iron Man Version atd.). Byly identifikovány základní oblasti, do kterých dělit sw inženýrství. Jsou k dispozici návrhy, co tyto oblasti tvoří, kritika těchto návrhů, atd. Velmi cenný zdroj informací již nyní a hlavně do budoucna. Projekt koordinuje The Software Engineering Coordination Committee (SWECC), společný výbor ACM a IEEE CS.	http://www.swebok.org http://computer.org/tab/swecc
The Formal Technical Review Archive komentář: archív zdrojů k formálním technickým přezkoumáním.	http://www.ics.hawaii.edu/%7Ejohnson/FTR
Institute for Zero Defect Software	http://www.izdsw.org
Rational Software komentář: firma, kde mimo jiné jsou pan Booch, Rumbaugh a Jacobson, všechno autoři široce používaných OO metodologií. Tvůrci UML notace (OMG standard) a nyní i UML Procesu pro vývoj. Firma dlouhé roky pracující pro armádu (velké projekty v jazyku Ada), firma produkující metody a nástroje pro mnoho činností v rámci sw procesu. Článek od Royce týkající se souvislostí mezi postupem vývoje, architekturou, požadavky a vedením projektu, který ukazuje nutné nové možnosti pro postup vývoje velmi usnadněné OO přístupem si autoři Acq. Guidelines dali do přílohy jako vzorové shrnutí co udělá v informacích z cca 2000 stran jasně (jedná se o stejnou debatu, která vedla k přechodu od standardu 2167A k MIL-STD-498 se všemi jeho moderními rysy; článek je uveden v sekci č. 3.9 Vedení sw projektu). Atd. Prostě firma Rational představuje nebývalou syntézu. Na serveru fy Rational je mnoho velmi kvalitních technický zpráv (tzv. White Paper) k mnoha tématům sw inženýrství. Velmi se věnují modelu postupu vývoje (model SDLC), což po „pádu“ modelu vodopád je věčné téma.	http://www.rational.com
Center for Software Engineering, University of Southern California (CSE) komentář: tým Dr. Boehma, legendy v Computer Science a Softwarovém inženýrství. Zabývají se modely vývoje, tj. modely SDLC (to je jinak řečeno koncepční strategie jak vést projekt, bez toho se neví podle jaké logiky plánovat!), zabývají se softwarovou architekturou, a to hlavně ve vztahu k vedení projektu, zabývají se požadavky zákazníka, atd. Samá zásadní témata. Pozn. Boehm je autor COCOMO, Software Economics, Spiral Model. Výzkumné zprávy jež jsou k dispozici jsou velmi cenný zdroj.	http://sunset.usc.edu
Configuration Management Yellow Pages komentář: obrovský zdroj informací k problematice konfiguračního řízení. Teorie, praxe, články, produkty, konference, atd. Místo je sponzorováno Software Engineering Laboratory of the University of Colorado.	http://www.cs.colorado.edu/users/andre/configuration_management.html
Software Configuration Management komentář: zdroj odborných materiálů pro činnost konfigurační řízení na serveru Software Engineering Institute.	http://www.sei.cmu.edu/legacy/scm
Software Productivity Center (SPC) komentář: zdroj odborných materiálů. Nabízí např. sadu vzorů pro plány, pracovní produkty a zprávy/ formuláře nutné pro dobrou praxi při projektech. Věnují se problematice vedení projektů, softwarovému procesu, odhadování velikosti/ času/ zdrojů, predikci defektů, atd. Většina věcí je za poplatek, nicméně spousta věcí je k dispozici zadarmo, či v omezené podobě. Mají elektronický newsletter, automatické info o novinkách, atd.	http://www.spc.ca
Internet FAQ Consortium komentář: archív FAQs (Frequently Asked Questions) jednotlivých diskusních skupin (Newsgroups) v rámci USENETu. Diskusní skupiny s předponou „comp“ se týkají počítačových věcí. Těchto počítačových diskusních skupin jsou desítky. Jsou jednak zaměřené na jednotlivá témata typu testování, konfigurační řízení, softwarové inženýrství jako celek, vedení projektů, paralelní programování, operační systémy, databázové systémy, objektově orientované programování atd. Dále jsou zaměřené na konkrétní programovací jazyky, operační systémy atd. Příslušné FAQ obsahují základní shrnutí tématu, přehled literatury, přehled zdrojů dosažitelných přes Internet, přehledy podpůrných nástrojů a jejich vlastností a dodavatelů atd. FAQs představují velmi cenný zdroj informací. Samotné diskuse vedené v diskusních skupinách jsou často velmi podnětné. Pár příkladů existujících diskusních skupin: - comp.realtime - comp.software-eng - comp.lang.c - comp.object - comp. software.config-mgmt - comp.software.testing - comp.programming.threads - comp.security - ...	http://www.faqs.org

Quality Checked Software komentář: tato firma dává k dispozici mnoho tzv. „White Papers“. Jedním z cílů je propagovat jejich nástroje pro testování. Nicméně články jsou stručné, maximálně prakticky orientované, ale zároveň nic nezjednodušující. Témata článku sahají od přehledu celé problematiky testování, přes návrh testovacích případů pro unit testování, přes zaobírání se tím jak navrhovat software, aby byl testovatelný, přes probírání problematiky testování programů psaných v C++ až po ilustraci jak naplnit nároky na testování ve všech jenom trochu známých sw standardech (příklady těch nejznámějších MIL-STD-498, TickIT, CMM, ESA PSS-05-0 Issue 2). Přes určité konkrétní zaměření články – white papers obsahují velmi mnoho univerzálního. A mnohým zaměření na C++ může bezprostředně vyhovovat a zaměření na Adu může být velkou inspirací. Dále je zde vidět obrovská snaha jak známé principy softwarového inženýrství dostat do každodenní praxe; to je velmi cenné.	http://www.qcsltd.com
Software QA and Testing Resource Center komentář: server obsahující materiály anebo odkazy k tématům zajišťování jakosti (Software Quality Assurance, SQA) a testování. Jsou k dispozici následující věci: - Software QA and Testing Frequently-Asked-Questions - Other Software QA and Testing Resources - Software QA and Test Tools - Web Site Test Tools and Site Management Tools - Software QA and Testing Bookstore	http://www.softwareqatest.com

Tabulka 5-1: URL adresy zdrojů odborných textů

6 Příloha č. 3: Co tvoří disciplinu softwarové inženýrství – přehled

V této příloze je uvedeno několik přehledů dekompozice a výčtu co tvoří disciplinu softwarové inženýrství. Je čerpáno z následujících textů/ zdrojů:

1. A Software Engineering Body of Knowledge, Version 1.0, SEI-99-TR-004
2. Guide to the Software Engineering Body of Knowledge, ACM & IEEE CS
3. Guidelines for Software Engineering Education, Version 1.0, WGSEET
4. SPICE.

Jednotlivé přehledy jsou různě detailní. Jednotlivé texty vznikly z různých důvodů. První a druhý text (v druhém případě se ve skutečnosti jedná o více textů) usilují popsat, co vlastně konstituuje disciplinu softwarové inženýrství. Třetí text se snaží rozčlenit a vymezit softwarové inženýrství z pohledu univerzitního vzdělávání. A konečně čtvrtý text je pracovní verze normy, která má sloužit hodnocení softwarového procesu a navržení jak ho zlepšit. U všech uvedených zdrojů bylo dříve v textu uvedeno, kde je lze získat a poskytnut stručný komentář.

6.1 A Software Engineering Body of Knowledge, Version 1.0; SEI-99-TR-004

Software Engineering Body of Knowledge je rozčleněno do Knowledge categories, tyto se skládají z Knowledge Areas (KA) a tyto se skládají z Knowledge Units (KU).

Computing Fundamentals (kategorie 1)
- The Algorithm and Data Structures (KA 1.1)
  - Basic Data Structures (KU 1.1.1)
  - Design of Algorithms (KU 1.1.2)
  - Analysis of Algorithms (KU 1.1.3)
- The Computer Architecture (KA 1.2)
  - Digital Systems (KU 1.2.1)
  - Computer System Organization (KU 1.2.2)
  - Alternative Architectures (KU 1.2.3)
  - Communications and Networks (KU 1.2.4)
- The Mathematical Foundations (KA 1.3)
  - Mathematical Logic and Proof Systems (KU 1.3.1)
  - Discrete Mathematical Structures (KU 1.3.2)
  - Formal Systems (KU 1.3.3)
  - Combinatorics (KU 1.3.4)
  - Probability and Statistics (KU 1.3.5)
- The Operating Systems (KA 1.4)
  - Operating Systems Fundamentals (KU 1.4.1)
  - Process Management (KU 1.4.2)
  - Memory Management (KU 1.4.3)
  - Security and Protection (KU 1.4.4)
  - Distributed and Real-time Systems (KU 1.4.5)
- The Programming Languages (KA 1.5)
  - Theory of Programming Languages (KU 1.5.1)
  - Programming Paradigms (KU 1.5.2)
  - Programming Language Design and Implementation (KU 1.5.3)

