KIV/ASWI 2007/2008

Konfigurační management

- Co je SCM
- Správa změn
- Správa verzí
- Řízení sestavení

You spend more time coordinating with (and tripping over) the other team members than you do programming.

-- W.Babich

Výchozí problém

- Při vývoji produktu ve více verzích a/nebo ve více lidech je nutné zamezit zmatkům při provádění změn na jednom objektu (dokumentu, souboru, tabulce)

klasifikace zmatků

- *Separátní kopie* ⇒ problém: *dvojí údržba*
  - každý má svou kopii, kterou nesdílí s ostatními
- Jedna *sdílená kopie* ⇒ *konfliktní změny*
  - kopie putuje mezi vývojáři, pracuje na ní vždy jen jeden
- *Soukromé kopie* + sdílený originál ⇒ *překrývání změn*
  - originál ve sdíleném adresáři, pracuje se na lokálních kopiích, po úpravách se kopie překopíruje zpět

Co je Konfigurační management

- Proces identifikování a definování prvků systému, řízení změn těchto prvků během životního cyklu, zaznamenávání a oznamování stavu prvků a změn, a ověřování úplnosti a správnosti prvků [IEEE-Std-729-1983]
  - » „jak vytvářet, sestavovat a uvolňovat produkt, identifikovat jeho části a verze, a sledovat změny“
  - Software Configuration Management (SCM)
  - řízení konfigurace, konfigurační řízení

Prvek konfigurace

- *Prvek konfigurace* = konstituující složka systému
  - » různé typy: zdrojový soubor, dokument, model, knihovna, script, spustitelný soubor, testovací data, …

- Ve správě SCM
  - ☞ ví se o jeho existenci, vlastníkovi, změnách, umístění v produktu
  - atomický z hlediska identifikace, změn
  - jednoznačně identifikovatelný: např. MSW/WS/IFE/SD/01.2
    - » typ prvku (dokument, zdrojový text, testovací data)
    - » označení projektu
    - » název prvku
    - » identifikátor verze

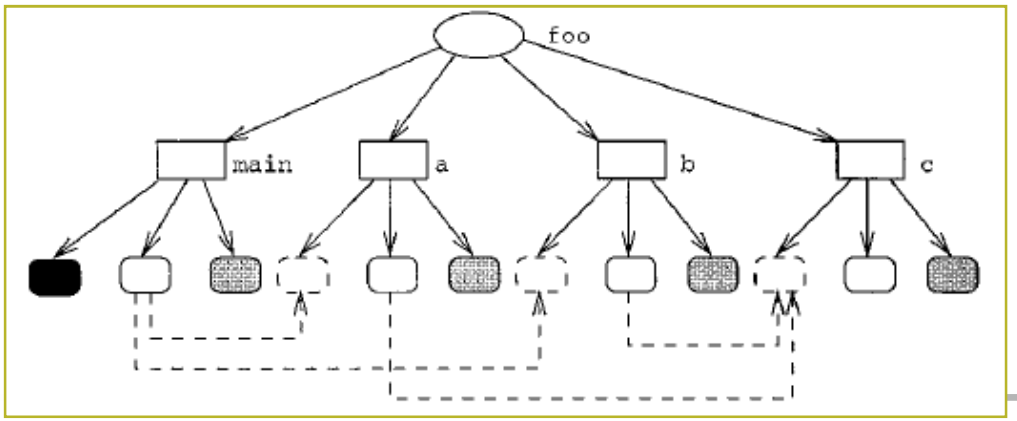
Konfigurace

- *SW konfigurace* = souhrn prvků konfigurace reprezentující určitou podobu daného SW systému
  - » příklad: „první kompilovatelná verze programu XY pro Linux"
  - ☞ v konfiguraci musí být vše, co je potřebné k jednoznačnému opakovatelnému vytvoření příslušné verze produktu
    - » včetně překladačů, build scriptů, inicializačních dat, dokumentace
  - může být jednoznačně identifikovatelná

- *Konzistentní konfigurace* = konfigurace, jejíž prvky jsou navzájem bezrozporné
  - » příklad: zdrojové soubory jdou přeložit, knihovny přilinkovat

Popis konfigurace

- Struktura produktu jako množiny prvků a jejich vzájemných vztahů
- Prvek = výsledek sw procesu
  - různá granularita a reprezentace (soubor/modul/deklarace)
- Vztahy
  - celek-část, master-dependent → určují strukturu a závislosti
  - *zdrojový-odvozený* → určují způsob produkce, tj. *build* produktu

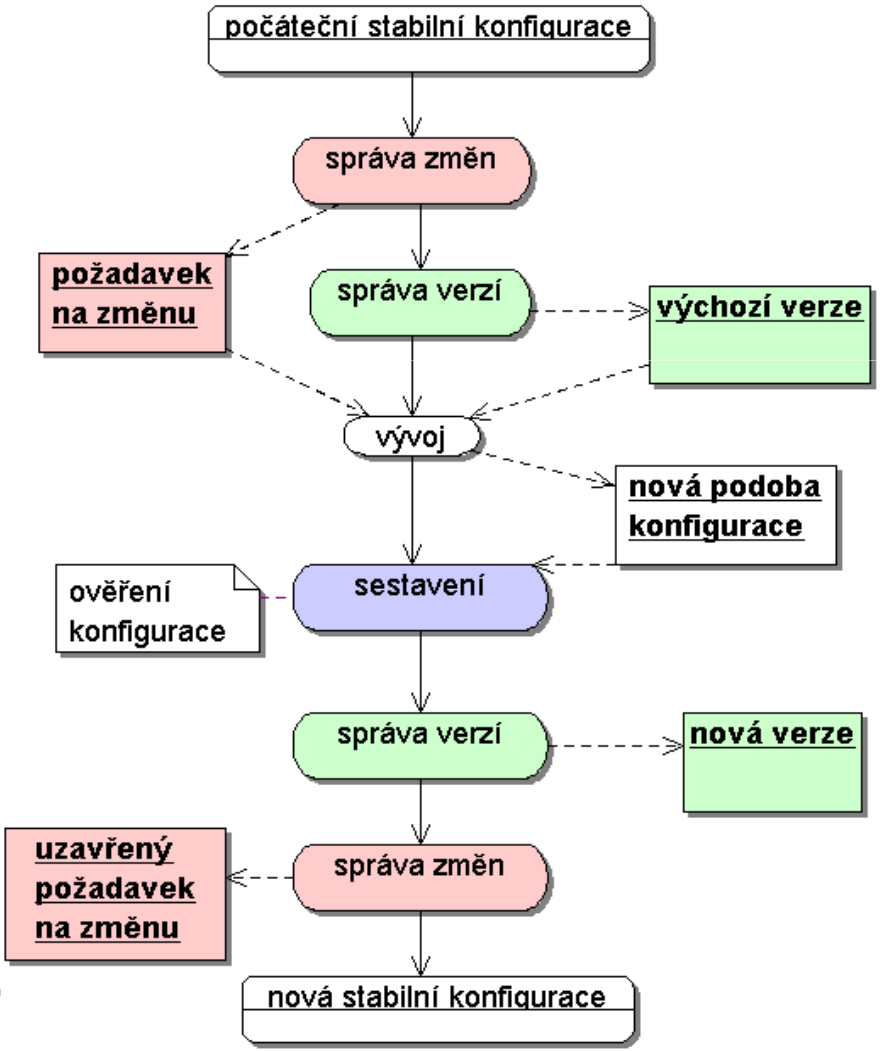


Úlohy SCM

- Určení a správa konfigurace *cfg identification and control*
  - určení (identifikace) prvků systému, přiřazení zodpovědnosti za správu
  - identifikace jednotlivých verzí prvků
  - kontrolované uvolňování (release) produktu
  - řízení změn produktu během jeho vývoje
- Zjišťování stavu systému *status accounting*
  - udržení informovanosti o změnách a stavu prvků
  - zaznamenávání stavu prvků konfigurace a požadavků na změny
  - poskytování informací o těchto stavech
  - statistiky a analýzy (např. dopad změny, vývoj oprav chyb)
- Správa sestavení (build) a koordinace prací *release management*
  - určování postupů a nástrojů pro tvorbu spustitelné verze produktu
  - ověřování úplnosti, konzistence a správnosti produktu
  - koordinace spolupráce vývojářů při zpracování, zveřejňování a sestavení změn

Aktivity SCM v cyklu vývoje

- Správa změn
- Identifikace a správa verzí
- Sestavení



SCM: správa změn

Change management

Change is inevitable,
except from vending machines.

