

# 1 Configuration management

## 1.1 Ciel

Cielom tejto casti prednasky je presvedcit studentov o dolezitosti CM, teda preco by sme ho mali pouzivat.

Mali by sme predstavit zakladne koncepty CM - pojmy a zodpovednosti.

Pripadne, by sme chceli ukazat konkretny priklad implementacie CM, teda konkretny nastroj, model CM, proces, zodpovednosti a ulohy.

## 1.2 Na uvod si dajme priklad...

Predstavte si, ze ste programator vramci mensieho vyvojoveho timu, povedzme 6 ludi. Vas projekt je stredne velky (trvanie asi 4-5 mesiacov) a jedna sa o virtualny obchod na internete pre velkeho (a velmi znameho) klienta. Momentalne ste vo faze "velkeho kodovania", cize asi tak v 2/3 projektu a samozrejme nestihate.

Klient vam dal termin, ze o dva tyzdne chce choditelnu verziu toho, co ste doposial urobili, aby to mohol predviest vykonnej rade (executive board), koja rozhoduje o financovani (resp. nefinancovani) celeho projektu.

Samozrejme cakaju vas dva perne tyzdne - teda bude sa tvrdo pracoat.

Kedze ste mladi, ambiciozni, a zalezi vam na uspechu celeho projektu aj vasej firmy, rozhodnete sa, ze zamakate a stanoveny termin stihnete.

Verzia systemu, na ktorej ste doposial pracovali, je relativne "chodiva", teda obcas padne, ale dala by sa klientovi slusne odprezentovat. Ale cosi jej chyba. Ten user interface este nie je uplne idealny, a bolo by super keby fungovalo prepojenie na DB. Zatial to uklada a cita iba do/zo suboru.

Spolu s dvoma ostatnymikolegami, ktorí pracuju na tejto casti s vami, ste si dali zavazok, ze to zvladnete. Kedze ste dobri programatori a designeri, uz na zaciatku ste sa rozhodli, ze si vytvorite niekoľko kniznic, ktoré budu obsahovat spolocne funkcie (resp. objekty), napr. aj na pracu s DB.

Jeden vas kolega vsak nahle ochorie, a musi ostat doma. Nechce vas vsak nechat v stychy, a preto si vezme pracu domov a pokracuje off-line. Vy spolu s tretim kolegom pracujete a modifikujete spolocnu kniznicu v praci (staru ste uz samozrejme prepisali). Spolocne sa vsak synchronizujete, aby ste si nesahali do svojho kodu. Robota ide celkom fajn...

Zatial vas chory kolega pokracuje na svojej casti doma, ale obcas urobi zasah aj do spolocnej kniznice, pretoze obsahuje nejake chyby. Na konci tyzdna sa uz citi lepsie, a rozhodne sa ist na vikend do prace. Vy ste sa rozhodli, ze si date na chvilu pokoj a oddychnete si. Vas kolega v praci si neuvedomi, ze vy ste zatial mohli zmodifikovat tu kniznicu, a preto ju prepise svojou verzio, v domnienke ze vsetko je O.K., resp. ze je to dokonca lepsie akopred tym, kedze uz nejake bugy odstranil.

Na druhy tyzden, si vy pokracujete v praci, bez toho aby ste si vsimli, ze sa zmenila verzia kniznice. Mate uz len tri dni na dokoncenie. Preto sa rozhodnete, ze system trosku otestujete pred samotnym prezentovanim.

Zrazu sa vsak system zacne spravat divne, pada to co predtym chodilo a cele je to nejaku cudne. Zacnete debugovat a zistite, ze zmeny, ktoré ste robili su zraz fuc. Vasa cela tyzdnova robota je v cudu. Nastastie mate nejku zalohu so zaciatku minuleho tyzdna. Nie

je sice uplna ale obsahuje aspon nejake zmeny, nuz lepsie ako nic... Prepisete preto aktualnu verziju vojou starou a pokracujete v praci. No neostava vam vela casu, a uz to nestihate dokoncit. Do prace dojde aj vas chory kolega, ktory si chce otestovat svoju cast (to uz mate len jeden den do predvadzania).