Software Product Engineering (kategorie 2)
- The Software Requirements Engineering (KA 2.1)
  - Requirements Elicitation (KU 2.1.1)
  - Requirements Analysis (KU 2.1.2)
  - Requirements Specification (KU 2.1.3)
- The Software Design (KA 2.2)
  - Architectural Design (KU 2.2.1)
  - Abstract Specification (KU 2.2.2)
  - Interface Design (KU 2.2.3)
  - Data Structures Design (KU 2.2.4)
  - Algorithm Design (KU 2.2.5)
- The Software Coding (KA 2.3)
  - Code Implementation (KU 2.3.1)
  - Code Reuse (KU 2.3.2)
  - Coding Standards and Documentation (KU 2.3.3)
- The Software Testing (KA 2.4)
  - Unit Testing (KU 2.4.1)
  - Integration Testing (KU 2.4.2)
  - System Testing (KU 2.4.3)
  - Performance Testing (KU 2.4.4)
  - Acceptance Testing (KU 2.4.5)
  - Installation Testing (KU 2.4.6)
  - Test Documentation (KU 2.4.7)
- The Software Operation and Maintenance (KA 2.5)
  - Software Installation and Operation (KU 2.5.1)
  - Software Maintenance Operations (KU 2.5.2)
  - Software Maintenance Process (KU 2.5.3)
  - Software Maintenance Management (KU 2.5.4)
  - Software Reengineering (KU 2.5.5)

Software Management (kategorie 3)
- The Software Project Management (KA 3.1)
  - Project Planning (KU 3.1.1)
  - Project Organization (KU 3.1.2)
  - Project Forecasting (KU 3.1.3)
  - Project Scheduling (KU 3.1.4)
  - Project Control (KU 3.1.5)
- The Software Risk Management (KA 3.2)
  - Risk Analysis (KU 3.2.1)
  - Risk Management Planning (KU 3.2.2)
  - Risk Monitoring (KU 3.2.3)
- The Software Quality Management (KA 3.3)
  - Software Quality Assurance (KU 3.3.1)
  - Verification and Validation (KU 3.3.2)
  - Software Metrics (KU 3.3.3)
  - Dependable Systems (KU 3.3.4)
- The Software Configuration Management (KA 3.4)
  - Software Configuration Identification (KU 3.4.1)
  - Software Configuration Control (KU 3.4.2)
  - Software Configuration Audit (KU 3.4.3)
  - Software Configuration Status Accounting (KU 3.4.4)
- The Software Process Management (KA 3.5)
  - Quantitative Software Process Management (KU 3.5.1)
  - Software Process Improvement (KU 3.5.2)
  - Software Process Assessment (KU 3.5.3)
  - Software Process Automation (KU 3.5.4)
  - Software Process Engineering (KU 3.5.5)
- The Software Acquisition (KA 3.6)
  - Procurement Process (KU 3.6.1)
  - Acquisition Process (KU 3.6.2)
  - Performance Management (KU 3.6.3)

Software Domains (kategorie)
- Artificial Intelligence (KA 4.1)
- Database Systems (KA 4.2)
- Human-Computer Interaction (KA 4.3)
- Numerical and Symbolic Computing (KA 4.4)
- Computer Simulation (KA 4.5)
- Real-Time Systems (KA 4.6)

6.2 Guide to the Software Engineering Body of Knowledge, ACM & IEEE CS

Proposed Baseline of a List of Knowledge Areas (Stone Man Version):
- Software Requirements Analysis
- Software Design
- Software Construction
- Software Testing
- Software Maintenance
- Software Configuration Management
- Software Quality Analysis
- Software Engineering Management
- Software Engineering Infrastructure
- Software Engineering Process

Proposed list of Related Disciplines (Stone Man Version):
- Computer Science
- Mathematics
- Project Management
- Systems Engineering
- Management and Management Science
- Cognitive Sciences and Human Factors

6.3 Guidelines for Software Engineering Education, Version 1.0; WGSEET

Software Engineering Body of Knowledge
Core Area
Core Area zahrnuje ty komponenty, které definují esenci softwarového inženýrství:
Software Requirements
Software Design
Software Construction
Software Project Management
Software Evolution

Foundations Area
Foundations Area zahrnuje ty komponenty, které formují základ pro Core Area a Recurring Area:
Computing Fundamentals
Human Factors
Application Domains

Recurring Area
Components in the Recurring Area are threads that occur through all of the Core area Components:
Ethics and Professionalism
Software Process
Software Quality
Software Modeling
Software Metrics
Tools and Environment
Documentation

6.4 SPICE

Členění do střední úrovně hrubosti granularity
Dle SPICE
CUS / Kategorie procesů – Zákazník–Dodavatel
Procesy:
CUS.1 Získání softwarového produktu anebo služby
CUS.2 Ustanovení kontraktu
CUS.3 Identifikování potřeb zákazníka
CUS.4 Vykonání společných auditů a přezkoumání
CUS.5 „Zabalení“, dodání a instalování softwaru
CUS.6 Podporování provozování softwaru
CUS.7 Poskytnutí servisu zákazníkovi
CUS.8 Zhodnocení spokojenosti zákazníka

ENG / Kategorie procesů – Inženýrských
Procesy:
ENG.1 Vypracování požadavků na systém a návrhu systému
ENG.2 Vypracování požadavků na software
ENG.3 Vypracování návrhu softwaru
ENG.4 Implementování návrhu softwaru
ENG.5 Integrování a testování softwaru
ENG.6 Integrování a testování systému
ENG.7 Udržování systému a softwaru

PRO / Kategorie procesů – Projektu
Procesy:
PRO.1 Plánování životního cyklu projektu
PRO.2 Ustanovení plánu projektu
PRO.3 Sestavení projektových týmů
PRO.4 Řízení (Manage) požadavků
PRO.5 Řízení (Manage) kvality
PRO.6 Řízení (Manage) rizik
PRO.7 Řízení (Manage) zdrojů a harmonogramu / časového plánu (schedule)
PRO.8 Řízení (Manage) subkontraktů

SUP / Kategorie procesů – Podpůrných
Procesy:
SUP.1 Vypracování dokumentace
SUP.2 Provádění konfiguračního řízení
SUP.3 Provádění zajišťování jakosti
SUP.4 Provádění řešení problémů
SUP.5 Provádění odborných přezkoumání (Peer reviews)

ORG / Kategorie procesů – Organizace*
Procesy:
ORG.1 Engineer the business
ORG.2 Definování procesu
ORG.3 Zkvalitňování procesu
ORG.4 Provádění školení a vzdělávání
ORG.5 Umožnění / podpora znovupoužívání
ORG.6 Poskytnutí SE environment
ORG.7 Poskytnutí pracovních prostředků / pomůcek (facilities)

Poznámky: Výše uvedených 35 činností (v terminologii SPICE procesy) se dále člení na 201 jemněji stanovených činností (v terminologii SPICE base practice).

Řazení pozic uvedených činností neznamená časové řazení. Jména inženýrských činností nejsou jména fází vývoje. – Tato poznámka je téměř doslova převzatá z ISO 12207, str. 55. Podobně formulovanou poznámku se stejným významem lze najít v MIL-STD-498, SPICE, SMG, ESA.
* Pozn.: Organizací je míněna organizační jednotka, ne činnost.

7 Příloha č. 4: Jiné seznamy principielních praktik

Pro představu je zde uvedeno několik jiných seznamů „nejdůležitějších“ věcí. Protože každý je z jiného úhlu pohledu, tak vycházejí trochu jinak. Stručný komentář a místo, kde je lze získat je k dispozici v kapitole č. 3.

7.1 Podle IEEE Software

Software's Ten Essentials[14]:

1. A product specification
2. A detailed user interface prototype
3. A realistic schedule
4. Explicit priorities
5. Active risk management
6. A quality assurance plan
7. Detailed activity lists
8. Software configuration management
9. Software architecture
10. An integration plan

7.2 Podle Software Program Manager's Network

SPMN Best Practices[15]:

1. Principal Best Practices:
   - 1.1. Formal Risk Management
   - 1.2. Agreement on Interfaces
   - 1.3. Formal Inspections
   - 1.4. Metric-Based Scheduling and Management
   - 1.5. Binary Quality Gates at the Inch-Pebble Level
   - 1.6. Program-Wide Visibility of Progress vs. Plan
   - 1.7. Defect Tracking Against Quality Targets
   - 1.8. Configuration Management
   - 1.9. People-Aware Management Accountability
2. Breathalyzer Test (pokud vedoucí projektu neumí na tyto otázky odpovědět nebo odpoví negativně, pak se projekt musí okamžitě přezkoumat):
   - 2.1. Do you have a current, credible activity network supported by a Work Breakdown Structure (WBS)?
   - 2.2. Do you have a current, credible schedule and budget?
   - 2.3. Do you know what software you are responsible for delivery?
   - 2.4. Can you list the current top ten project risks?
   - 2.5. Do you know your schedule compression percentage?
   - 2.6. What is the estimate size of your software deliverable? how was it derived?
   - 2.7. Do you know the percentage of extern interfaces that are not under your control?
   - 2.8. Does your staff have sufficient expertise in the project domain?
   - 2.9. Have you identified adequate staff to allocate to the scheduled tasks at the right time?