Správa změn

- Problém: Jak zvládat množství požadavků na úpravy produktu (opravy, vylepšení)? Jak poznat kdy už jsou vyřešeny? Jak dohledat, co bylo změněno?
  - » změnové řízení, *change management*
- Nutný striktní postup akcí
  - vyřešení prioritních, udržení konzistence a stability
  - informovanost o změnách, prevence duplikování práce
- Význam ve všech fázích ŽC
  - evidence požadavků během jejich sběru
  - přiřazení příslušné práce během vývoje
  - hlášení a opravy chyb nalezených při testování
  - úpravy produktu během provozu a údržby

Problémy a změny

- *Hlášení problému (issue report)* = popis nalezeného nedostatku nebo požadovaného vylepšení
  - strukturovaný dokument
    - » „bug report" , „ticket report", „RFE", …
  - obvykle výsledek QA aktivit nebo od uživatele
- *Požadavek na změnu (change request)* – popisuje změny které se mají provést na prvku/prvcích konfigurace
  - 1 hlášení problému $\Rightarrow$ 1..N požadavků na změny
- Někdy (často) spojovány do jednoho dokumentu
  - » hlášení problému

Detaily hlášení problému

dress code: informal × formal

- Při vytvoření
  - nutné náležitosti: shrnutí (+ id, autor, datum)
  - popis, co nejpřesnější
    - » jak chyba vznikla
    - » jak reprodukovat
  - screenshot, vzorek dat, …
  - závažnost, priorita (odhad)
  - informace o použitém software
  - konfigurace, OS, knihovny
    - » včetně identifikace verzí
- Při vyhodnocení
  - závažnost, priorita
  - odhad pracnosti
    - » přímé (kód), návazné (doc)
  - komponenta, verze
  - závislosti („meta-bug“)
  - zodpovědný vývojář
- Po uzavření
  - shrnutí, zdůvodnění
  - skutečná pracnost
  - výsledná revize souborů/aplikace

Hlášení problému: příklad

```
Package: hello
Version: 1.3-2
Severity: serious

When I invoke `hello' without arguments from an ordinary
shell prompt it prints `goodbye' instead of `hello, world'.

 $ hello
 goodbye

I am using Debian 1.1, kernel version 1.3.99.15z
and libc 5.2.18.3.2.1.3-beta.
```

» již uzavřené hlášení a neformální seznam chyb

Životní cyklus změny

- vytvoření/přijetí
  - » přidělení ID
- vyhodnocení
  - » možná řešení, jejich dopady a odhad pracnosti
  - » doplnění
- rozhodnutí
  - » způsob vyřízení (vyřešit – odmítnout – duplikát – odložit)
  - » závažnost (kritická chyba – problém – vada na kráse – vylepšení), priorita (vyřídit okamžitě – urgentní – vysoká – střední – nízká)
- zpracování
  - » vygeneruje příslušné požadavky na změny
- uzavření (nejprve všech požadavků na změny)
  - » → build: ověření konzistence; → verzování: vytvoření nové baseline
  - » informovat zadavatele hlášení a další zájemce

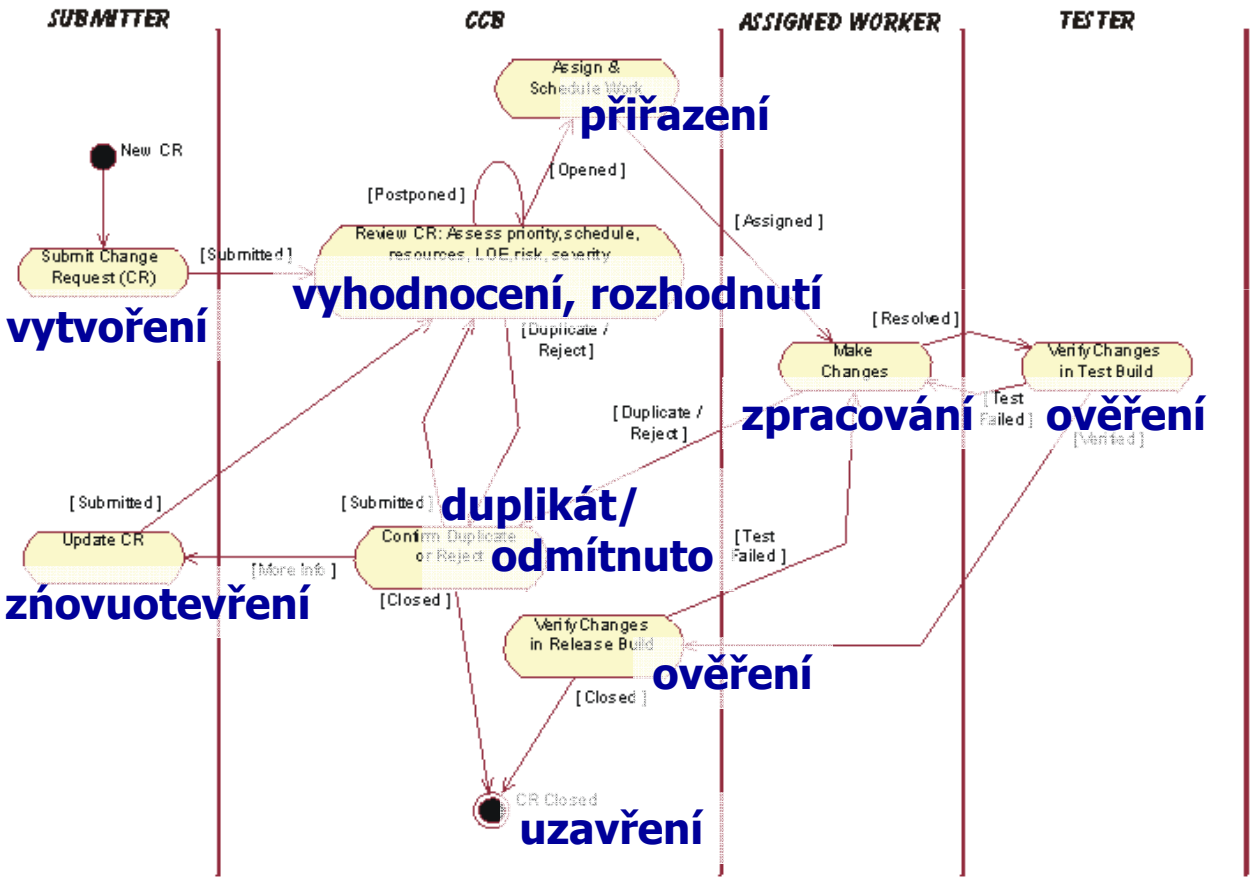
Cyklus správy změn

Požadavek na změnu

- vytvořený
- prováděný
- schválený
- ověřený
- uzavřený
- znovuotevřený
- odmítnutý

Během provádění

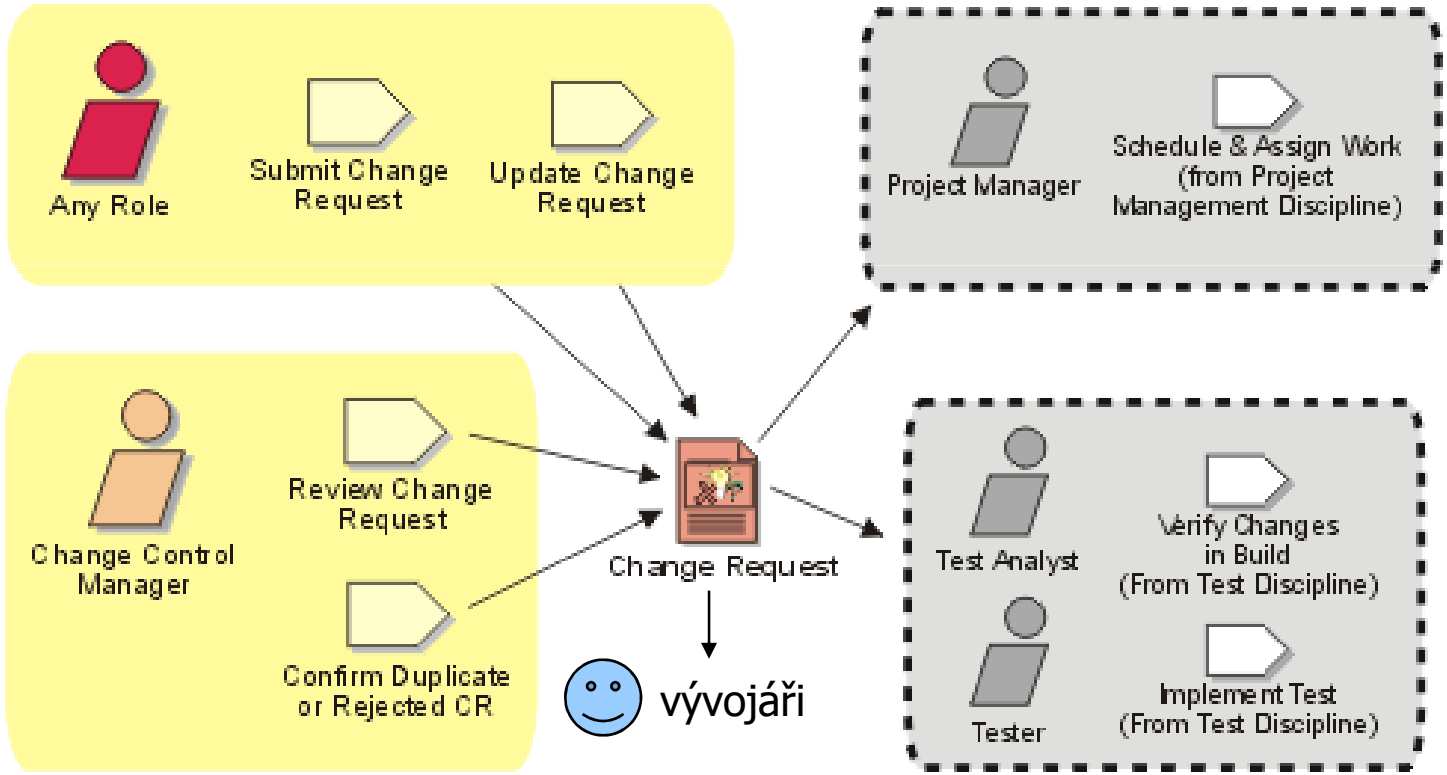
- znovuotevření problémů
- vygenerování nových hlášení



Případy nouze

- Kdy porušit pravidla (proces)
  - marginální oprava těsně před release
  - vyřešení problému u zákazníka
- Pravidla pro porušování pravidel
  - je jasný (dlouhodobý) přínos výjimky
  - nekamuflovat
    - » každý má vidět, že nebyl dodržen standardní proces
    - » zdůvodnitelné, proč se tak stalo
  - zpětně zdokumentovat provedené akce
    - » vytvořit hlášení problému, popsat provedené změny
  - opakované porušení musí vést k úpravě procesu
    - » učit se z toho, co život přináší

