Manažment softvérového systému a vplyv na manažment softvérového projektu

MILOŠ RADOŠINSKÝ

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
radosinm@szm.sk

Abstrakt. Obsahom eseje je stručný úvod do problematiky a oboznámenie sa so základnými konceptmi systémov pre manažment softvéru. V práci je vyhradený tiež priestor problematike vplyvu na manažment projektu.

Úvod

Manažment softvérového systému alebo v angličtine *software configuration management* (SCM) je súčasťou každého softvérového projektu. Jeho úlohou je zaistiť integritu a konzistenciu v softvérovom výrobku počas celého procesu vývoja a údržby. Tieto požiadavky kladené aj na podporné nástroje vyplývajú z podstaty a povahy softvérových systémov.

Softvér je neviditeľný a neuchopiteľný. Na jeho zviditeľnenie používame abstraktné modely ako napr. model štruktúry, funkčné modely a iné. Tieto modely sú súčasťou softvérového systému vo forme *dokumentácie* ako špecifikácia požiadaviek, špecifikácia softvéru a iné. Softvér je zložitý, lebo rieši zložitý problém. To vyžaduje dekompozíciu problému a jeho riešenia na menšie podproblémy a celky. Na najnižšej úrovni delenia softvéru možno rozpoznať *súčiastky* alebo *komponenty*, ktoré sa už nedelia.

Komponenty a dokumenty definované v tomto texte podliehajú ustavičným zmenám. Zvyčajne existujú vo viacerých verziách. Navyše medzi nimi existujú rôzne závislosti (napr. "používa", "je abstrakciou" a iné). Keď k tomu všetkému pripočítame ešte fakt, že softvérový systém môže existovať v niekoľkých variantoch, je nutné zaviesť pravidlá a definovať postupy na udržiavanie konzistencie výrobku. Nie bezvýznamná sa javí podpora spolupráce členov tímu, ktorá môže ovplyvniť celkovú organizáciu tímu.

V tejto práci sa budem venovať podpore nástrojov pre spoluprácu v tíme a organizácii údajov v nástrojoch na podporu manažmentu softvéru.

Manažment v softvérovom inžinierstve, december 2005, s. 1-8.

Základné úlohy v manažmente softvéru

Na podporu manažmentu softvéru treba v projekte plánovať štyri základné úlohy. Tieto úlohy možno rovnako dobre aplikovať v prostredí výrobcu v procese vývoja ako aj v cieľovom prostredí počas údržby. Všetky úlohy by mali byť v rámci organizácie štandardné, ale mali by sa prispôsobovať pre potreby konkrétnych projektov.

Identifikovanie entít

Tu treba definovať časti systému (entity), ktoré budú podliehať kontrole pri zmenách. Tieto entity je nutné jednoznačne identifikovať. To možno docieliť organizovaním systému a jeho častí do hierarchie. Miesto v hierarchii potom môže entitu jednoznačne identifikovať. Identifikované entity môžu byť napr. zdrojový kód programu, plán projektu, akceptačné testy, teda komponenty a dokumentácia systému.

V tomto bode treba definovať aj tzv. základné konfigurácie - *baselines*. Základná konfigurácia (baseline) je dobre definovaný stav výrobku. V projekte možno definovať niekoľko základných konfigurácií. Pre každú z nich treba definovať, ktoré entity majú byť jej súčasťou a v akom stave. Riešenie projektu sa potom plánuje a realizuje s ohľadom na tu definované základné konfigurácie. Možno definovať napr. analytické, návrhové, testovacie základné konfigurácie. V nich sa definujú stavy systému, ktoré budú na konci analýzy, návrhu a testovania. Základná konfigurácia je podobný koncept ako míľnik.

Riadenie zmien

Jeden z najdôležitejších cieľov manažmentu softvéru je zachovať konzistenciu výrobku aj napriek ustavičným zmenám. To sa dosahuje riadením zmien na identifikovaných entitách. Riadenie je súčasťou celého procesu zmeny od zadania až po implementáciu zmeny. Požiadavka na zmenu obyčajne obsahuje tieto informácie: entity, ktoré sa majú zmeniť, pôvodca, dátum, priorita zmeny a popis zmeny. Treba si pritom uvedomiť, že nie je zmena ako zmena. Treba brať na zreteľ stav entity, ktorá sa má zmeniť. Entita, ktorá je súčasťou základnej konfigurácie (baseline), má vyššiu váhu ako entita, ktorá je iba prechodom medzi dvoma základnými konfiguráciami. Toto treba brať do úvahy aj pri posudzovaní zmeny.