7.3 Podle CMM

CMM Level 2[16]:

1. Requirements Management
2. Software Project Planning
3. Software Project Tracking and Oversight
4. Software Subcontract Management
5. Software Quality Assurance
6. Software Configuration Management

CMM Level 3 (některé):

7. Software Product Engineering
8. Peer Reviews

[14] S. McConnell, „Software's Ten Essentials,“ In *IEEE Software*, Best Practices Column, March/ April 1997.
[15] The Program Manager's Guide to Software Acquisition Best Practices, SPMN (Software Program Management Network).
[16] Capability Maturity Model. Dosažení úrovně 2 znamená posun ze stavu *ad hoc* do stavu opakovatelných výsledků při podobném projektu. Je uveden seznam klíčových oblastí procesu pro úroveň 2.

8 Příloha č. 5: Kořeny a okolnosti vzniku tohoto textu

8.1 Ustanovení a rozvoj standardního softwarového procesu organizace + „Assets“

Tento text, jak bylo naznačeno v předmluvě, je do samostatně použitelného tvaru rozpracovaná část původně šířeji zamýšleného úsilí, a to ustanovení[17] a rozvoje standardního softwarového procesu organizace a „Assets“. Zde do sekce č. 8.1.1 vkládám část úvodní kapitoly k textu, který měl začít toto úsilí (takže pokud se ve vloženém textu odkazuje např. na sekci č. 1.2, pak zde to je sekce č. 8.1.1.2; vložený text má jako pozadí 5% šeď a je vytištěn menším písmem). Tato vložená část textu rozebírá některé otázky v souvislosti s tím, co znamená ustanovit standardní softwarový proces, jaké jsou třeba překonat problémy atd. Domnívám se, že tento stručný rozbor může být někomu užitečný.

8.1.1 Úvod

8.1.1.1 Účel textu

Text navrhuje počáteční předpis procesu vývoje softwaru pro celou organizaci. Spolu s tím navrhuje přístup k neoddělitelně souvisícím problémům: výbava užitečnými věcmi, znalostmi, údaji, informacemi atd. (tzv. Software Process Assets); ustanovení a další rozvoj předpisu procesu a výbavy (assets.) Dále materiál mnoho otázek spíše identifikuje a nastoluje (např. otázky týkající se určitých částí politiky, která má být součástí přepisu softwarového procesu, viz dále) a má být podkladem pro přemýšlení a zasvěcenou diskusi o stanovení a rozvoji předpisu softwarového procesu pro celou organizaci a souvisící výbavy (assets).

8.1.1.2 Problém a jeho vymezení

Organizace vyvíjející software potřebuje mít určitý předpis, platný pro celou organizaci, týkající se procesu vývoje softwaru. Potom každý projekt vývoje softwaru v organizaci má odpovídat tomuto předpisu. Na podporu přípravy a provedení softwarových projektů je dále vhodné mít k dispozici vzory plánů, vzory předpisů produktů, zpracování odborných témat, informace o existujících postupech, technikách a metodách, údaje o minulých projektech, atd., prostě výbavu potřebnými užitečnými věcmi, znalostmi, údaji, informacemi, kde sehnat další informace atd. Je-li organizace větší anebo vývoj software probíhá v různých kontextech je třeba předpis celoorganizačně platný přizpůsobit lokálním podmínkám.[18]

Je zřejmé, že předpis procesu může mít různý záběr, různou podobu, různou úroveň detailu atd. Různý záběr je jasný, tj. čeho všeho se týká. Různou podobu, představme si že se předpis procesu vyjadřuje k softwarové architektuře, pak může, např. nárokovat „pouze“ výsledný produkt procesu tvorby sw architektury nebo se může snažit tento proces dokonce popsat, může předepsat kontrolní seznam otázek pro odborné přezkoumání pracovního produktu sw architektura, může předepsat zásady použití určitého podpůrného nástroje (typu CASE), může předepsat detailní postup a pokyny jak architekturu udržovat atd. Velmi podobný postřeh platí i pro výbavu (assets). Dále je zřejmé, že vybudovat předpis procesu, který má dostatečný záběr a je dostatečně detailní, pro úroveň celé organizace, že vybudovat vhodnou doprovodnou výbavu (assets), že dále vypracovat zohlednění pro různé specifické kontexty, je velký úkol a že vlastně nikdy není u konce. (Pozn. procesu vývoje softwaru se také a hlavně říká ***softwarový proces***, přičemž tento pojem zahrnuje všechny činnosti a pracovní produkty (plány, artefakty typu návrh, dokumentace, odhady, ...), které existují a mají existovat při projektu, když se vyvíjí sw produkt (definice jinde); předpisu procesu – rozumí se softwarový proces – se může říkat definice procesu, dokumentovaný proces.)

Úkolem tohoto dokumentu je:

1) navrhnout počáteční/ úvodní podobu předpisu procesu pro celou organizaci
2) úvodní podobu pro software process assets
3) dále navrhnout jak tyto ustanovit a dále rozvíjet; to znamená:
   a) jednak jako takové
   b) jejich pokud možná rychlé a široké zavedení do skutečné praxe při softwarových projektech v organizaci (v různých kontextech).

Základní koncepční potíže při plnění tohoto úkolu jsou následující: (1) v případě úvodní podoby předpisu procesu pro celou organizaci je základní potíž co předepsat (tj. čemu se věnovat, co je tak důležité a podstatné) a jak to předepsat (tj. jak se tomu věnovat, aby se to mohlo co nejdříve promítnout do praxe);[19] (2) v případě úvodní podoby výbavy (software proces

[17] Přesněji řečeno, konkrétně šlo o ustanovení definice softwarového procesu s poměrně konkrétně formulovanými nároky. Pro účely této přílohy je to však nepodstatné.

[18] Běžně používané termíny v odborné literatuře pro to, co zde bylo velmi stručně a neformálně popsáno jsou: ***Standardní softwarový proces organizace*** (Organization's Standard Software Process) pro předpis týkající se procesu vývoje softwaru, který platí pro celou organizaci. ***Softwarový proces projektu*** (Software Project's Process) pro proces vývoje softwaru určitého projektu. ***Software Process Assets*** pro výbavu potřebnými užitečnými věcmi, znalostmi, údaji atd. Zatím nepovažuji za vhodné text zatížit přemírou definic a pojmů, které se v této oblasti používají. Pojmy budou používány intuitivně. (Pozn. Slovo „Assets“ se špatně překládá a české „výbava“ není nejpřesnější proto budu psát výbava (assets); v některých anglických textech, které nejsou úplně formální se lze setkat i ze slovem „goodies“.)

[19] Vybrat co předepsat a hlavně jak to předepsat je, i když se to nemusí zdát, netriviální problém. Rozsah softwarového inženýrství je obrovský (mnoho různých činností); dále každá činnost má mnoho aspektů (teorie, principy, notace, techniky/ metody, postupy, produkty, dokumentace, nástroje, měření atd.). Vezme-li se velká část činností, co tvoří softwarové inženýrství, a jde-li se jen malý kousek v aspektech, které se vezmou v úvahu a míře detailu vyjádření, pak rozsah má stovky či spíše tisíce stran a stále pro mnoho věcí najdete jen zmínku, že

assets) základní problém je, co představuje efektivní podporu žádané dobré praxe softwarového inženýrství a to poskytnout; (3) v případě ustanovení úvodní varianty a dalšího rozvoje definice procesu a výbavy (software process assets) jsou základní problémy následující: (i) přijetí změn personálem provádějícím vývoj; (ii) způsob zohlednění různých kontextů, ve kterých probíhá vývoj (definice procesu je platná pro celou organizaci; pro různé kontexty dané typem projektu (nový vývoj, rozvoj, údržba; malý projekt, velký projekt), typem sw produktu, použitými nástroji, atd. může být však vhodné definici procesu platnou v celé organizaci vhodně rozpracovat, dopracovat a přizpůsobit lokálním podmínkám, např. detailní instrukce pro práci s nástrojem pro podporu konfiguračního řízení splňující celoorganizační politiku); (iii) koncepce, cíl a způsob dalšího rozvoje definice procesu a výbavy (software process assets).

Z výše naznačených problémů je zřejmé, že výběr co předepsat a způsob jak to předepsat pro úvodní definici procesu je zcela zásadní. Proto zde bude stručně popsána zvolená logika tohoto výběru (podrobněji viz dále). Nejprve výběr nutně musí limitovat rozsah, aspekty a míru detailu úvodního předpisu, aby tento byl vypracovatelný v rozumném čase, absorbovatelný praxí v rozumném čase a případně přizpůsobitelný lokálním podmínkám v rozumném čase. (Pozn. toto je vysloveně praktická záležitost. Úvodní předpis procesu bude v každém případě znamenat „skok" a nesmí se to přehnat. Z dlouhodobého hlediska bude naopak zřejmě žádáno pro mnohé věci postupně vypracovat detailní pokyny a postupy; srov. dokumentovaný proces v Raytheon. Co se týká výbavy (software process assets), tak zde není nebezpečí, že by mohla vadit mnohost poskytnutých věcí, zde je problém se zajištěním jejich tvorby a zpřístupnění.) Dále pro výběr byla zvolena následující logika (resp. zásady, které se tento návrh bude snažit naplnit):

1) je třeba se v definici procesu vyjádřit k (nej)podstatnějším věcem a problémům (pozn. samo o sobě je toto triviální zásada, nicméně otázkou jistě je, které věci a problémy to jsou);
2) předpis/ definici procesu je třeba komponovat tak, aby:
   a) bylo možné, i když třeba jen velmi jednoduše a základně – třeba jen na úrovni snahy, jej praktikovat téměř „hned", bez žádných náročných příprav;[20]
   b) tvořil, když ne celé, podstatnou část jádra dobré praxe softwarového inženýrství v tom, *co* se má dělat;
   c) představoval rámec a stimul nárokující takové věci a chování, byť zpočátku do určité míry nutně mechanicky (pro někoho formálně), které systematicky začnou s praktikami, kterými se vyznačuje každá dobrá praxe softwarového inženýrství; (a tímto zároveň začal u personálu dané organizace vytvářet povědomí, čím se vyznačuje dobrá praxe softwarového inženýrství);
   d) představoval věci, o které je vždy smysluplné se snažit a jejichž způsob provádění lze postupně zlepšovat, upřesňovat, zdetailňovat, lokalizovat atd.; (viz též pozn. p. č. 20).