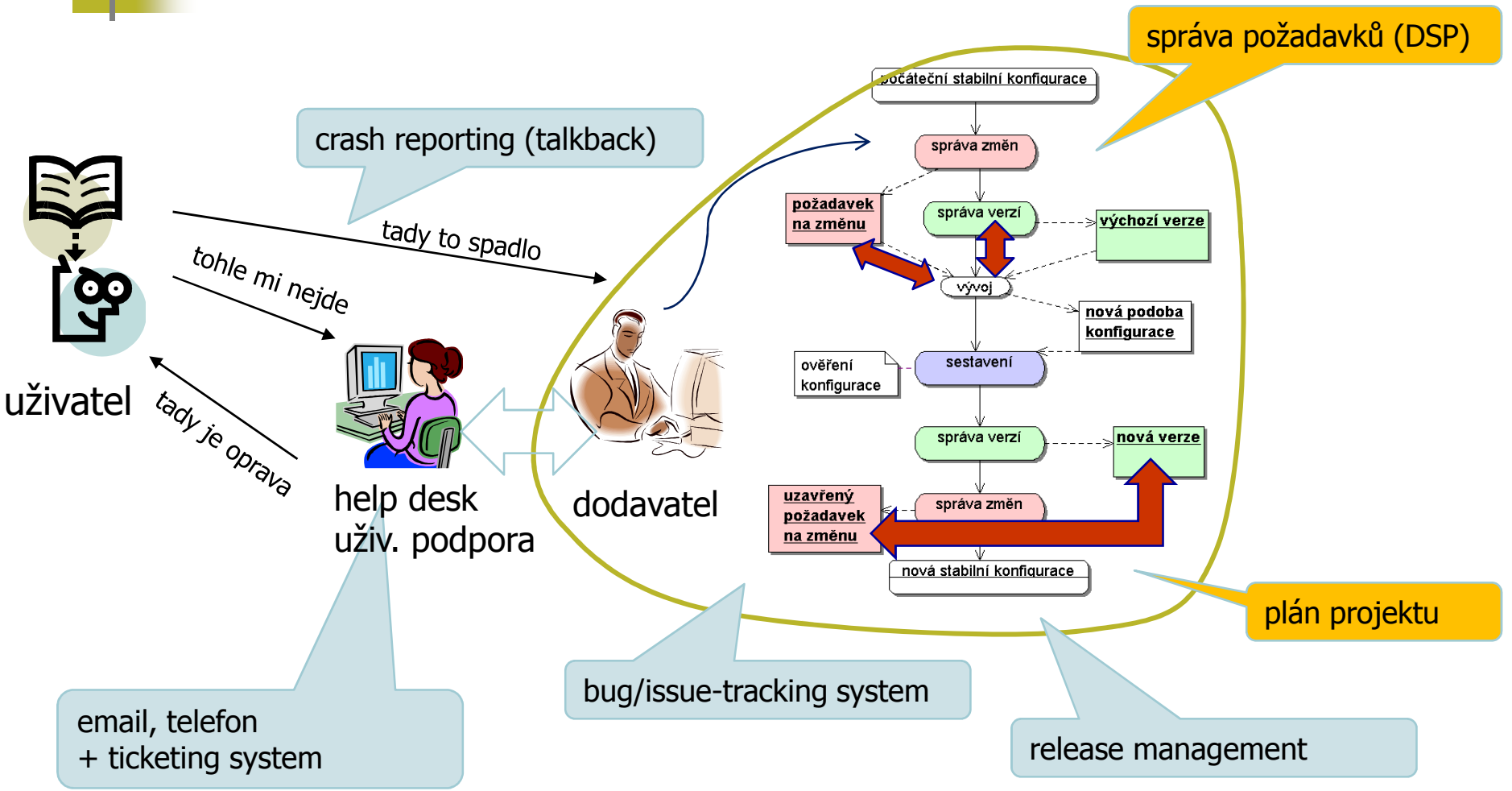
Role ve správě změn



Change Control Board

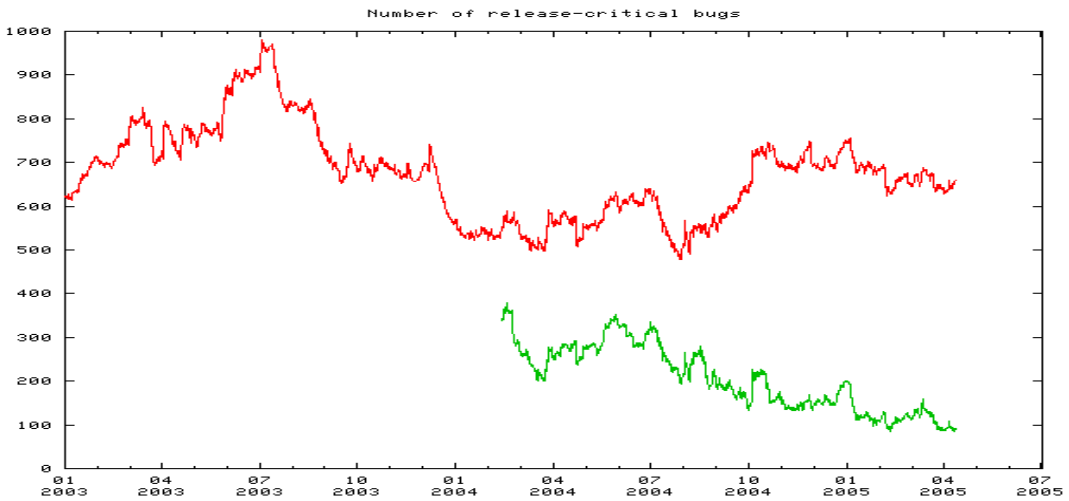
- Skupina členů projektu, která má zodpovědnost za změnové řízení
  - vyhodnocování a schvalování hlášení problémů
  - rozhodování o požadavcích na změny
    - » CCB může významně ovlivňovat podobu a chod projektu
  - sledování hlášení a požadavků při jejich zpracování
  - koordinace s vedením projektu
- Složení CCB
  - jedinec – vývojář, QA osoba
  - tým – technické i manažerské role
    - » vhodné pokud má změna mít velký dopad; znalost účelu produktu

Správa změn není izolovaná aktivita



Správa změn a řízení projektu

- Kritéria dodávky (*release management*)
  - časový termín
  - implementovaná funkčnost / vlastnosti
  - kvalita produktu
- Jak určit termín podle kvality?



By Severity	Open	Resolved	Closed	Total
feature	463	62	581	1106
trivial	30	6	92	128
text	24	7	92	123
tweak	58	12	152	222
minor	318	132	721	1171
major	116	54	414	584
crash	12	7	50	69
block	7	12	89	108

příklad z debianu: počty chyb a přehledový seznam

Správa změn a verzování

- Cíl: trasovatelnost
- Vyhodnocení požadavku: jaké verze se týká
  - » uživatelská verze
  - » interní verze ze správy verzí
- Uzavření požadavku
  - » do BT výsledou verzi
  - » do správy verzí ID vyřešeného požadavku

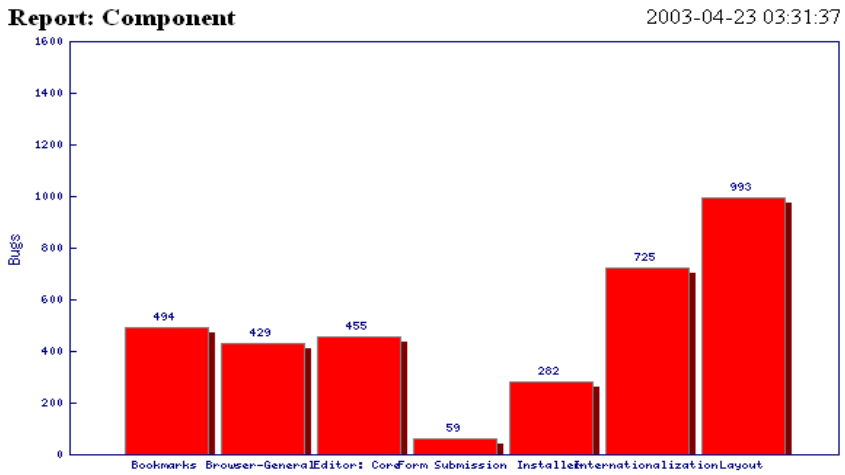
```
Fixed in CVS.

core/print_api.php ->1.126
core/project_hierarchy_api.php ->1.5
core/user_api.php -> 1.95
graphs/graph_by_release_delta.php -> 1.9
```

Správa změn a údržba

- Provoz produktu („uživatelská podpora“)
  - » vs. aktivní vývoj
  - » nutná o to větší pečlivost při úpravách
- Hlášení problému
  - » oprava
  - » garance, servisní smlouva / vícepráce
- Požadavek na vylepšení
  - » ihned ⇒ vícepráce
  - » naplánovat do přírůstku

Systémy pro správu změn

- Bug tracking (BT) systémy
  - evidence, archivace požadavků
  - vyhledávání
  - přehled, reporty, grafy, statistiky
  - sledování stavu požadavku
- Realizace
  - emailové, webové
  - klientské