Evidovanie stavu systému

Evidovanie stavu systému slúži hlavne manažérom projektu na sledovanie súčasného stavu systému. Bez týchto informácii by neboli schopní posúdiť, či projekt beží v súlade s plánom. Evidované informácie môže využiť napr. komisia na schvaľovanie zmien (CCB) – evidujú sa správy o navrhovaných a schvaľovaných zmenách a správy o problémoch. Komisia CCB môže tiež sledovať stav realizovaných zmien, takže môžu byť rýchlo odhalené a vyriešené potenciálne problémy.

Overovanie stavu systému

Hlavným cieľom tejto úlohy je zabezpečiť kvalitu a udržovať integritu výrobku. Overuje sa tu, či existujúca konfigurácia systému je kompletná, správna a konzistentná. Okrem toho sa overuje, či sprievodné informácie získané v úlohe evidovanie stavu systému sú spoľahlivé a konzistentné s reálnym stavom výrobku.

Podpora spolupráce v tíme

Dnes sa aj tie najmenšie projekty riešia v tíme. Úsilie ľudí v tíme musíme koordinovať tak, aby si vzájomne nezasahovali do práce. Na druhej strane by sme mali vytvoriť také podmienky, aby boli čo najviac produktívni. Koordinácia práce spočíva v riadení zmien. K zmenám nemôže dochádzať náhodne a bez kontroly.

Ak sa komponenty v tíme zdieľajú bez obmedzenia, môže vzniknúť potreba uskutočniť zmeny paralelne. Na to sú používané dve rôzne stratégie: zakázať alebo podporiť paralelné zmeny. Keď sú zakázané, zmeny treba vykonať sekvenčne. Na to stačí zamykať komponenty predtým, ako môžu byť zmenené. Zámok dáva istotu, že iba jedna osoba v čase môže zmeniť komponent. Nástroje, ktoré umožňujú paralelnú zmenu komponentov, jednoducho vytvoria vetvu, aby dali najavo fakt, že sa vykonávajú dve paralelné zmeny, ako v prípade nástroja CVS. Väčšina nástrojov je tiež schopná zlúčiť dve vetvy späť do jednej verzie (vetvy), aby vzniklo zloženie dvoch zmien.

Čo by sme chceli získať, je rozdelenie času na obdobie, kedy vykonávame zmeny izolovane od iných zmien a obdobie, kedy integrujeme vlastné zmeny so zmenami ostatných členov tímu. Na tento účel môžeme použiť tzv. *Pracovný priestor* a tzv. *Centrálne úložisko* (repository). V pracovnom priestore dochádza k zmenám. V centrálnom úložisku dochádza k integrácii zmien. Podrobnejšie sa tomu venujem v kapitole SCM modely.

SCM modely

V nástrojoch na podporu manažmentu softvéru boli podľa [2] vypozorované štyri modely organizácie údajov. Všetky modely majú spoločné to, že používajú jedno centrálne úložisko súborov – *repository*. Jednotlivé modely sa však líšia jeho organizáciou a obsahom údajov. V praxi sa používajú oddelene alebo v kombinácii s malými odchýlkami. Všetky štyri modely realizujú správu konfigurácií a spoluprácu členov tímu rôznym spôsobom.

Check Out/Check In Model

Je základný SCM model, ktorý zavádza koncept centrálneho úložiska (repository), v ktorom sú uložené viaceré verzie jednej súčiastky. Model je zameraný na prácu s individuálnymi súbormi. Nevytvára model systému a nepozná pojem konfigurácie. Verzie súborov sa identifikujú explicitne číslami. Pojmy ako check-out a check-in

označujú operácie výberu a vrátenie súčiastky z a do centrálneho úložiska. Tieto pojmy sa uplatňujú aj pri ďalších modeloch. Do tejto kategórie možno zaradiť systémy Source Code Control System (SCCS) alebo Revision Control System (RCS).

V tomto modeli sa uplatňuje pesimistický model spolupráce. Jednu súčiastku môže modifikovať najviac jeden člen tímu. Ak aj druhý člen chce modifikovať rovnakú súčiastku, musí vytvoriť jej novú vetvu (variant) v grafe verzií. Neskôr sa môžu obidve vetvy spojiť do jednej vetvy pomocou vizualizačného nástroja, ktorý zobrazí ich rozdiely. Ak chce druhý člen tímu obsah súčiastky iba čítať, druhá vetva sa nevytvorí. Systém (model) pritom rozlišuje výber súčiastky s právom na zápis a na čítanie. Právo na zápis dáva možnosť modifikovať a vrátiť súčiastku do úložiska (repository).