Uvedené zásady týkající se provádění výběru co předepsat a jak to předepsat v úvodní definici procesu schematicky ilustrujme na následujícím případě: Softwarová architektura je zcela zásadní věc, proto je zahrnuta do předpisu. Předepíšeme, že součástí činnosti návrhu softwaru musí být vytvoření softwarové architektury. Nyní se jedná o to, co vše k softwarové architektuře předepsat a jak to předepsat. Materiálu je nepřeberně,[21] problémů a otázek je velmi mnoho: co to je softwarová architektura (co vše ji tvoří), na základě čeho vzniká, architektonické styly, role v procesu vývoje a vedení, dodržování architektonických rozhodnutí v průběhu vývoje a údržby, údržba architektury, dokumentace, analýza atributů sw produktu na základě architektury atd. My se z počátku spokojíme s předepsáním specifikace, co má splňovat pracovní produkt sw architektura (tedy nároky na požadovanou činnost předepíšeme nároky na její výsledek). Jak jsou tím naplněny zásady pro komponování definice procesu: (1) bude identifikována a předepsána naprosto zásadní věc a to způsobem, který lze téměř hned aplikovat; (2) ustanoví se činnost a pracovní produkt přítomné v každé dobré praxi sw inženýrství. Jiná věc je, jak dobře se to bude dělat. Další rozvoj předpisu procesu a assets vzhledem k této věci může spočívat v detailnějším popisu, co má sw architektura splňovat v používaných vývojových a cílových prostředích, zpracování tématu v assets, tvorbu školení, vypracování vzorů odpovídající praxi v určitých lokalitách atd. – co konkrétně se bude dít, je rozumné nechat zatím otevřené v závislosti na reakci na úvodní předpis procesu.

8.1.1.3 Přehled navrženého řešení

V této sekci je řečeno co je předmětem navrženého řešení, stručně popsáno samo řešení a v případech, kde je to třeba, řečeno proč to tak bylo navrženo.

8.1.1.3.1 Počáteční předpis softwarového procesu a Software Process Assets

Nejprve, co je předmětem navrhovaného řešení a co není. Návrh se týká:

---

existují a žádné detailní postupy (Pozn. pro představu o rozsazích: příručka pro vedení sw projektů vzdušných sil má cca 2000 stran, přičemž mnohé je tam pouze schematicky; vezmeme-li jednu z mnoha (pod)činností, které tvoří činnost vedení projektu (jistě záleží na konkrétním strukturování, viz dále), např. měření, a chceme-li se k ní poctivě podrobněji vyjádřit, pak, např. ministerstvo obrany vydalo příručku Practical Software Measurement mající přes 500 stran; jiná důležitá (pod)činnost tvořící vedení, a to odhadování, lze provádět různými způsoby, popis jednoho z rozšířených modelů COCOMO II má přes 60 stran atd. Dalším problémem je členění/ strukturování celé disciplíny softwarové inženýrství; možných strukturování je celá řada a je dobré mít o nich představu. Z principu, návrh v tomto textu je nucen činit výběr z celého „univerza", co tvoří softwarové inženýrství, a je nutno použít nějaké strukturování celé problematiky. Aby tento návrh a jeho dosah mohly být plně pochopeny je třeba mít představu, co to vlastně je softwarové inženýrství, co vše jej tvoří, co aspoň zhruba představují jednotlivé konstituující části a jaké jsou přístupy k strukturování disciplíny.

[20] Zjednodušeně řečeno, jde o to, aby u principielních činností a pracovních produktů se dosáhlo změny stavu z „neprovádí se" na stav „provádí se" a ví se, že je to důležité. Jiná věc je způsob „provádění". Zde pro mnohé důležité věci typu sw architektura, odhadování, odborná přezkoumávání atd., vzhledem k jejich potenciální komplikovanosti, nezbývá než začít opravdu jednoduše, zabránit vysloveně špatnému provádění a postupně různými prostředky (např. obohacování definice procesu, obohacování assets, lokalizace/ přizpůsobení různým kontextům, vzdělávání atd.) dospívat k vyspělejšímu způsobu „provádí se".

[21] O softwarové architektuře se konají konference, vycházejí monografie, publikuje bezpočet článků, softwarové architektuře se věnují týmy ve firmách a na univerzitách (např. na katedře počítačů na CMU, v Institutu softwarového inženýrství), softwarové architektuře jsou věnovány celé výzkumné programy, např. agentury DARPA atd.

- počátečního obsahu předpisu softwarového procesu pro celou organizaci;
- logického členění počátečního předpisu softwarového procesu;
- počátečního obsahu některých částí výbavy (software process assets) (myšleno na celoorganizační úrovni);
- logického členění počáteční výbavy (software process assets).

Návrh se naopak netýká[22]:

- fyzického řešení a členění předpisu softwarového procesu a assets (oboje myšleno na celopodnikové/ celoorganizační úrovni);
- fyzického řešení a členění případných přizpůsobení a lokalizací předpisu softwarového procesu specifickým kontextům a podmínkám, ve kterých v organizaci probíhá;
- fyzického řešení a členění lokálních software process assets.

Následuje stručný popis samotného řešení:

Základní členění:

- Veškeré vyjádření k softwarovému procesu (ať už jde o předpis procesu, vzorové dokumenty, popisy principů / technik / postupů, údaje z minulých projektů, detailní postupy a pokyny k postupům / nástrojům, popisy modelů životních cyklů a další nutné a užitečné věci) se běžně anglicky[23] nazývá Software Process Assets, což už bylo výše řečeno. Součástí „assets" je předpis procesu vývoje softwaru platný pro celou organizaci (tj. standardní softwarový proces organizace). V tomto textu se budeme zabývat jednak předpisem softwarového procesu a jednak některými dalšími částmi „assets"[24].
- Předpis softwarového procesu, tj. jedna z částí „assets", pro podrobný popis toho, co nárokuje, doporučuje, dává k dispozici atd. využívá odkazů do jiných částí „assets", které tyto podrobné popisy obsahují (např. předpis softwarového procesu nárokuje vznik pracovního produktu plán sw projektu, samotná specifikace tohoto produktu je však obsažena v jiné části „assets" (pracovní název má „Oblasti/ činnosti")). Tímto postupem se dosáhne velké flexibility, např. (a) předpis procesu může z existující výbavy určovat, co je závazné, vysoce doporučené, doporučené atd.; (b) předpis procesu může obsahovat různé politiky k různým typům projektů; (c) lze lehce dosáhnout možných variant; (d) lze se předem připravovat na přechod mezi doporučený -> vysoce žádoucí -> závazný; atd. Dále má to také ten důvod, že členění části "assets", která obsahuje materiály k jednotlivým činnostem/ tématům je členěno právě podle činností. Takže u dané činnosti je napsáno v zásadě, co se zvládne, přičemž některé věci mohou být z pohledu předpisu procesu povinné, jiné jen doporučené a další materiály mají např. roli vzdělávací, obecně odborně informující (např. co danou činnost konstituuje) atd.

Počáteční předpis softwarového procesu:

Aby se dosáhlo řešení problému vymezeného v sekci č. 1.2 výše, je postupováno následovně:

a) *Co je softwarové inženýrství a jeho dobrá praxe*: Poskytne se základní vědomost (spíše povědomost) o tom, co je softwarové inženýrství a co je dobrá praxe softwarového inženýrství.

Komentář: Tato vědomost je poskytnuta za pomoci celkové struktury a pomocí některých popisů v části „Oblasti/ činnosti" software process assets. Např. podsekce „Čím je tato oblast / činnost konstituována" u jednotlivých činností / oblastí byla zařazena právě s tímto záměrem. Dále část „Různá používaná a navrhovaná strukturování" software process assets byla zařazena specielně se záměrem ukázat záběr softwarového inženýrství a jeho strukturování. Díky poskytnutí těchto informací je celkový materiál poněkud rozsáhlejší. Ale je třeba mít představu o tom, co je softwarové inženýrství a jaký má rozsah jako podklad, aby šlo vůbec pochopit, co je navrhováno v počátečním předpisu softwarového procesu a jaké jsou toho důsledky. Dále tyto informace musí být jako součást předpisu procesu a assets neustále k dispozici personálu provádějícím vývoj za účelem budování povědomí, co tvoří softwarové inženýrství a co je dobrá praxe softwarového inženýrství.

b) *Podstatné věci pro začátek*: Vyjmenují se činnosti, oblasti, praktiky a produkty, o kterých vedení organizace řekne, že si uvědomuje, že jsou maximálně důležité a že si je přeje mít co nejdříve a co nejšíře používané v praxi, jako počáteční krok k ustanovení dobré praxe softwarového inženýrství při projektech. Tedy vedení organizace deklaruje, vybírá podstatné věci pro začátek z toho, co tvoří softwarové inženýrství a jeho dobrou praxi.