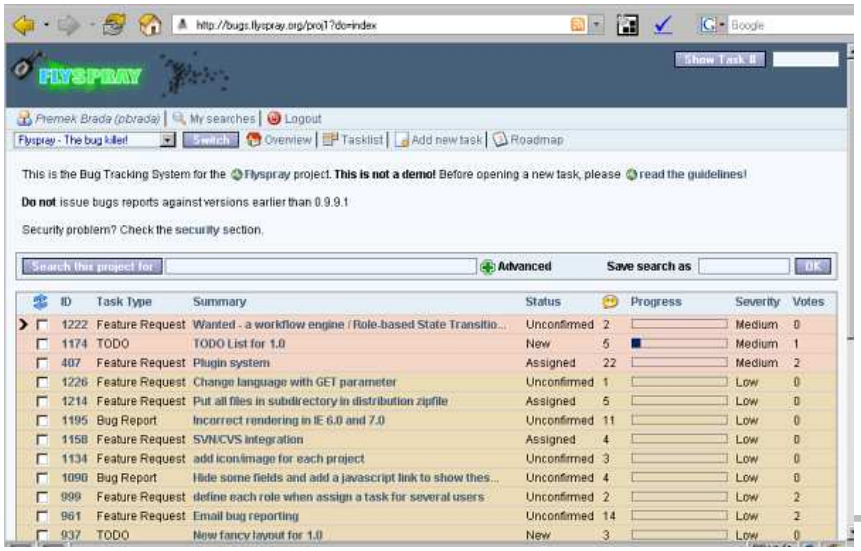
Struktura projektu

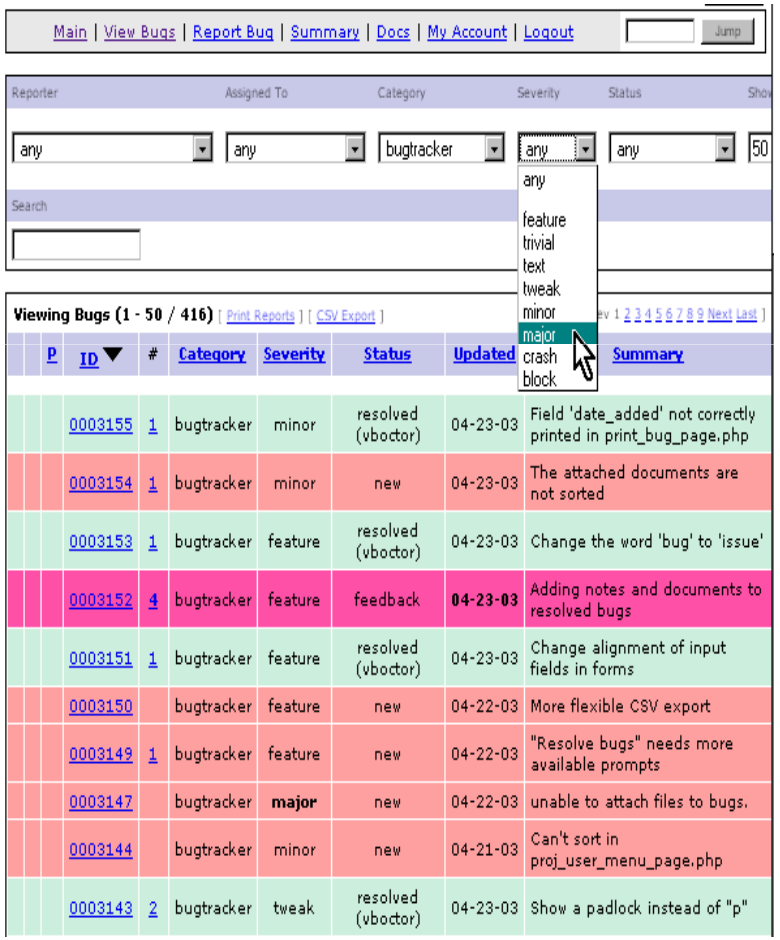
- Název
- Kategorie hlášení
- Úrovně závažnosti, priority
- Verze produktu, operační systém

- Přidělení vývojáři

Flyspray, Mantis

- Jednoduché
- Snadná instalace
- Webové rozhraní
  - emailová notifikace





Bugzilla

- Robustní
- Velké projekty
  - mozilla
- Konfigurovatelná

- Příklady: vyhledávání > seznam > detail > historie

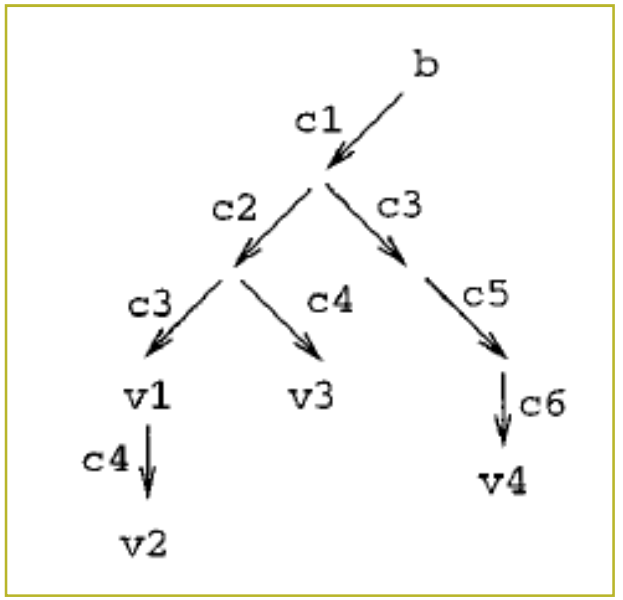
SCM: správa verzí

Správa verzí

- Součást úlohy SCM „identifikace konfigurace“
  - » aby prvek konfigurace mohl být ve správě SCM, musí být identifikovatelný, včetně všech svých podob
- Účel správy verzí: udržení přehledu o podobách prvků konfigurace
  - verze popisuje jednu konkrétní podobu prvku
  - v úložišti jsou skladovány všechny verze
- Druhy verzí
  - historická podoba → *revize* („Word 6.0“)
  - alternativní podoba → *varianta* („Word pro Macintosh“)

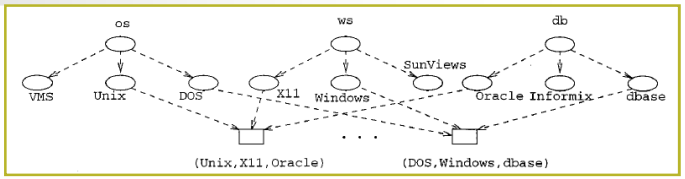
Určení konkrétní verze prvku

- Verzování podle stavu
  - identifikují se pouze prvky
- Verzování podle změn
  - identifikují se (také) změny prvků
    - » changeset
  - výsledná verze vznikne aplikací změn



diff* a *patch
- rozdíl mezi verzemi (text, binární)
- aplikace rozdílu na verzi
  - viz changeset

Možnosti verzování



- Granularita
  - jednotlivé prvky (*verzování komponent*)
    - » konfigurace nemá verzi
  - celé konfigurace (*úplné verzování*)
    - » verze konfigurace indukuje verze prvků
  - verze produktu
- Popis verze
  - *extenzionální* verzování: každá verze má jednoznačné ID
    - » např. 1.5.1 = verze pro DOS, 1.5.2 pro UNIX, 2.1.1 = nový release pro DOS
    - » jednoduchá implementace × problémy při větším počtu verzí
  - *intenzionální* verzování: verze je popsána souborem atributů
    - » např. `OS=DOS and UmiPostscript=YES`
    - » nutné pro větší prostory verzí × potřeba vhodných nástrojů

Informace o verzi

- Identifikátor verze (extenzionální)
  - » klíčový požadavek: jedinečnost
  - „major.minor.micro + build" schéma – např. 6.0.2800.1106 (MSIE 6)
    - » význam pozic identifikátoru a změny hodnot
    - » standardní sémantika: kompatibilita
  - daný nástrojem – např. `$Revision 1.1.2.1$` v CVS, changeset v SVN
    - pro prvky konfigurace
  - kódové či „marketingové" jméno – např. "One tree hill " (Firefox 0.9)
    - pouze pro produkt
- Další meta-data prvku
  - datum/čas vytvoření, autor
  - stav prvku/konfigurace
  - předchůdce (předchůdci)

Prostředí pro verzování: úložiště

- Úložiště (databáze projektu, repository) = sdílený datový prostor, kde jsou uloženy všechny prvky konfigurace projektu
  - centrální místo
  - určitě všechny prvky ve všech verzích
    - » zdrojové kódy
    - » knihovny (přeložené) a kód třetích stran
    - » konfigurační soubory, datové soubory
    - » scripty pro build, testování a instalace
    - » dokumentace, modely, prototypy
    - » odpadkový koš
  - ☞ řízený přístup (udržení konzistence)

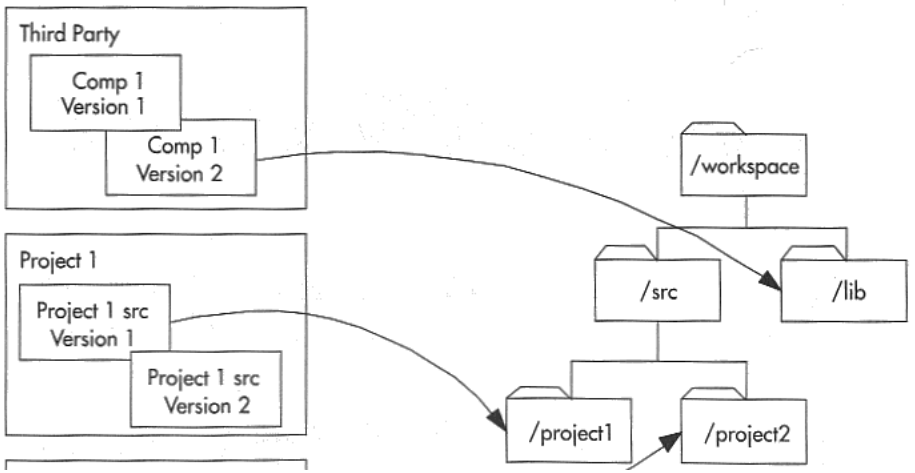
Implementace:
• souborový systém + dohodnutá pravidla používání
• verzovací nástroj
• databáze s rozhraním podporujícím postupy SCM

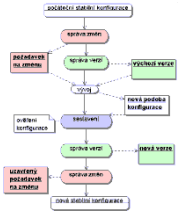
Práce s úložištěm

- Základní operace
  - *inicializace* - vytvoření úložiště, naplnění boostrap verzí projektu
  - *check out* - kopie prvku do lokálního pracovního prostoru
  - *check in (commit)* - uložení změněných prvků do úložiště
  - *zjišťování stavu* - sledování změn v úložiště vs. prac.prostor
- Přístup k zamykání při ci/co
  - read-only pro všechny
  - pesimistický: read-write kopie prvku jen pro pověřeného
  - optimistický: read-write pro kohokoli, řešení konfliktů