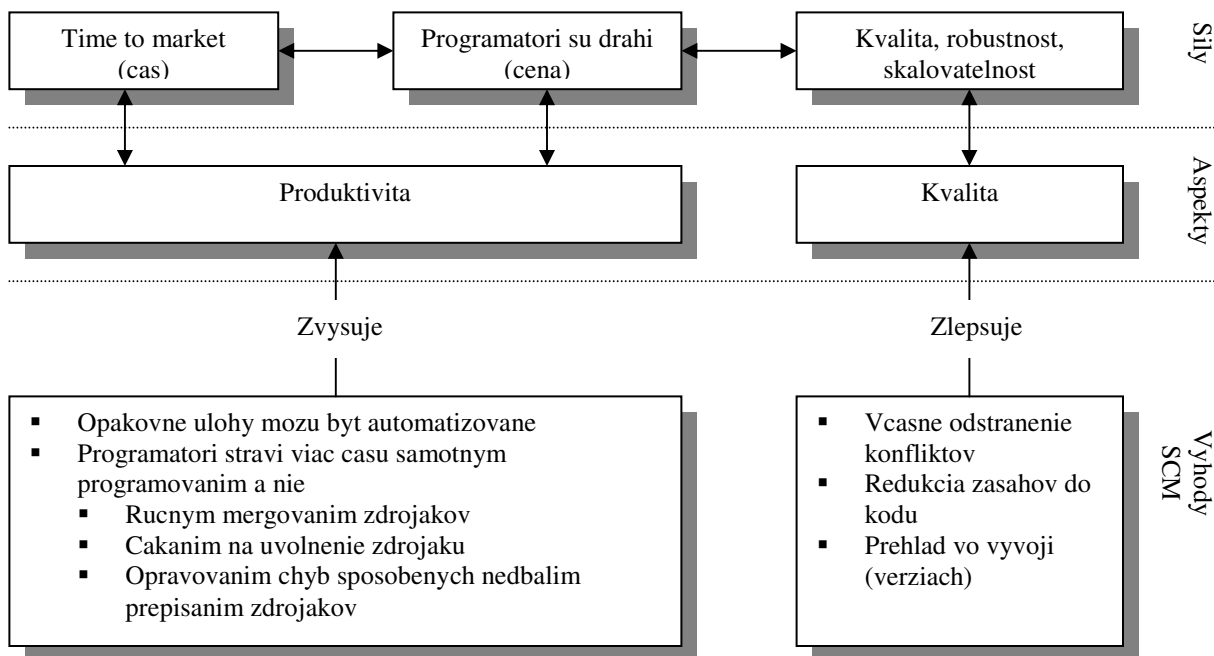
Zrazu zisti, ze jeho rutiny padaju, a nevie preco, ved to uz chodilo. Pozrie do zdrojakov, a zisti ze jeho zmeny su fuc. Onedlho spolocne zistite, co sa stalo a zacnete riesit krizovu situaciju. Uz nie je cas dorabat nove veci, uz to potrebujet len spojzdnit. Zacnete preto do kou vkladat stare veci, len aby to chodilo. Ale kedze mixujete stare s novym, neustale to pada. Robite este zufale pokusy, stravite nad tym celu noc, ale npomohlo to, system stale pada a uz sa to neda ani nstartovat. A stara verzia je samozrejme nenavratne stratená. Prezencaci sa blizi a vy nemate nic... !?!

**Moral of the story** : Pomohla by sprava zdrojoveho kodu (Source Code Management)!

### 1.3 Teoria

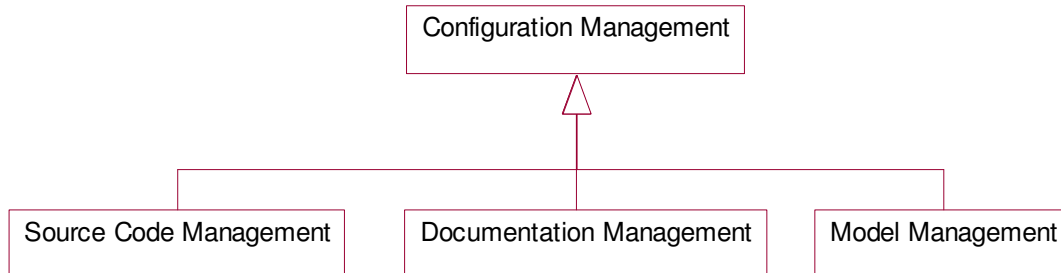
Zdrojovy kod je jednym z dolezityh vysledkov pri vyvoji softveru, preto by sme sa on mali starat a starostlivo ho spravovat.

Zamyslime sa nad silami (externymi faktormi), ktore nas tlacia pri vyvoji softveru a ich suvis so SCM (alebo CM)...



Doposiaľ sme hovorili o zdrojovom kóde (zdrojakoch), ale súčasťou práce vyvojárov je nielen kód, ale aj dokumenty, modely, testovacie procedúry, atď. (voláme ich aj artefakty)

Takže keď urobíme zovšeobecnenie tak máme okrem SCM aj Document Management, Model Management,...



...a teda dostávame sa ku Configuration Managementu

### **1.4 Co je to Configuration Management (Správa konfigurácii)?**

Je to súbor pravidiel, zodpovedností, úloh/cielov a nástrojov na správu konfigurácii (teda zdrojov, dokumentácie, modelov, atď).