Komentář: Na abstraktní a koncepční úrovni je identifikována a popsána určitá část věcí z ohromného rozsahu softwarového inženýrství, která je maximálně důležitá, maximálně užitečná a realizovatelné aspoň v nějaké jednoduché podobě. (Pozn. realizovatelné se myslí v kontextu dané organizace.) Pozn. zásady pro výběr podstatných věcí pro začátek viz sekce č. 1.2.

---

[22] Fyzické řešení a členění předpisu softwarového procesu pro celou organizaci, software process asstes pro celou organizaci, případného přizpůsobení a rozpracování předpisu softwarového procesu pro celou organizaci lokálnímu kontextu (dále jen lokalizace), případných lokálních software process assets lze provést mnoha vhodnými způsoby. Nedomnívám se, že by tento text měl určovat a rozhodovat zbytečně věci, které budou daleko lépe stanoveny po dohodě se zainteresovanými stranami. Dále fyzické řešení a členění výše uvedených věcí je věc ryze praktická a praxi by také měla sloužit, nicméně to není podstata problému. Podstata, esence problému je, jaký obsah budou předpis procesu pro celou organizaci a software process assets mít; to zde navrženo je. Doplňme na vysvětlení, v relativně velké organizaci s různorodým vývojem může být potřeba předpis procesu platný pro celou organizaci přizpůsobit lokálním podmínkám (lokalizace), dále pro lokální podmínky může být vhodné vytvořit lokální software process assets (je vlastně nutné mít předpis procesu a assets hierarchické nebo mít v předpise procesu od určité úrovně detailu a pro určité věci varianty; pozor není to úplně to samé jako různá míra detailu, srov. three-tier documented process v Raytheon na centrální úrovni, kde pak mají v rámci SEPG skupinu pro nástroje). Lokalizace procesu pro celou organizaci musí předpisu pro celou organizaci odpovídat. Lokální software process assets jsou prostě další potřebné a užitečné věci, které se hodí pro daný kontext a nemusí mít k celoorganizačním assets žádný přímý vztah. V tomto textu navrhujeme úvodní obsah pro předpis sw procesu a assets na úrovni celé organizace. Předpis procesu a software process assets, pro všechny úrovně (centrum, lokalita), jsou většinou dokumenty nejrůznějšího druhu, přístupné co možná nejpohodlněji.

[23] Slovo „asset" zde znamená vše co má nějakou hodnotu, co je nějak cenné, užitečné, použitelné a využitelné vzhledem k softwarovému procesu. Protože nenacházím jedno vhodné české slovo, tak buď používám původní termín anebo české „výbava" plus v závorce assets.

[24] Počáteční podoba software process assets, navržená v tomto dokumentu je navržena tak aby realizovala záměry tohoto návrhu (viz sekce č. 8.1.1.2 výše) a byla co nejjednodušší. Není to tudíž např. kopie toho co mají v CMM, spíše to má blíže podobě, o které mluví v materiálech Process Impact.

c) *Počáteční sada konkrétních nároků na praxi softwarového inženýrství*: Je poskytnuta sada konkrétních nároků na praxi softwarového inženýrství, které pomáhají dosáhnout – tím, co „rozumně, uměřeně a přitom zásadního“ nárokují – že se v reálné praxi při sw projektech objeví vybrané podstatné věci pro začátek.
Komentář: Nároky mohou mít nejrůznější podobu, např. specifikace pracovních sw produktů, vzory plánů, zásady dobrého návrhu, popisy činností, popisy postupů atd. Nároky jsou takové, které umožní byť velmi jednoduše a základně začít se základními elementy dobré praxe sw inženýrství. Nároky jsou takové, které je možno poměrně jednoduše formulovat a předepsat; a také zároveň realizovat, tj. zavést do praxe. Pozn. zásady pro výběr počáteční sady nároků viz sekce č. 1.2. Tato počáteční sada konkrétních nároků tvoří samotné jádro návrhu počátečního předpisu softwarového procesu.

d) *Politika vzhledem k nárokům na praxi*: Je potřeba stanovit, které konkrétní nároky (z výše uvedených) budou povinné, vysoce žádoucí, doporučené; pro koho; kdy; politiku vzhledem k výjimkám; časový výhled.
Komentář: Toto umožní zohlednit spoustu věcí (např. typ projektu); umožní pro jednotlivé projekty domlouvat, co vyzkouší atd. Stanovení této politiky se musí po zasvěcení zúčastnit příslušné zainteresované strany – tento dokument poskytne návrhy. Tato politika představuje velmi důležitou část předpisu procesu.

e) *Politika vzhledem k organizačním záležitostem*: Je potřeba stanovit věci typu, že plán sw projektu se dá k přezkoumání centrální skupině SEPG (Software Engineering Process Group); způsob žádání o výjimky; odpovědnost vedoucích za naplňování politiky a schváleného plánu sw projektu; možnost dozoru pomocí SEPG, útvaru jakosti; atd.
Komentář: Musí být prostředek jak prosazovat předpis procesu. Ad stanovení, stejně jako minule.

Předpis softwarového procesu bude obsahovat tyto zásadní elementy:

1) Politika:
   a) Deklarativní část: Co je softwarové inženýrství a jeho dobrá praxe; Podstatné věci pro začátek
   b) Nároky: Politika vzhledem k nárokům na praxi; Politika vzhledem k organizačním záležitostem
2) Konkrétní nároky:
   a) Počáteční sada konkrétních nároků na praxi softwarového inženýrství.

Poznámky:
a) K rozsahu, definice procesu by měla obsahovat jen to nejnutnější, aby se řeklo, co se chce a u věcí, kde je to potřeba, i jak se to chce (pozn. firma Raytheon má definovaný proces ve třech úrovních (politika, standardy/ praktiky sw inženýrství (*co*), detailní pokyny a postupy (*jak*)), který vypracovávala v průběhu času; tento definovaný proces má přes 770 stran (8, 220, 550) s tím, že detailní procedury zdaleka nejsou ke všemu, co se nárokuje (jsou k problematickým věcem anebo k věcem s největším ziskem). Software process assets může být naproti tomu libovolně rozsáhlý, prostě má to být výbava všeho potřebného a užitečného.

Počáteční podoba software process assets:
Návrh počítá, že software process assets budou mít na počátku pouze[25] tři části:

- Předpis softwarového procesu; (pozn. byl stručně uveden výše);
- Činnosti/ oblasti softwarového inženýrství;
- Různá používaná a navrhovaná strukturování softwarového inženýrství.

Část *činnosti/ oblasti softwarového inženýrství*: Tato část má výhledově obsahovat všechny potřebné a vhodné informace k činnostem a oblastem tvořící softwarové inženýrství, které jsou potřebné pro podporu požadované dobré praxe softwarového inženýrství při projektech. Potřebné a vhodné informace zajišťují několik věcí: (1) povědomost o softwarovém inženýrství; (2) specifikace nároků na softwarový proces; (3) další potřebné, užitečné a vhodné materiály a informace zprostředkovávající znalost softwarového inženýrství a podporující dobrou praxi sw inženýrství při projektech. Mezi potřebné a vhodné informace patří: definice činnosti; co činnost/ oblast tvoří; popisy principů, zásad, technik, metod, postupů, atd.; popisy, vzory, specifikace asociovaných pracovních produktů v to počítaje i plány; otázka asociovaných měření; otázka nástrojů; informace, kde nalézt odborné zdroje atd. Shromáždit potřebné a vhodné informace pro hlavní aspekty k činnostem tvořícím sw inženýrství a ještě relativně detailně je doslova jako shromáždit knihovnu (k rozsahu viz sekce č. 1.2). Proto bude postupováno v intencích zásad uvedených v sekci č. 1.2 výše. Nejprve je třeba vypracovat ty věci, které přímo podporují počáteční předpis procesu (tj. jsou jím referovány). Dále je třeba poskytnout podrobnější materiály k obtížným věcem (pozn. pořadí a způsob vypracování assets do plné podoby přesahuje záměr tohoto textu, co se týká ustanovení a rozvoje počáteční podoby). Minimální rozpracování pro začátek je: (a) popsat rozsah a náplň jednotlivých činností/ oblastí; (b) materiály a informace nutné pro aspoň minimální podporu toho, co se nárokuje v předpisu procesu; (c) další vhodná a potřebná výbava materiály a informacemi, co se podaří vypracovat. K struktuře, zde hovořím o logickém členění; tj. např. instrukce k používání nástrojů mohou být fyzicky u sebe a separátně, atd.

Část *různá používaná a navrhovaná strukturování softwarového inženýrství*: Tato část je zařazena aby bylo možno ukázat rozsah softwarového inženýrství, co je to softwarové inženýrství a jak se dá členit. (Pozn. tato část je součástí tohoto návrhu a bude součástí skutečných assets se stejným účelem.)