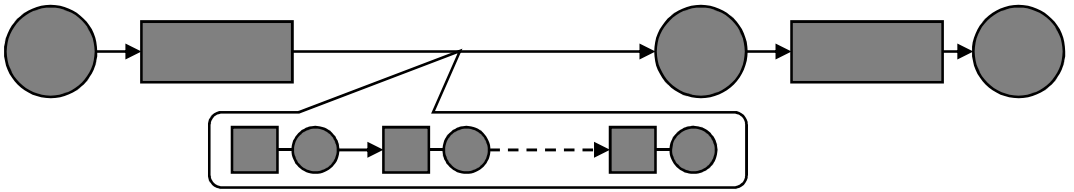
Pracovní prostor

- Pracovní prostor (workspace) = soukromý datový prostor, v němž je možno provádět změny prvků konfigurace, aniž by byla ovlivněna jejich „oficiální" podoba používaná ostatními vývojáři
  - vývojářský (soukromý)
  - integrační (sdílený)





Postup vývoje a verzování



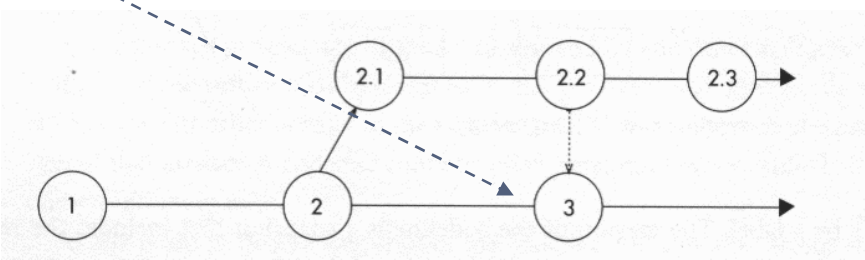
- *check-out* výchozí verze
- vývoj
- lokální testování a opravy
- *check-in* nové verze
- integrační testy a opravy
- check-in nové *baseline*

Kolik? Viz změnové řízení:
- commit per task
- commit per bug/request
- lokální commity

Kam? Viz verzování:
- hlavní vývoj
- větev

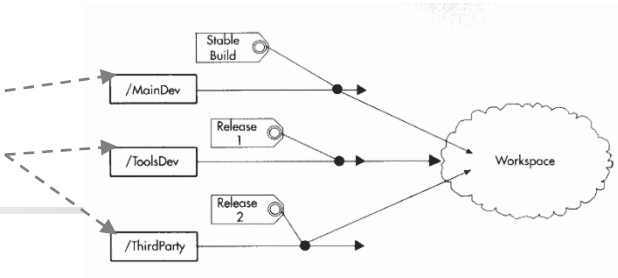
Codeline (vývojová linie)

- Codeline = série podob (verzí) množiny prvků konfigurace tak, jak se mění v čase
  - má přiřazena pravidla práce s codeline (*policy*)
    - » zejména kdy a jak je možno provádět změny
  - *vrchol* codeline obsahuje nejčerstvější verzi



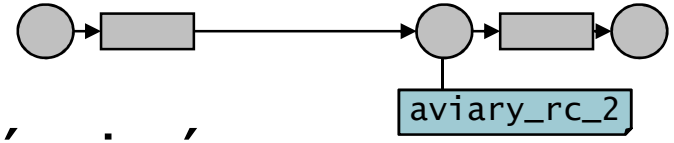
Pravidla linie
- typ prací (vývoj, údržba, release, …)
- pravidla/akce před ci, po checkout, …
- jak a kdy ci, co, branch, merge
- přístup pro osoby a skupiny
- kam (codeline) exportovat změny, odkud importovat
- doba platnosti či podmínky odstavení

- Konfigurace se může skládat z více codelines
  - vlastní projekt
  - knihovní codeline

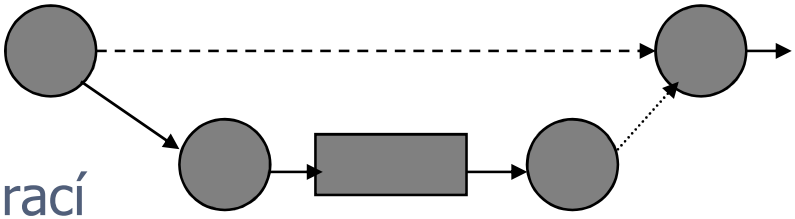


Postup vývoje a verzování (2)

- Tagging
  - označení konfigurace symbolickým jménem



- Paralelní práce
  - » vzájemná izolace paralelních prací

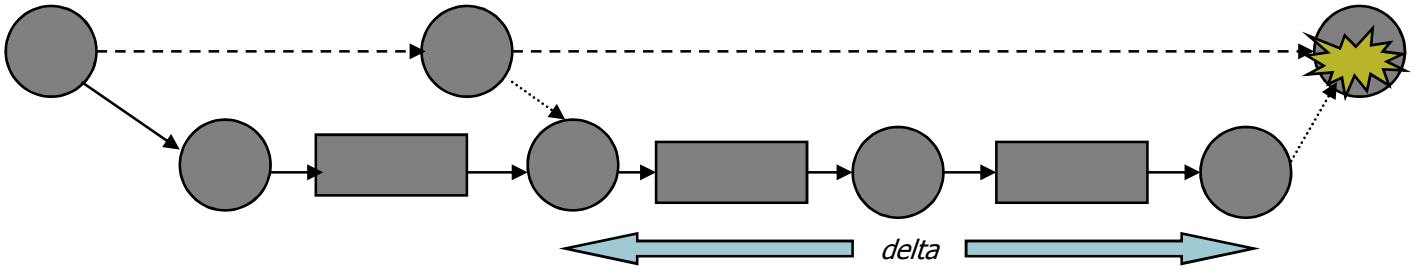


  - oddělení paralelních vývojových linií (varianty)
  - údržba release
  - spekulativní vývoj
    - » cena za izolaci od změn = řešení konfliktů

Větvení a spojování

- Kmen (*trunk*) – hlavní vývojová linie
- Větev (*branch*) – paralelní vývojová linie
  - operace vytvoření větve (branch-off, split)

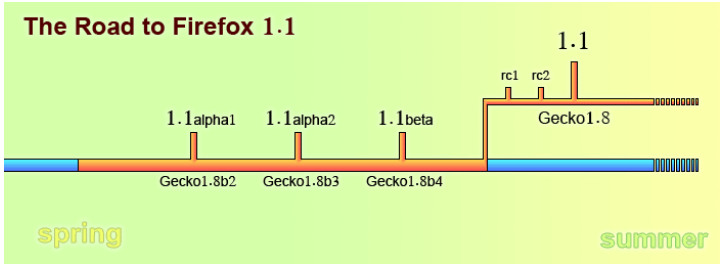
- Spojení (*merge*) – sloučení změn na větvi do kmene
  - slučuje se delta od branch-off nebo posledního merge
  - řešení konfliktů: automatizace vhodná, ale ne vždy možná



Baseline a delta

- Baseline = konzistentní konfigurace tvořící stabilní základ pro produkční verzi nebo další vývoj
  - » příklad: milník „stabilní architektura“, beta verze aplikace
  - stabilní: vytvořená, otestovaná, a schválená managementem
  - změny prvků baseline jen podle schváleného postupu
  - při problémech návrat k baseline
- Delta = množina změn prvku konfigurace mezi dvěma po sobě následujícími verzemi
  - » příklad: přidání sekce „kontext produktu“ do DSP, patch foo.c
  - v některých systémech jednoznačně identifikovatelná

Význačné baseline

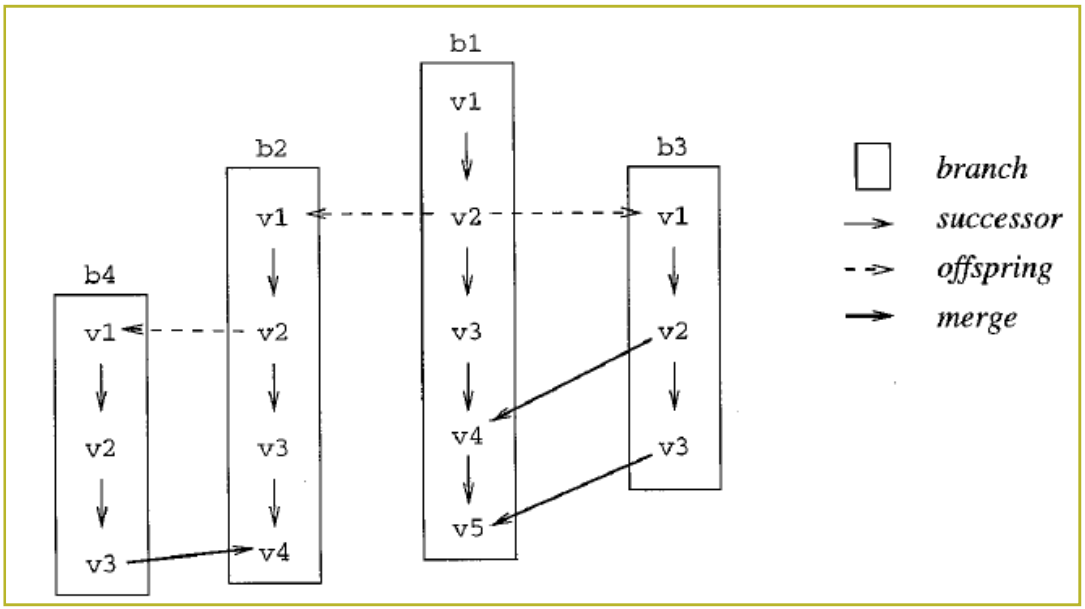


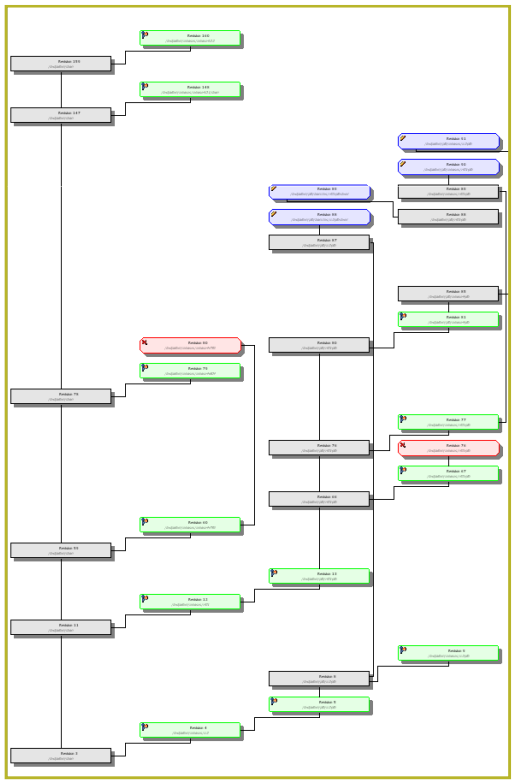
http://www.mozilla.org/projects/firefox/roadmap-1.5.html