Kompozičný Model

Kompozičný model rozširuje SCM z úrovne komponentov na úroveň celého systému. Zavádza model systému a konfiguráciu systému, ktoré popisujú štruktúru systému ako množinu komponentov, verzií komponentov a ich relácií. Model systému tu reprezentuje všetky možné konfigurácie systému. Konfigurácia vzniká z modelu systému aplikovaním výberových pravidiel, ktoré v rámci jednej rodiny komponentov vyberú konkrétnu verziu na základe určitých vlastností verzie. Konfigurácia je potom daná modelom systému a výberovými pravidlami.

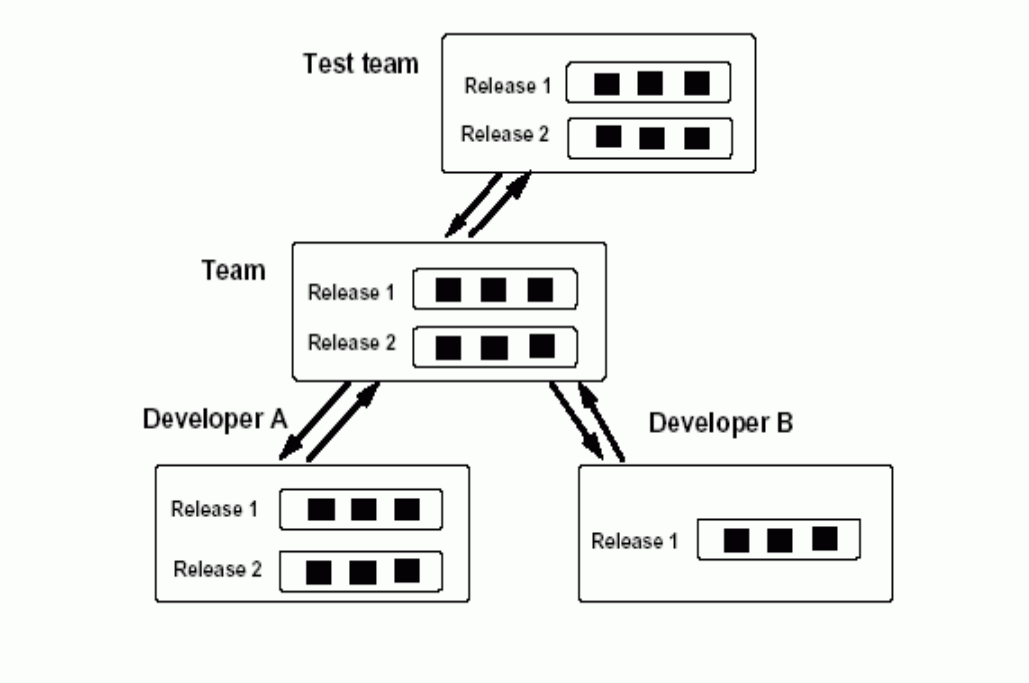
V modeli možno rozpoznať dva typy konfigurácii – viazaná a čiastočne viazaná. Viazaná konfigurácia je v čase jednoznačná. Výberové pravidlá vyberajú konkrétne verzie explicitne a jednoznačne. Táto konfigurácia môže byť pomenovaná názvom. Je trvalo uložená a vedená v centrálnom úložisku. Môžu sa na ňu odkazovať iné konfigurácie. Čiastočne viazaná konfigurácia nie je v čase jednoznačná. Výberové pravidlá v nej vyberajú verzie implicitne, napr. najnovšia verzia. Tento typ konfigurácie je označovaný ako konfiguračný predpis alebo šablóna (template). Používa sa na aktualizáciu pracovného priestoru (working area) vývojára. Rozdielne konfigurácie sa môžu vytvoriť použitím rôznych výberových pravidiel verzií. Na riadenie spolupráce tímu sa používa model Check In/Check Out.

Transakčný Model

Transakčný model chápe evolúciu systému ako postupnosť atomických – transakčných zmien. Tieto zmeny sa vykonávajú nad kópiou pôvodných údajov v tzv. *Pracovnom priestore*. Pracovný priestor je miesto, kde dochádza k izolovaným zmenám systému. Tieto zmeny sú počas transakcie pre pôvodné dáta (systém) neviditeľné. Po ukončení transakcie sa zmeny šíria na miesto pôvodu. Transakciu teda možno chápať ako postupnosť krokov a operácii, ktoré vedú k izolovanej zmene údajov (systému).