8.2 Možnost využití nedokončené pracovní verze Rámce softwarového inženýrství (RSI)

Jedná se o hrubě navržený Rámec softwarového inženýrství v podniku (RSI). RSI obsahuje v části G nedokončený rozsáhlý text (cca 130 stran) Poznámky k vedení projektu I, ze kterého lze získat řadu užitečných informací (úvodní kapitola je k dispozici v následující příloze č. 6):

- obsahuje mnoho komentářů k odborným zdrojům (příslušná sekce je v obsahu označena znakem hvězdička *)
- celkově text může dát představu o rozsahu nezúženě pojaté problematiky vedení projektu
- některé odkazy byly činěny již v textu výše

[25] Další částí bude „Údaje o procesu“ získaných sledováním projektů.

9 Příloha č. 6: Úvodní kapitola z textu: Poznámky k vedení projektu I

(Vložený text má jako pozadí 5% šeď a je vytištěn menším písmem. Jsou ponechány původní odkazy. Použité reference ve výňatku jsou uvedeny na konci této přílohy.)

Cílem vedení projektu je vyvinout produkt v rámci rozpočtu, v souladu s harmonogramem a v požadované kvalitě. ***Vedením softwarového projektu*** se rozumí proces plánování, organizování, personálního zajištění, monitorování, řízení a vedení. Tato definice je z malými obměnami použita v [ESA91, Sanders95, Tamayko89]. Vedení softwarového projektu (Software Project Management) lze též vedení projektu.

9.1 „Základní" metoda vedení projektu

Strašně zjednodušeně řečeno, „základní" metoda vedení projektu spočívá v
(i) ustavení organizace projektu;
(ii) zjistit a porozumět „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.
(Pozn.:
- body (i) až (iv) zhruba korespondují s „plánování, organizování, personální zajištění", bod (v) s „monitorování, řízení, vedení" z definice uvedené výše; pozor body (i) až (vi) nejsou míněny ***a priori*** sekvenčně po celou dobu trvání sw projektu.
- výše uvedená „základní" metoda je běžná, lze ji nalézt v NASA příručce Software Management Guidebook atd.; tato konkrétní formulace je na základě STSC Project Management Technology Report [STSC95] a neobsahuje stanovení modelu SDLC, jak to např. v SPICE, NASA příručce Software Management Guidebook, CMM, SEI Curriculum Module na Software Project Management [Tomayko89].)

Výše uvedená definice je velmi koncisní a elegantní, ale nechává vlastně vše otevřené, např. co se myslí pod pojmem plánování, monitorování, řízení, vedení atd. Dále uvedená „základní" metoda vlastně jen říká, že práci, co se má udělat je třeba rozložit na zvládnutelné úkoly; pomocí těchto úkolů (umožňují provést odhad rozsahu práce) provést analýzu nároků na čas, jiné zdroje, omezení a vzájemných závislostí; sestavit časový plán a přidělit zdroje; plán provádět, reálnou situaci monitorovat a vyhodnocovat, původní plán příslušně upravovat a udržovat. Tento nastíněný postup je vlastně, strašně zjednodušeně a hlavně mechanicky řečeno, jak vést prakticky jakýkoli projekt. Návod „práci, co se má udělat, je třeba rozložit na zvládnutelné úkoly" je zcela evidentní, správný a nevyhnutelný, nicméně pro praktické použití je to velmi abstraktní a spíš než návod je to konstatování nevyhnutelného postupu ovšem bez skutečného návodu, jak to učinit. Tedy dále zůstává otevřené, co se myslí „práce co se má udělat", jak se má ta celková práce rozložit na zvládnutelné úkoly, jak se ty úkoly mají naplánovat, jak se má postup práce monitorovat, jak plán průběžně upravovat atd.

Pozn. elementární / „základní" metodou se podrobněji zabývá hlavně sekce č. 4.2 dále v textu.

9.2 Nastínění nutných rozšíření k základní metodě

9.2.1 Nezúžený pohled na záběr a rozsah vedení softwarového projektu

Vedení projektu se týká veškeré práce, co se má v daných podmínkách udělat. Ta je dána požadavky zákazníka, omezeními daných zákazníkem, plněním standardů, plněním nároků konstituujících nutnou SE praxi (činnosti konstituující nutnou SE praxi viz např. SPICE, též řečeno v části B nebo v popisu části B v části A) atd. Plánování (součást vedení projektu) se tedy přirozeně týká všech činností, uveďme dvě velmi rozdílné: návrh a validace&verifikace. Plánování se týká všech aspektů: např. u návrhu je třeba stanovit, jak se dělá (metodu), nároky na tech. artefakty, které vyprodukuje 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 zdrojů; u návrhu je třeba stanovit jak se kontroluje – to je řekněme aspekt kvality. Obecně je třeba se vyvarovat zúženého vidění plánování pouze několika „typických" činností a aspektu zdrojů. Přehled činností přináší část B. Příklady plánů projektu, tedy co vše je třeba plánovat přináší část E. Vzory potřebných pracovních produktů jsou v části D; např. DID na architekturu může pomoci technickému aspektu v souvislosti s plánováním a provedením činnosti návrh.

Pozn. tímto tématem se podrobněji zabývá hlavně kapitola č. 5 dále v textu.

9.2.2 Zásadní důležitost koncepční organizace primárních aktivit SE – modelů SDLC

Jedna podmnožina problémů plánování a vedení projektu je „rozložení práce, co se má udělat na zvládnutelné úkoly" vzhledem k primárním aktivitám SE (tj. requirements, analýza, návrh, implementace, testování) a postupu vývoje samotného sw produktu – tj. jinými slovy organizace primárních aktivit SE při projektu (lze též p.a. SE). Tímto se zabývá problematika modelu SDLC (~ koncepční organizace p.a. SE). Vymyslet, rozhodnout, naplánovat a zajistit určitý postup vývoje vlastního sw produktu vzhledem k p.a. SE je svým způsobem myšlenkové jádro vedení projektu. Lze říci, že vedení / plánování ostatních věcí lze od zorganizování p.a. SE více méně odvodit – mám na mysli pochopitelně ty aspekty, kterých se to týká (tj.

např. u V&V, kdy se co kontroluje je navázáno na postup provádění p.a. SE; dalších aspektů plánování V&V jako konkrétní použité procedury pro V&V, nároky na zdroje, způsob zaznamenávání výsledků atd. se to tímto způsobem netýká). Proto se taky tato problematika vyčlenila jako samostatně artikulovaná problematika modelu SDLC nebo koncepční org. p.a. SE. Velmi zjednodušeně a schematicky řečeno jde o to, zda model SDLC waterfall, inkrementální, evoluční, spirálový atd. Totiž pouhé označení modelu SDLC např. inkrementální je nesmírně abstraktní, resp. vágní. Problematika modelů SDLC je v odborné literatuře velmi živá a je to nepochybně jeden ze základních diskusních námětů v SE komunitě. Je to např. vidět z následujících článků, technických zpráv, monografií, norem a jiných materiálů: [Boehm95, Nesi98, H-S90, Rational9x, Krutchen96?, Booch94, Boehm97, Boehm98, Scacchi87, Sorensen95, Veldhizen97, Karlsson97, Fowler97, Aoyama98, Royce96, Stevenson96, Graham97, Sanders95, MIL-STD-498, ESA91, NASA96]. (Pozn. problematika modelů SDLC je většinou ve standardech a různých příručkách [MIL-STD-498, ESA91, NASA96, ISO12207, SPICE95, Paulk93b, atd.] pojednána velmi stručně až schematicky, nicméně všichni zdůrazňují její důležitost; MIL se vyjadřuje na úrovni CSCI, pod touto úrovní pouze varuje, ať se zbytečně nepoužívá rigidní *en bloc* plánování a tím i model; tím vyzývá pro moderní přístup k modelům SDLC na běžně vnímané úrovni granularity.) Problematika modelů SDLC je zpracována v části F.

Pozn. tématem modelů životního cyklu systému/ softwaru – nezúženě – v souvislosti s vedením sw projektů se podrobněji zabývá hlavně sekce č. 4.3.2.2 dále v textu.

9.2.3 Danost: Postupné zpřesňování plánů a soustavná úprava a údržba plánů

Jedna z daností, se kterou je třeba při vedení projektu počítat, je nutné postupné zpřesňování a úpravy příslušných částí plánů (jakožto hlavního nástroje vedení projektu) z principielních důvodů.

Některé věci lze na začátku projektu stanovit, tj. naplánovat pro dobu trvání celého projektu celkem bez problému, např. nároky na architekturu (k tomu slouží mustry, DIDs atd. zpřístupněné v části D). U jiných věcí toto není možné, např. na začátku projektu, kdy jsou sotva známy požadavky zákazníka, lze těžko dělat WBS, kde figurují dekompoziční jednotky produktu, které nejsou zrovna na nejhrubší úrovni granularity, lze těžko připravovat test cases pro integrační testování atd. Prostě plánování a celé vedení je velmi závislé na stávajících již vyvinutých technických artefaktech, které v té dané době reprezentují vlastní vyvíjený produkt (což jsou požadavky zákazníka, požadavky na software, architektura, detailní návrh; přičemž na začátku může být požadavky na systém, návrh systému). Z tohoto důvodu se příslušné části plánů musí postupně zpřesňovat, resp. dodělávat tak, jak postupuje vlastní vývoj. Tomuto fenoménu věnuje zaslouženou pozornost kniha [Sanders95] (pod názvem "Two-tier approach to planning") a standard [ESA91].