- Milníky projektu
  - jednotlivé fáze životního cyklu
    - » vodopád / spirála / iterativní
  - interní release
  - alfa verze = „feature complete", interní testování
  - beta verze = testování u (vybraných) zákazníků
- Release (vydání, sestavení) produktu
  - finální verze dodané zákazníkovi/na trh
- Související techniky
  - » code freeze, stabilizační období

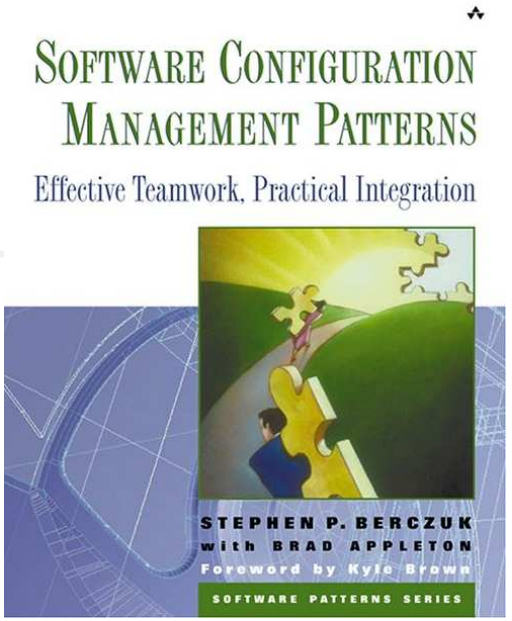
Graf verzí

- Zobrazení vazeb mezi verzemi prvku/konfigurace
  - uzly = verze, hrany = vazby mezi verzemi
  - grafická podoba codeline





Vzory pro verzování



- Hlavní vývojová linie
  - jedna codeline pro průběžný vývoj produktu
- Privátní verze
  - soukromé úložiště pro častější check-in
- Větev pro úkol
  - složitější úpravy s většími následky dělat na větvi
- Check-in pro každý úkol
  - po ukončení práce na jednom úkolu udělat check-in změn
- Větev pro release a jeho přípravu
  - místo code freeze mít samostatnou větev pro release
- Kód třetích stran na větev
  - vlastní větev pro každý kód od externího dodavatele

Nástroje pro verzování

- Ruční verzování
- Základní – správa verzí souborů
  - obvykle extenzionální verzování modulů
  - centrální úložiště
  - ukládání všech verzí v zapouzdřené úsporné formě
  - příklad nástrojů: rcs, cvs, subversion
- Distribuované
  - více úložišť, synchronizace
  - flexibilnější postupy
  - příklad nástrojů: SVK, git, Mercurial
- Pokročilé – integrace do CASE
  - obvykle kombinace extenzionálního a intenzionálního verzování
  - automatická podpora pro ci/co prvků z repository do nástrojů
  - příklad nástrojů: ClearCase, Adele

Co nástroj má umět

- Operace s úložištěm
  - ci, co, add, rename, move, import, export
  - stav prvků, diff, historie změn
- Verzování
  - ci, co, data revize (klíčová slova), branch, merge, značkování
- Podpora týmu a procesu
  - vzdálený přístup
  - konfigurovatelné zamykání a přístupová práva
  - automatické oznamování
  - spouštění scriptů při operacích
  - integrace do IDE, řádkové a webové rozhraní

rcs: Revision Control System

- Správa verzí pro jednotlivé textové soubory
  - UNIX, Windows, DOS
  - extenzionální stavové verzování komponent
- Ukládá (do `foo.c,v` souboru)
  - ☞ historii všech změn v textu souboru
  - informace o autorovi, datumu a času změn
  - textový popis změny zadaný uživatelem
  - další informace (stav prvku, symbolická jména)
- Používá diff(1) pro úsporu místa
  - poslední revize uložena celá
  - ☞ předchozí pomocí *delta* vygenerované diff-em
- Funkce
  - zamykání souborů, poskytování R/O kopií
  - symbolická jména revizí, návrat k předchozím verzím
  - možnost větvení a spojování změn z větví do kmene
- Složky (utility z příkazové řádky):
  - ci, co, rcs, rlog, rcsdiff, rcsmerge

```
head            1.3;
access;
symbols
                VERZE_1_1:1.2;
locks; strict;

1.3
...

1.2
date            2004.01.09.16.44.38;
author brada;                   state Exp;
branches;
next            1.1;
...

1.2
log
@- upraveny sablony na XHTML+CSS2 layou
- pridana fce KivPage.setLang()
- drobne upravy Database.class (cfg hodnoty
@
text
@d10 1
a10 1
 * @@version $Id: sitecfg.inc,v 1.1 2003/12
d41 2
```

cpp: Realizace variant

- C preprocessor umožňuje intenzionální stavové verzování
- Např. chceme variantu foo.c pro případ
  `OS=DOS and UmiPostscript=YES`:

```
/* vlozime definice varianty */
#include "sys-variant.h"
        ...
#if (defined OS_DOS)
#if (defined UmiPostscript)
... /* zdrojovy kod, který odpovida variante */
#else
        ... /* varianta OS=DOS, UmiPostscript=NO */
#endif
        ... /* varianta OS != DOS */
#endif
```

- Definice atributů pro popis variant
  - hlavičkový soubor (centrální místo def. varianty celého systému)
  - parametry příkazové řádky `gcc -DOS_DOS` (např. v Makefile)

CVS (Concurrent Versioning System)

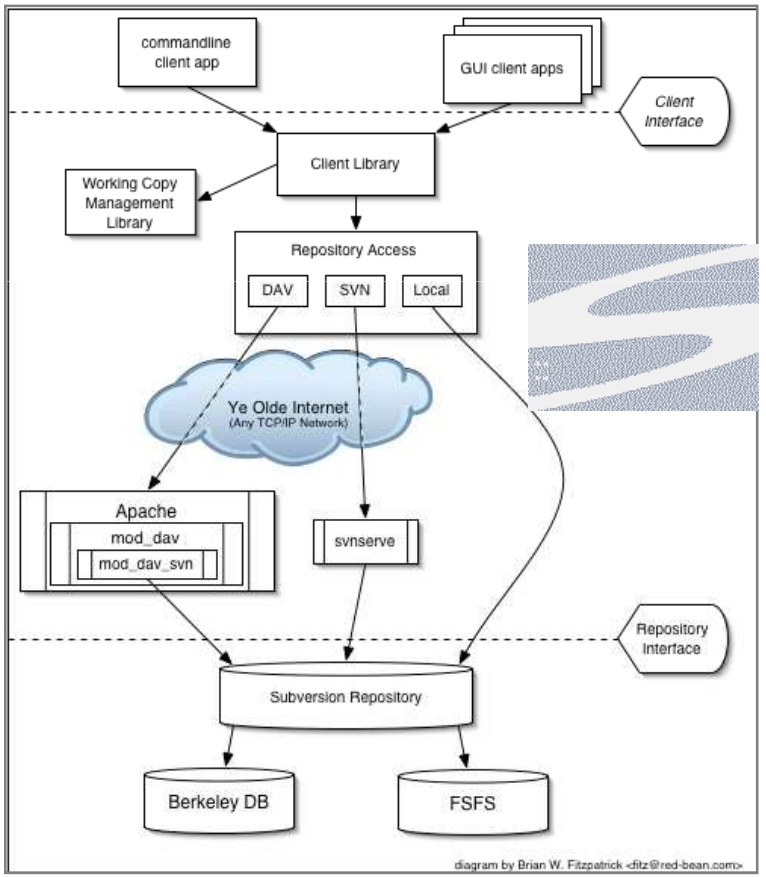
- Práce s celými konfiguracemi (projekty) najednou
- Sdílené úložiště + soukromé pracovní prostory
  - úložiště lokální nebo vzdálené (rsh/ssh, p-server)
  - optimistický přístup ke kontrole paralelního přístupu
    - místo *zamkni-modifikuj-odemkni (RCS)* pracuje systémem *zkopíruj-modifikuj-sluč*
  - zjišťování stavu prvků, rozdílů oproti repository
  - možnost definovat obsah a strukturu konfigurace
  - triggery
  - vše co umí rcs (zejm. klíčová slova)
- Free software
  - původně nadstavba nad rcs
    - používá ,v formát souborů
  - příkazová řádka, grafické nadstavby (UNIX, Windows, web)
  - integrace do mnoha IDE a CASE nástrojů

cvs: Základní dovednosti

- Vytvoření repository (obvykle zvané *cvsroot*)
  - musí být přístupné pro zápis pro všechny členy týmu
  - `cvs -d /usr/local/cvsroot init`
- Inicializace projektu do a z repository
  - `cd workspace/project`
  - `cvs import project MyName Initial`
    - Nyní je vhodné adresář workspace/project celý smazat
  - `cvs -d /usr/local/cvsroot checkout project`
- Ukládání změn do repository
  - `cvs add filename` `cvs remove filename`
  - `cvs commit [filename|pattern]`
  - `cvs update filename`
    - v souboru označí místa, kde nastal konflikt, pomocí `>>>>>>>`
- Administrativní příkazy
  - `cvs status [filename|pattern]`
    - *Up-to-date | Locally Modified | Needing Patch / Needs Merge*
  - `cvs tag značka-verze`