Tento koncept sa zatiaľ nijako nelíši od konceptu centrálneho úložiska (repository) v ostatných modeloch SCM. Nie je to celkom tak. Pridanou hodnotou transakčného modelu je pracovný priestor. Oproti iným modelom SCM má pracovný priestor v transakčnom modeli niekoľko zaujímavých vlastností. Pracovný priestor má podobné vlastnosti ako centrálne úložisko. Môže prijímať, uchovávať a posielať nové

konfigurácie systému – môže byť miestom vzniku alebo miestom uloženia konfigurácie systému. Okrem toho sa pracovné priestory môžu organizovať v hierarchii, v ktorej sú definované vzťahy podriadený a nadriadený. Podriadený pracovný priestor je zdrojom konfigurácie (údajov) pre nadradený pracovný priestor. Napríklad pracovný priestor celého tímu je podriadený pracovnému priestoru jedného člena tímu. V každom pracovnom priestore môžu prebiehať transakčné zmeny nad údajmi z podriadeného pracovného priestoru. Po ukončení transakcie sa zmeny šíria na miesto pôvodu – podriadený pracovný priestor.



Obr. 1. Organizácia tímu

Uvediem teraz príklad organizácie tímu podľa obrázku. Na obrázku **Obr. 1** je znázornená hierarchia pracovných priestorov. Každý vývojár má vlastný pracovný priestor, ktorý je nadradený pracovnému priestoru tímu a ten je nadradený testovaciemu tímu. Vývojár vyberá konfiguráciu z pracovného priestoru tímu. Zmena vo vybranej konfigurácii zahájí transakciu. V pracovnom priestore vývojára sa vytvorí pracovná konfigurácia, na ktorej sa robia zmeny. Po dokončení zmien sa otvorená transakcia uskutoční (ukončí). Vzniká nová konfigurácia v mieste pôvodu. Uskutočnené zmeny sa zviditeľnia v pracovnom priestore tímu.

Naznačený postup nebýva vždy tak priamočiary. Komplikácie môžu nastať v situácii, ako je na obrázku, keď na jednej vetve konfigurácie systému pracujú viacerí vývojári. Pracovný priestor tímu je zdieľaný a dochádza v ňom k súbežnej aktualizácii systému. V pracovnom priestore tímu sa zviditeľňujú čiastkové zmeny od jednotlivých

členov tímu. Každá zmena sa zviditeľní v novej konfigurácii v priestore tímu. Tá vznikne zlúčením zmenenej konfigurácie vývojára s najnovšou konfiguráciou v priestore tímu. V pracovnom priestore tímu tak vznikajú nové konfigurácie ako výsledok zlúčenia všetkých čiastkových zmien. Bezkolízne zlučovanie konfigurácií sa deje automaticky. Ak ale systém rozpozná kolíziu, konfigurácie sa musia zlúčiť manuálne. Konfigurácie možno zlučovať aj v pracovnom priestore jednotlivých členov tímu, keď požiadajú o aktualizáciu svojho pracovného priestoru. Po tom, ako sa úspešne zlúčia všetky čiastkové zmeny v pracovnom priestore tímu, môže sa ich transakcia ukončiť. V pracovnom priestore testovacieho tímu sa objaví najnovšia konfigurácia. Po úspešnom otestovaní testovací tím ukončí transakciu a otestovaná konfigurácia sa zapíše do centrálneho úložiska.

V transakčnom modeli používateľ pracuje primárne s konfiguráciou systému. To znamená, že používateľ primárne vidí konfigurácie. Model systému a použité verzie sa odvodia z konfigurácie systému – sú v nej definované. Tento prístup je opačný vzhľadom na kompozičný model SCM, kde používateľ vyberá najskôr model systému (štruktúru systému) a potom konkrétne verzie komponentov (konfiguráciu).

V modeli sú tri kategórie súbežnosti:

- súbežnosť v rámci jedného pracovného priestoru,
- súbežnosť medzi viacerými pracovnými priestormi,
- súbežnosť, nezávislý vývoj.

V prvom prípade sú súbežné zmeny zakázané. Obmedzuje sa buď prístup do pracovného priestoru alebo aktivita na jednu osobu. Okrem toho je tiež možné zamknúť individuálne komponenty. V druhom prípade zmeny v oddelených priestoroch spoločne vyvíjajú systém. Modely spolupráce, ktoré riešia túto súbežnosť, sú buď optimistický alebo pesimistický model. Pesimistický model zamyká zdieľané podriadené pracovné priestory alebo zdieľané centrálne úložisko. Tretí prípad predpokladá, že systém sa vyvíja na nezávislých konfiguráciách – variantoch.