Pozor nie sú to len "nástroje" ako si často mnohí ľudia mylne myslia!

Tu je definícia z Ovum report (špecializovaný na CM):

"SW-CM is a disciplined approach  
to managing the evolution of  
software development and maintenance practices,  
and their software products"

(Burrows, George, Dart, Ovum Report 1996)

### **1.5 Co umožňuje CM a prečo používame CM**

Ciastočne sme zodpovedali už na začiatku...

- Správa projektových artefaktov (ich verzie s metadátami - autor, dátum vytvorenia, metriky)
- Správa komplexných produktových verzii (viacero verzionovaných objektov)
- Evidencia a možnosť znovuvytvorenia v minulosti uvoľnených (released) verzii produktu
- Podpora zabezpečenia informácií (security) - kto&co
- Správa chybových hlásení a požiadaviek na zmenu (Problem & Change Request Management)
- Podpora a koordinácia tímovej práce (zefektívnenie práce a zabránenie konfliktom).
- Podpora projektového manažmentu (podpora pri plánovaní a riadení).

## 1.6 Ulohy CM

### 1.6.1 Sprava vsetkych artefaktov

Co sa spravuje?

- Source, Kniznice, Executables, Dokumenty, Protokoly, Testovacie data, Nastroje, Chybove protokloly, Poziadavky na zmenu (changerequests)
- A k tomy aj metadata (datum vytvorenia, autor, metriky...)
- Zavislosti medzi jednotlivymi verzionovanymi objektami

### 1.6.2 Changemanagement, Releaseplaning

- Metodické spracovanie chybových hlásení, požiadaviek na zmenu...
- Planovanie a vytváranie nových verzii produktu...
- Automatizácia produkcie verzii produktu
- Redukcia chybovosti

### 1.6.3 Podpora timovej prace

- Zabranenie konfliktom
- Verbesserung der Kommunikation
- Zabranenie nebezpečným nekonzistenciám
- Podpora veľkých tímov
- Distribúvaný vývoj (na fyzicky rozdielnych miestach)

### 1.6.4 Zakladne pojmy

- Verzionovaný element(objekt)
- Version tree
- Konfigurácia (Build, Release)
- Check-In
- Check-Out
- Merge
- Lock
- View
- Sand box

### 1.6.5 Merging versus Locking

Mame dve tradičné stratégie:

Lock-check out-change-check in    alebo    Change-lock-check out-merge-check in

V prvej stratégii môže dochádzať často k zdržaniam, keď jeden programátor čaká kým source uvoľní iný. Je to síce bezpečné, ale neefektívne. Poznáte to, objavíte nejakú chybu, chcete to opraviť, tak si poviete za hodinku to je, ale z hodinky byvajú hodiny, z hodín dni,...

Najčastejšie vznikajú kolízie v interfacesoch (\*.h). A častokrát dochádza ku kolízii aj u jedného programátora...

Druha strategia je zalozena na automatickom mergovani. Ale funguje to vobec? Nuz, nie je to foolproof, ale skusenosti ukazuju, ze vo vacsine pripadov to funguje dostatočne, a castokrat nie je potrebna ani asistencia. Asistovat treba v pripade napr. ze jeden koder nejaku funkciu vymazal a druhy ju zmodifikoval (subezne)...

Pri tejto metode, je dobrým pravidlom, ze sa vychadza z poslednej stabilnej verzie, nie z poslednej verzie. Zabrani sa tým zanasaniu neziaducich chyb...(infecting sandbox).

### **1.7 Koncepty na ktore treba pri CM mysliet**

Tu su heslovite popisane koncepty, na ktore je potrebne mysliet pri definovaní a zavadzani CM. Je to zatiaľ iba draft.

#### **1.7.1 Subor pravidiel**

- Co sa spravue
- Ako sa to spravuje
- Kto to sprauje
- Ako sa postupuje pri vytvarani novej verzie (resp. modifikacii starej)
- Ako sa postupuje pri konfliktoch

#### **1.7.2 Zodpovednosti/Roles**

- Uzivatske skupiny
- Prava na citanie, modifikaciu vytvoreie novej verzie, modifikaciu metadat
- Role pre notifikaciu pri zmene

#### **1.7.3 CM Plan**

- Ake verzionovane elementy
- Aka bude vnutorna struktura (napr. adresare pre moduly,...)
- Zivatelske prava
- Identifikacia konfiguracii produktu (napr. labels)
- Co sa ma kedy produkovat a za akych podmienok (suvis s procesom)

#### **1.7.4 CM Zodpovednosti**

- CM Koordinator
- CM Spravca
- CM Pouzivatel

#### **1.7.5 CM infrastruktura**

- Nastroje (nasdenie, konfigurovanie)
- Skolenia
- Automatizovanie uloh (skripty, makra)