Subversion (svn)

- Následník CVS
  - » Karl Fogel (autor *Open Source Development with CVS*) najatý CollabNet v r. 2000 na vytvoření „lepšího“ CVS
  - bez omezení předchůdce – přejmenování, verzování adresářů, atomický commit, http přístup
  - nové možnosti – binární diff, meta-data, abstraktní síťová vrstva (DAV), čisté API
  - ☞ způsob práce a příkazy velmi podobné CVS

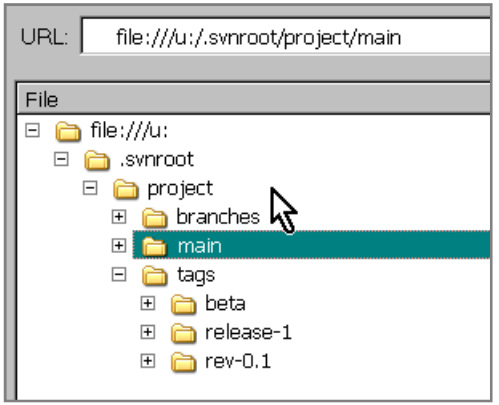


svn: Základní dovednosti

- Vytvoření repository (obvykle zvané *svnroot*)
  - musí být přístupné pro zápis pro všechny členy týmu
  - `svnadmin create <path>`
- Inicializace projektu do a z repository
  - `cd workspace/project`
  - `svn import <repository-url>`
    - nyní je vhodné adresář workspace/project celý smazat
  - `svn checkout <repository-url>`
- Ukládání změn do repository
  - `svn add|remove filename` `svn move oldpath newpath`
  - `svn commit [filename|pattern]`
  - `svn update filename`
    - v souboru označí místa, kde nastal konflikt
- Administrativní příkazy
  - `svn status [filename|pattern]`
  - `svn copy currentpath newpath`

svn: Několik poznámek

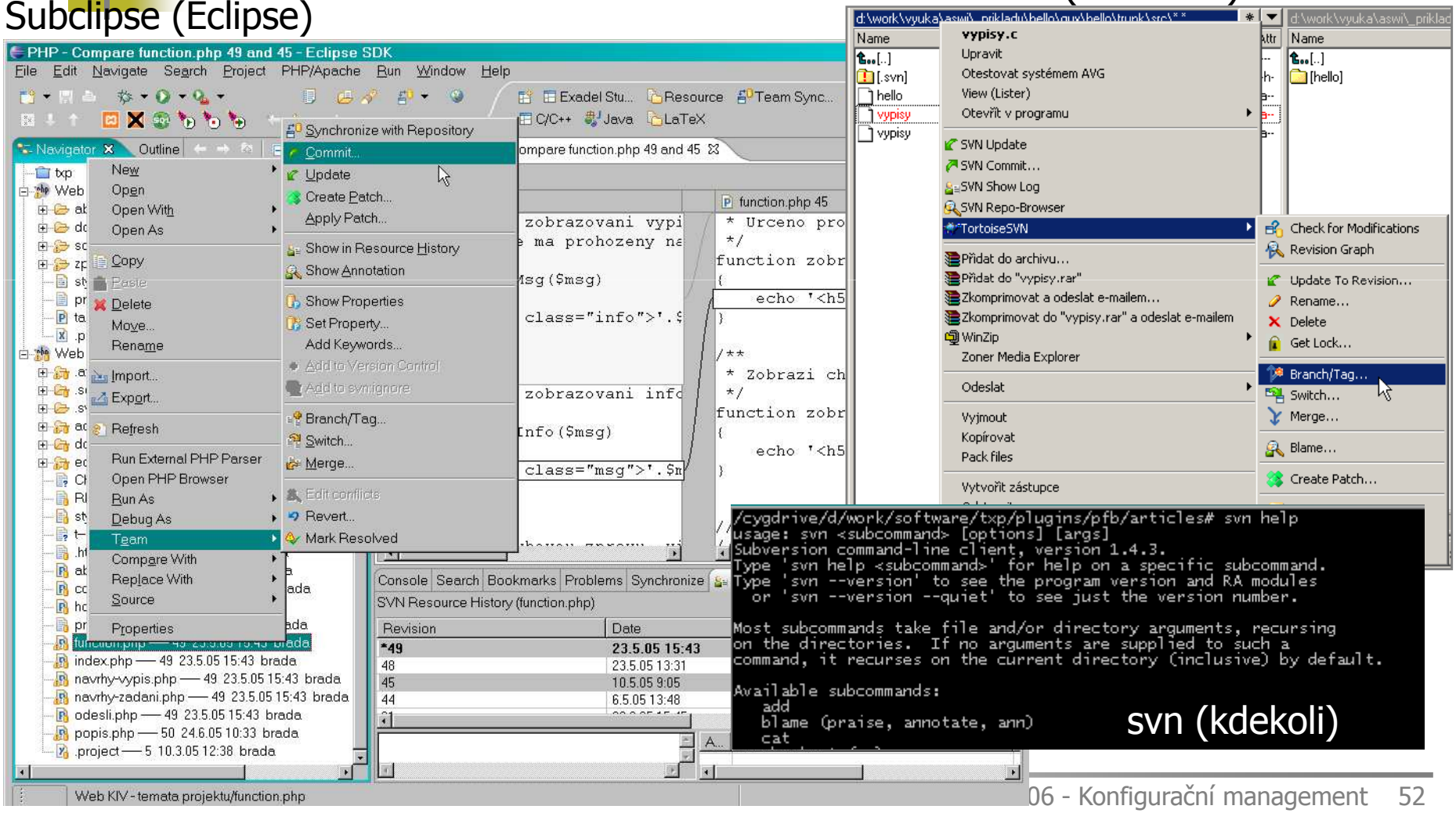
- Identifikace verzí
  - globální kontinuální jednočíselné identifikátory (číslují commit)
  - není koncept „značek" (tag) jako v CVS
- Obecná operace „copy"
  - kopíruje jednu část úložiště/projektu na jiné místo v úložišti
  - význam kopie dle potřeby ⇒ název
    - » značky – označení aktuální/vybrané verze
    - » vytvoření větve
- Doporučená struktura úložiště
  - oddělené adresáře pro kmen, větve a značky



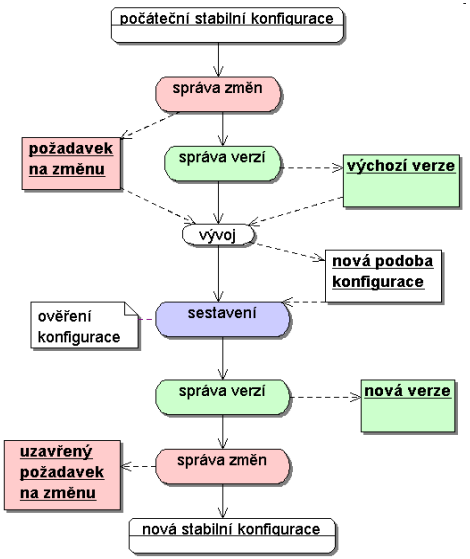
svn: nástroje

Subclipse (Eclipse)

Tortoise SVN (Windows)



SCM: řízení sestavení (build)



Řízení sestavení

- Aktivity podporující transformaci zdrojových prvků konfigurace na odvozené
  - zejména sestavení celého produktu
- Cíl: vytvořit systematický a automatizovaný postup
- Pojmy
  - *build* (též *integration*; proces sestavení, sestavení) – proces a výsledek vytvoření částečné nebo úplné podoby aplikace
  - *zdrojové* a *odvozené* prvky konfigurace

Příklad sestavení

```
/cygdrive/d/tmp/hello# rm -rf *
/cygdrive/d/tmp/hello# svn co file:///d:/data/svnroot/hello.ukazkove/trunk/ .
Restored 'Makefile'
A    lib
A    src
A    src\hello.c
A    src\vypisy.c
A    src\vypisy.h
A    bin
U    Makefile
Checked out revision 46.
/cygdrive/d/tmp/hello# make
gcc  src/*.c -o bin/hello
Preklad OK, spustis pres "make run"
/cygdrive/d/tmp/hello# make test
*** Test spusteni
bin/hello.exe
Hello, world!
*** Test napovedy
bin/hello.exe -h
Program Hello verze 1.
Vypisuje uvitaci zpravu.
Volani: hello
/cygdrive/d/tmp/hello#
```

Postup při vytváření sestavení

- Build process
  - míra formálnosti
  - míra preciznosti
- Kroky
  - (příprava)
  - check-out
  - preprocessing, překlad, linkování
  - nasazení
  - spuštění
  - testování
  - značkování, check-in
  - informování

Vlastnosti sestavení

- Jedinečnost a identifikovatelnost
  - » PROJEKT_v2_build2134_20041236T1954
  - identifikátor jednoznačný, čitelný
  - vytvořitelný a zpracovatelný automaticky (schema pro id sestavení)
- Úplnost
  - tvoří kompletní systém, obsahuje všechny komponenty
- Konzistence
  - vzniklo ze správných verzí správných komponent
    - » tj. z konzistentní konfigurace
- Opakovatelnost
  - možnost opakovat build daného sestavení kdykoli v budoucnu
    - » se stejným výsledkem
- Dodržuje pravidla vývojové linie
  - build odpovídající baseline, zejména release, má striktní pravidla