Dále se příslušné části plánů (jakožto hlavní nástroj vedení projektu) upravují z důvodů reakce na skutečný vývoj událostí, které se monitorují a vyhodnocují; úpravy se týkají hlavně věcí spojených s plánováním úkolů, času a zdrojů.

Dále se příslušné části plánů upravují (ať už doplňují, zpřesňují, či opravují) v souvislosti se změnou požadavků v průběhu projektu. Se změnami požadavků je třeba počítat, protože to je fakt. Nicméně v nové době někteří prosazují takový model vývoje, kdy v počátečních stádiích projektu je cíl získat funkční robustní architekturu na základě „podstatných" požadavků a pak teprve zahájit samotný „velký" projekt; přichází se z fenoménem, že ne všechny požadavky jsou stejné. Tento přístup v různé míře vyargumentovanosti a v různých souvislostech lze nalézt v [Boehm95, Royce96, Fowler97].

Pozn. tématem postupného a průběžného plánování (lze i evoluční, iterativní, progressive refinement atd.) se podrobněji zabývá hlavně sekce č. 4.3.2.1 dále v textu.

9.2.4 Zásada: umožnit různé způsoby vedení vzhledem k různým částem produktu a projektu

Standard MIL-STD-498 [MIL-STD-498] velmi varuje před *en bloc* časovým plánováním pro vývoj CSCI a i komponent CSCI. Tento standard na nejhrubší úrovni granularity dekompozice produktu, tj. CSCI, umožňuje a podporuje svébytný autonomní přístup (co se vedení, plánování, technických věcí atd. v smysluplné míře týká) k jednotlivým CSCI; např. různé modely SDLC pro různé CSCI, různé harmonogramy pro různé CSCI i jejich části atd. MIL-STD-498 jednak dbá, aby na úrovni nejhrubší granularity umožnil relevantní Program strategies. Dále standard dbá, aby na úrovni CSCI připustil jakýkoli smysluplný model SDLC. Dále standard dbá, aby zamezil *a priori* pojímání vývoje produktu jako monolitu (celý analyzovat, celý navrhnout atd.) a toho dbá do několika úrovní dekompozice. Prostě standard podporuje různé přístupy (technické i manažerské) k různým částem produktu, pokud to je smysluplné či nutné. Samozřejmě, že výše uvedená flexibilita je možná po předchozí konsolidaci architektury, rozhraní atd., to je zřejmé, jinak by to byl chaos. (Pozn. stejný názor na tyto věci lze vidět i v ISO 12207, Annex B; občas jsou věci tohoto typu z MIL-STD-498 převzaty i do SMG/NASA [NASA96].) To co bylo řečeno si ilustrujme výňatkem z MIL-STD-498 Overview and Tailoring Guidebook [MIL-STD-498-O&T]: „ MIL-STD-498 byl strukturován tak, aby podporoval různé varianty program strategies a poskytoval vývojové organizaci flexibilitu v ustanovení vývojového procesu softwaru, který nejlépe vyhovuje práci, která má být udělána. Všechna tato flexibilita může být zničena rigidním časovým plánováním produktů k dodání. Běžné chyby jsou:
1) Zacházení se všemi CSCIs, jakoby musely být vyvinuty synchronně (in „lock-step") a dosahovat klíčových milníků ve stejný čas. Umožnění, aby CSCI měly různé harmonogramy může vyústit v optimálnější vývoj.
2) Zacházení se všemi software units, jakoby musely být vyvinuty synchronně (in „lock-step"), všechny navrženy do nějakého data, implementovány do nějakého data, atd. Flexibilita v plánování software units může být také efektivní.
3) Uvalení předurčené sekvence na MIL sw products. Zatímco sekvenční dodávání je nejjednodušší si představit, často to není nejefektivnější přístup …"

Pozn. podstatné koncepty pro postup procesu vývoje softwaru uvedené v MIL-STD-498 jsou převzaty a uvedeny převážně v sekci č. 4.3.3.2 dále v textu.

9.3 Role tohoto textu

Z doposavad řečeného v tomto úvodu je zjevné, že k tomu, aby byla nezjednodušeně a nezúženě pokryta problematika vedení projektu, je třeba říci a vysvětlit mnoho různých věcí. Role tohoto textu, tj. komentáře k vedení projektu je následující:

(a) zmapovat, podat a popsat problematiku vedení softwarového projektu v celé její šíři; podat nezúžený, nezjednodušený, do důsledků/ detailů jdoucí a někdy i kontroverzní obrázek této problematiky; uvést základní principy, danosti, zásady, pomůcky; (tedy poskytnout strukturu problematiky);

(b) poskytnout podrobnější výklad/ informace k vybraným tématům.

K čemu to bude sloužit:

(a) Nezúženě pojatá problematika vedení softwarového projektu je svým způsobem brána k celému softwarovému inženýrství a pro mnoho lidí brána fakticky dominantní. Tj. tento text by měl pro tento případ umožnit hned od začátku rozumně používat věci, které jsou k dispozici v částech B, C, D, E, F; (tj. maximální úvodní pomoc pro vedení projektu).

(b) To, že tento text poskytne strukturu problematiky vedení projektu, umožňuje orientaci v problematice (i) jednak pro účel samotného vedení projektu, (ii) jednak pro používání tohoto rámce, (iii) jednak, aby byl ustanoven kontext pro zabývání se jednotlivými věcmi, např. monitorování & sledování postupu (EV), SPMN Best Practices, přičemž toto zabývání může být již nyní v příslušné kapitole s výkladem jednotlivých témat, nebo později v rámci SPI atd.; (pozn. a protože je to brána k celému SE, tak to dá kontext pro zabývání se tématy nejen přímo spjatými s vedením projektu).

(c) Možnost napsat věci, které jsou důležité a mohou mít velký dopad, aniž by se zatím důkladně formálně zpracovala celá oblast software project management (která zahrnuje např. měření, odhady atd.).

9.4 Poznámky

9.4.1 Co se myslí nezjednodušený, nezúžený atd.

Když je řeč o nezjednodušeném, nezúženém, do důsledků/ detailů jdoucím podání problematiky anebo dočasně pouze její struktury, tak je míněno také následující: Mnoho dobrých nebo velmi dobrých textů má tu nevýhodu, že problematiku vedení projektu pojednává hlavně z některých aspektů anebo v určitém kontextu. Např. standard [ESA91] má vynikající důraz na postupné zpřesňování plánování vývoje (two-tier), bohužel zcela postrádá důraz na možnost jisté autonomnosti vedení/ plánování v souladu se strukturou produktu, což je jeden z klíčových důrazů vojenského standardu [MIL-STD-498] (a umožňuje to ba přímo vybízí k velké flexibilitě, co se modelů SDLC na různých úrovních granularity týká). Dále, např. ve zprávě [STSC95], ve které je podrobně popsána „základní“ metoda vedení projektu, nijak nezohledňují model SDLC. (Pozn. popis tvorby harmonogramu začíná tím, že se vytvoří WBS postupným rozkladem činností, které se mají udělat k vykonání projektu.) Tedy tento text chce v tomto smyslu podat nezúžený pohled. (Pozn. toto neříká, že texty typu [ESA91, STSC95] jsou špatné, nejsou špatné, jsou prostě psány z určitých výchozích pozic a v určitém kontextu.)

9.4.2 Rozsah a záběr textu versus konkrétní situace daných projektů

Konkrétní projekty jsou velmi různorodé. Může se jednat o vývoj systému z nuly; může se jednat o permanentní rozvoj a údržbu stávajícího systému; může jít o projekt malý, střední, velký; může jít o projekt typu se kterým jsou zkušenosti a naopak; může jít o projekt, ve kterém se zkouší nové technologie a naopak. Dále každý projekt má svůj kontext, tj.: požadavky a omezení zákazníka (nejde jen o funkční/ nefunkční požadavky na produkt, ale i o další významná omezení co se časování dodávky týká **à** možný vliv model SDLC, požadované participaci na kontrole atd.); zkušenosti vývojové organizace; vývojové a cílové prostředí atd. Výše uvedené možnosti mohou být různě kombinovány. Všechny tyto zmíněné věci mají velký vliv na typy problémů a situací, které je nutno zvládat a tím i potenciálně velký vliv na výsledný způsob vedení projektu. Např. pokud jde o rozvoj již existujícího systému, tak oproti „vývoji z 0“, zde není palčivá otázka modelu SDLC, není palčivá otázka přístupu k získání architektury (může mít velký vliv na model SDLC, viz [Boehm95, Royce 96]) atd., dále je zde dost pravděpodobně daleko menší stupeň volnosti v mnoha dalších věcech. Např. pokud jde o „vývoj z 0“ v kontextu známém a historicky daném oproti stejnému typu vývoje, ale v neznámém kontextu, tak je mezi nimi potenciálně z pohledu vedení velký rozdíl (např. co se týká odhadování atd.).