Model založený na zmenách

Ako vyplýva z názvu, tento model sa zameriava na zmeny, nie na verzie. Pri vývoji systému vznikajú požiadavky na rozšírenie systému alebo opravu chýb – *požiadavky na zmenu*. V jednej požiadavke na zmenu sa zvyčajne špecifikuje iba jedna *logická zmena*. Avšak tá môže spôsobiť zmeny vo viacerých komponentoch – *fyzické zmeny*. Teda jedna požiadavka na zmenu môže vyvolať niekoľko fyzických zmien. Orientácia na zmeny má preto oproti orientácii na verzie niekoľko výhod. Vývojári pristupujú a pracujú naraz so skupinou komponentov, ktoré patria k jednej logickej zmene. Požiadavky na zmeny sa môžu jednoducho priradiť k fyzickým zmenám komponentov.

Konfiguráciu systému tu tvorí tzv. *baseline* (model systému) a množina logických zmien. Model systému je v tomto prípade daný množinou komponentov a ich relácií. Pričom každý komponent je v modeli zastúpený práve jednou verziou, na rozdiel od kompozičného modelu systému (súčasťou modelu systému sú všetky verzie).

Rozdielne konfigurácie môžu vzniknúť aplikovaním rozdielnej sady logických zmien na základ (baseline). Avšak nie všetky kombinácie zmien sú konzistentné. Niektoré zmeny sú vzájomne závislé alebo konfliktné. Konfliktné zmeny sa nesmú použiť súčasne. Naopak závislé zmeny sa musia použiť súčasne. Na riadenie súčinnosti tímu sa používa model CheckIn/CheckOut.

Zhrnutie vlastností modelu:

- Každá fyzická zmena komponentu je súčasťou logickej zmeny.
- Každý komponent môže byť súčasťou logickej zmeny.
- Každá konfigurácia môže mať niekoľko logických zmien.

Identifikácia verzií

Podľa [4] existujú tri spôsoby ako rozlíšiť verzie komponentu:

1. Číslovanie alebo značkovanie verzií sa používa na značkovanie variantov alebo číslovanie revízií. Revízie sa číslujú, pretože sú usporiadané do postupnosti a ich poradie je dôležité. Varianty sa značkujú názvom, pretože nie sú usporiadané do konkrétnej postupnosti a názov variantu (vetvy) obyčajne odráža niektoré vlastnosti variantu (vetvy). Čísla a značky sú súčasťou grafu histórie verzií. V tomto grafe má každá revízia svoje číslo a každý variant svoje meno – každá revízia a variant má v grafe svoje miesto.
2. Identifikácia na základe vlastností (atribútov) sa používa na identifikáciu variantov. Pre každý komponent sa najskôr definuje n-rozmerný vektor vlastností. Tento vektor slúži na identifikáciu variantov jedného komponentu. Variant komponentu sa potom vyberá z n-rozmerného priestoru verzií, kde každý rozmer reprezentuje jednu vlastnosť komponentu. Takýto prístup nepoužíva a nie je obmedzený na graf verzií.
3. Identifikácia pomocou zmien sa používa na identifikáciu revízií. Miesto číselného značenia revízie možno použiť značenie na základe logických zmien, pričom každá logická zmena má svoje meno. Revízia sa potom môže identifikovať ako postupnosť pomenovaných zmien, ktoré sa aplikovali na základ (baseline).

Uvedené spôsoby možno kombinovať. To vyplýva z rozdielnej podstaty verzií. Pre revízie možno použiť identifikáciu na základe čísla alebo na základe zmien. Pre varianty možno použiť značkovanie alebo vlastnosti.

Záver

V tomto príspevku som sa snažil podať stručný úvod do problematiky manažmentu softvéru a oboznámiť čitateľa so základnými konceptmi systémov pre manažment softvéru.

Použitá literatúra

1. L. Bendix, A. Dattolo, F. Vitali: Software configuration management in software and hypermedia engineering: A survey. Department of Computer Science, Aalborg University.
2. Peter H. Feiler: Configuration Management Models in Commercial Environments. Technical Report CMU/SEI-91-TR-7 ESD-9-TR-7, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, March 1991
3. Jari Vanhanen: Improving configuration management processes of a software product. Helsinki University of Technology, december 97
4. Andreas Zeller.: Configuration Management with Version Sets A Unified Software Versioning Model and its Applications. Der Technischen Universität Braunschweig, April 1997

Annotation

Software Configuration Management and the impact on Software Project Management

This essay introduces the basic concepts of Software Configuration Management and describes different types of software tools for SCM. The different types of SCM tools are described from technical and from teamwork point of view. It handles the SCM impact on the project management too.