Tento text se snaží při popisu problematiky vedení softwarového projektu, jak už bylo výše několikrát řečeno, podchytit nezúženě a nezjednodušeně celou tuto různost. Tento text jakožto celopodnikový musí usilovat o podchycení problematiky nezúženě se všemi možnými stupni volnosti. Proto pro určitý projekt bude z tohoto textu relevantní vždy jen určitá podmnožina zmiňovaných problémů a situací; naopak text jako celek by měl být ale schopen „pomoci“ pokud možno všem situacím. Je velmi nepravděpodobné, že by se u jednoho skutečného projektu vyskytly všechny zde zmiňované stupně volnosti, tj. věci, které je nutno rozhodnout; spousta věcí je prostě dána konkrétní situací. Např. problém sledování vývoje reálné situace, tedy monitorování a jeho vyhodnocování je relevantní vždy. Naopak problematika modelů SDLC může být v některých situacích věc relativně bezproblémová (např. u údržby) jindy velmi palčivá. Text se samozřejmě věnuje obojímu atd. (Poznamenejme, že např. dřívější materiály NASA k vedení projektu [NASA90, NASA92] co se týká modelu SDLC velkou volnost neměly, měli *de facto* waterfall s tím, že od určité specifikované velikosti produktu se má v implementaci nebo v detailním designu a implementaci iterovat. Pozdější materiál NASA [NASA96] již žádný konkrétní model SDLC nepředepisuje. V tomto ohledu byly jinak vynikající a pořád relevantní materiály [NASA90, NASA92] zužující.)

Na nevyhnutelnou různost konkrétních projektů a nevyhnutelnou nutnost pro ně přizpůsobit různé standardy a přístupy je vhodné nějak reagovat. Proto má vojenský standard MIL-SDT-498 [MIL-STD-498] z prosince 1994 k sobě příručku Overview and Tailoring Guidebook [MIL-STD-498-O&T] z ledna 1996, která obsahuje instrukce k přizpůsobení standardu konkrétnímu projektu sestávající se z 18ti kroků a mnoha vysvětlování. Proto má Krutchen v článku [Krutchen96] popisujícím „A Rational Development Process“, který stojí ve vývojové posloupnosti krystalizace názoru na proces vývoje u fy Rational někde mezi Booch „mikro x makro proces“ [Booch94] a procesem vydaným k UML „Rational Objectory Process 4.1“ [Rational97], koncept diskriminantů, které představují různé charakteristiky projektu, které je třeba zohlednit při aplikaci procesu, který fa Rational používá. Charakteristiky to jsou následující: business context, velikost, stupeň novosti, typ aplikace. V článku je dále náznak taxonomie vývojových procesů a skica přizpůsobení pro velký vývoj na kontrakt a malý vývoj na prodej. Proto Software Technology Support Center vzdušných sil vydalo technický report [Budlong96], týkající se

přizpůsobení a dokumentace standardního softwarového procesu organizace pro konkrétní projekt. Proto má mezinárodní standard ISO 12207 [ISO122207] annexes A, B. Atd.

Existencí různých typů projektů se zabývá sekce č. 6.2.2. Dále celá kapitola č. 4, obzvl. sekce č. 4.4 se snaží pro „základní“ činnosti typu odhadování ukázat souvislosti tak, aby mohly být správně používány vždy.

Reference použité ve výňatku:

Aoyama98 — Aoyama, M. “Agile Software Process and Its Experience,” In *Proceedings of the 20th International Conference on Software Engineering*. IEEE Computer Society, pp. 3-12, Kyoto, Japan, April 1998.

Boehm95 — Boehm, B. *Anchoring the Software Process*. Technical Report, USC/CSE-95-TR-507, Center for Software Engineering, University of Southern California, Los Angeles, CA, November 1995. (také In *IEEE Software*, July 1996.)

Boehm97 — Boehm, B., Egyed, A., et al. “Developing Multimedia Applications with the Win Win Spiral Model,” In *Proceedings of 6th European Software Engineering Conference Held Jointly with the 5th ACM SIGSOFT Symposium on the Foundations of Software Engineering*. ACM SIGSOFT SEN, Vol. 22, No. 6, November 1997 / Lecture Notes in Computer Science Vol. 1301, pp. 20-39, Zurich, Switzerland, September 1997.

Boehm98 — Boehm, B., Egyed, A., Kwan, J., *et al*. “Using the Win Win Spiral Model: A Case Study,” In *Computer*. Vol. 31, No. 7, IEEE Computer Society, July 1998.

Booch94 — Booch, G. *Object-Oriented Analysis and Design with Applications*, 2nd ed. The Benjamin Cummings Publishing Company, Inc., Redwood City, CA, 1994.

Budlong96 — Budlong, F., Szulewski, P., and Ganska, J. *Process Tailoring for Software Projects Plans*. Version 1.04, Software Technology Support Center, Hill AFB, Utah, January 1996.

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

Fowler97 — Fowler, M. “Evolutionary Development,” In *A Survey of Object Oriented Analysis and Design Techniques*. Addison-Wesley, on-line, `http://www.awl.com/cp/fowlerindex.htm`, 1997.

H-S90 — Henderson-Sellers, B., and Edwards, J. “The Object Oriented Systems Life Cycle,” In *Communications of the ACM*. Vol. 33, No. 9, ACM, NY, September 1990. Standardization and Control, European Space Agency, Paris Cedex, France, February 1991.

ISO12207 — ISO/IEC 12207:1995(E). *Information technology - Software life cycle processes*. Joint Technical Committee ISO/IEC JTC 1, Information technology, Subcommittee SC 7, Software Engineering, First edition, 1995-08-01.

Karlsson97 — Karlsson, E., and Taxen, L. “Incremental Development for AXE 10,” In *Proceedings of 6th European Software Engineering Conference Held Jointly with the 5th ACM SIGSOFT Symposium on the Foundations of Software Engineering*. ACM SIGSOFT SEN, Vol. 22, No. 6, November 1997 / Lecture Notes in Computer Science Vol. 1301, pp. 519-520, Zurich, Switzerland, September 1997.

Krutchen96 — Krutchen, P. “A Rational Development Process,” In *CrossTalk*. The Journal of Defense Software Engineering, Software Technology Support Center, Hill Air Force Base, Utah, July 1996. (také White Paper, Rational Software Corporation, CA, 1996.)

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

MIL-STD-498-O&T — *MIL-STD-498 Overview and Tailoring Guidebook*. Joint Logistics Commanders, Joint Policy Coordinating Group on Computer Resources Management, Department of Defense, January 1996.

NASA90 — *Manager’s Handbook for Software Development*. Revision 1, SEL-84-101, Software Engineering Laboratory Series, GSFC, NASA, November 1990.

NASA92 — *Recommended Approach to Software Development*. Revision 3, SEL-81-305, Software Engineering Laboratory Series, GSFC, NASA, June 1992.

NASA96 — *Software Management Guidebook*. Software Program, NASA Guidebook, NASA-GB-001-96, National Aeronautics and Space Administration, Washington, DC, November 1996.

Nesi98 — Nesi, P. “Managing OO Projects Better,” In *IEEE Software*. Vol. 15, No. 4, IEEE Computer Society, July/August 1998.

Paulk93b — Paulk, M., Weber, C., *et al*. *Key Practices of the Capability Maturity Model for Software, Version 1.1*. CMU/SEI-93-TR-25, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, February 1993.

Rational97 — *Rational Objectory Process 4.1 – Your UML Process*. White Paper, Rational Software Corporation, CA, 1997.

Royce96 — Royce, W., and Devlin, M. “Improving Software Economics in the Aerospace and Defense Industry,” (Volume 1, Chapter 15, Addendum A) In *Guidelines for Successful Acquisition and Management of Software-Intensive Systems:*

Weapon Systems, Command and Control Systems, Management Information Systems, Version 2.0. Software Technology Support Center, Department of the Air Forces., June 1996. (také Technical Paper TP-46, Rational Software Corporation, CA, 1995.)

Sanders95 — Sanders, J., and Curran, E. *Software Quality: A Framework for Success in Software Development and Support*. ACM Press & Addison-Wesley Publishing Company, 1995.

Scacchi87 — Scacchi, W. Models of Software Evolution: Life Cycle and Process. SEI Curriculum Module, SEI-CM-10-1.0, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, October 1987.

Sorensen95 — Sorensen, R. "A Comparison of Software Development Methodologies," In *CrossTalk*. The Journal of Defense Software Engineering, Software Technology Support Center, Hill Air Force Base, Utah, January 1995.

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

Stevenson96 — Stevenson, R. "Focus on: Process, Process? What Process?" In *Object Currents*. Vol. 1, No. 2, http://www.sigs.com/objectcurrents, SIGS Publications, Inc., NY, February 1996.

STSC95 — *Report on Project Management and Software Cost Estimation Technologies*. Document no. STSC-TR-012-Apr95, Software Technology Support Center, Hill AFB, Utah, April 1995.

Tomayko89 — Tomayko, J., and Hallman, H. *Software Project Management*. SEI Curriculum Module, SEI-CM-21-1.0, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, July 1989.

Veldhuizen97 — Veldhuizen, T. "Session 2: The Software Process," In *Lecture Notes of SYDE 221 Software Design*. Winter 1997 Course, University of Waterloo, Canada, on-line, http://seurat.uwaterloo.ca/sd221/index.html, 1997.