Součásti prostředí pro sestavení

- Pravidla (neměnit)
  - vývojová linie
  - součásti a vlastnosti sestavení
    - » adresářová struktura, identifikátory sestavení
- Scripty
  - check-out, značkování, check-in
  - preprocessing, překlad, linkování
  - nasazení, spouštění, testování
    - některé nebudou u interpretovaných jazyků, dokumentů apod.
  - informování vývojářů, vytváření statistik
  - vytvoření distribuční podoby (packaging)
- Vyhrazený stroj a workspace
  - „build machine"
  - zejména pro integrační a release sestavení

Typy sestavení

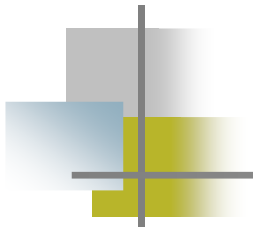
- Co je použito pro sestavení
  - ušetřit čas na překladu
  - *čistý*
  - *úplný*
  - *přírůstkový (inkrementální)* build
- Účel sestavení
  - lokální/neoficiální komponenty povoleny
  - *soukromý*
  - *integrační (oficiální)*
  - *release* build

Postupy pro sestavení

- Základní postupy
  - soukromé sestavení + sdílení součástí
  - integrační sestavení
  - release
- Podpůrné aktivity
  - kusovník (Bill of Materials) a zapouzdřená identifikace
  - zkouška těsnosti (smoke test)
  - regresní testy
  - archivace prostředí
  - balení a distribuce (packaging)

- Obecný cíl: odchytit co nejdříve okamžik kdy „se to rozbilo“

Soukromé sestavení



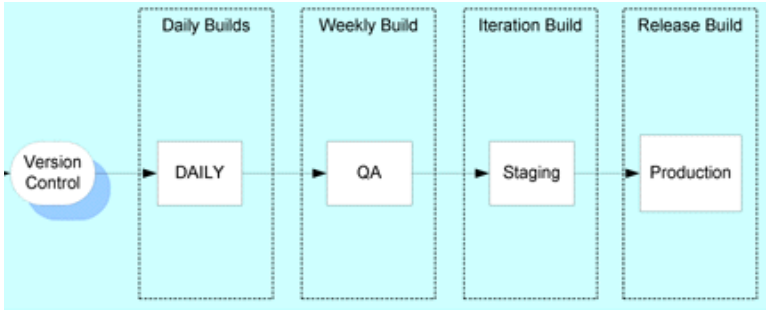
- Cíl: ověřit si konzistenci konfigurace
  - produkt lze sestavit po mnou provedených změnách
  - před check-in (problémy řeším já × všichni)
- Postup: sestavit produkt v soukromém prostoru
  - pomocí sestavení (scriptu) *co nejpodobnějšího oficiálnímu*
    - » postup, verze nástrojů, adresářová struktura
    - » rozdíly ⇒ problémy
  - obsahuje všechny závislé součásti produktu
  - ze zdrojových textů zejména změněné a přímo související
    - » ze správy vzít pro build verzí vše / kromě označeného / pouze označené
- Urychlení průběhu
  - použít inkrementální sestavení, kde je to vhodné
    - » pozor při přidávání souborů, rozsáhlých a/nebo významných změnách
    - » úplné sestavení: dělat na čistém (novém) workspace
  - vynechat postupy pro balení, vkládání info o verzi
  - pomoci si sdílením odvozených prvků (shared version cache)

[At least] make sure that everyone compiles the same way, using the same tools, against the same set of dependencies.

Integrační sestavení

- Cíl: spolehlivě ověřit, že produkt jde sestavit
  - soukromý build nestačí
    - » složité závislosti, špecifiká ve workspace, zjednodušení pro zrychlení
  - úplné sestavení trvá dlouho
- Postup: celý produkt (vč. závislostí) sestaven centrálně, automatizovaným a opakovatelným procesem
  - postup co nejpodobnější sestavení pro release
    - » vždy „na zelené louce" (*clean full build*)
  - maximální automatizace - typicky běží přes noc
  - spolehlivé mechanismy zaznamenání chyb a informování o nich
    - » emailové notifikace začátek, konec, výsledek
    - » web s přehledy a detaily
  - úspěšné sestavení může být označkováno ve verzovacím systému

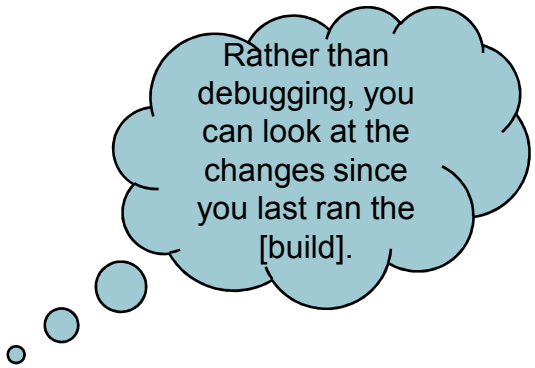
Release build



- Význačné integrační sestavení: dodáno zákazníkovi
  - » může být interní zákazník, např. QA
- Náležitosti release
  - revize/verze konfigurace použité pro sestavení
    - » které prvky, v jakých verzích (nezapomenout na použité knihovny a nástroje)
  - datum vytvoření
  - identifikátor sestavení
  - další metadata
    - » zodpovědná osoba
    - » zdrojová značka konfigurace (z verzovacího systému)
    - » zda prošlo testy, výsledky testů
    - » cesta k logům překladu, testů
  - „marketingová verze"
    - » „Projekt 2.3"

Diskuse o sestavení

- Automatizace, automatizace, autmoatizace, atumoazitace, tmozaicetaua
  - celý proces (jinak bezpředmětné)
    - » check-out, překlad, sestavení, spuštění, logování, check-in
  - plánovač spouštění buildu
  - vytváření čísel/identifikátorů sestavení
  - ukládání metadat do databáze a do verzování
    - » vč. výsledků testů, logů, identifikace sestavení
- Frekvence integračního sestavení
  - čím častěji tím lépe (pro menší Δ je snazší nalezení chyb)
  - kompromis trvání buildu $\times$ frekvence změn $\times$ velikost změn
- Samotné sestavení nestačí
  - viz vzory pro testování dále



Kusovník, archivace prostředí

… a další triky

- Kusovník
  - kompletní seznam prvků sestavení
    - » reprodukovatelnost sestavení kdekoli, kdykoli
    - » zejména při distribuovaném nebo jinak složitém buildu
  - samoidentifikující konfigurace pomůže
    - » znalost verzí bez přístupu k verzovacímu systému
  - viz strojírenství ; automatizace (ClearCase make)
- Archivace prostředí
  - správa verzí objektů, které nejsou v úložišti
    - » nástroje, platformy, hardware, prostředí - identifikovat sestavení
  - klíčové pro dlouho žijící software (např. povinné v letectví)

Nejlepší praktiky: SCM+QA

- Ověřené postupy sestavení pro největší zisk
  - zejména pro iterativní a přírůstkový vývoj

- Daily build and smoke test
  - » denní sestavení a zkouška těsnosti
- Continuous build
  - » soustavné sestavování

Zkouška těsnosti (Smoke Test)

- Cíl: ověřit, že sestavení vytvořilo funkční produkt
  - samotné sestavení toto nezaručuje
  - kompletní testování trvá dlouho
    - » verifikace (bezchybný produkt) a validace (správný produkt)
    - » potřebujeme rychlý základní test, jehož provedení „nebolí"
- Postup: vytvořit testy ověřující základní funkčnost, bez nároku na kompletní otestování
  - odchytí nejkřiklavější chyby, odpustí drobnosti
  - automatizace - spuštění, vyhodnocení
  - spuštění při každém buildu (vč. soukromého)
    - » podle toho náročnost, rozsah, počet testů
    - » může být založena na testech modulů
    - » přidání nových funkcí/vlastností ⇒ nové testy těsnosti

Regresní testy

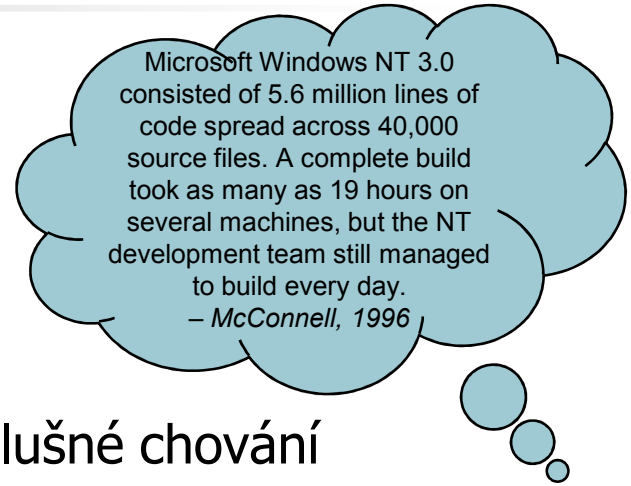
re·gress Pronunciation Key (rĭ-grĕs′)
v. re·gressed, re·gress·ing, re·gress·es
v. intr.

1. To go back; move backward.
2. To return to a previous, usually worse or less developed state.

– *dictionary.reference.com*

- Cíl: zajistit, aby nové funkce a vylepšení nesnižovaly kvalitu již hotového kódu
  - prevence stárnutí kódu: zanášení chyb při implementaci vylepšení
  - vymyslet mechanismus jak vytvářet vhodné testy
- Postup: ověřit potenciální baseline produktu pomocí testů, kterými musí projít a/nebo kterými v minulosti neprošel
  - indikátor existence systémových problémů
    - » určení zdroje chyby není nutné ⇒ testy integrační, modulů
  - testování změn v (nečekaných) integračních aspektech
  - spuštění při integračním sestavení, před velkými změnami
  - zdroj testů: chyby objevené QA, při validaci, zákazníkem
    - » produkt selhal → napsat test, který to dokáže → přidat jej do sady regresních testů (odebírání testů ze sady není dobrý trik)

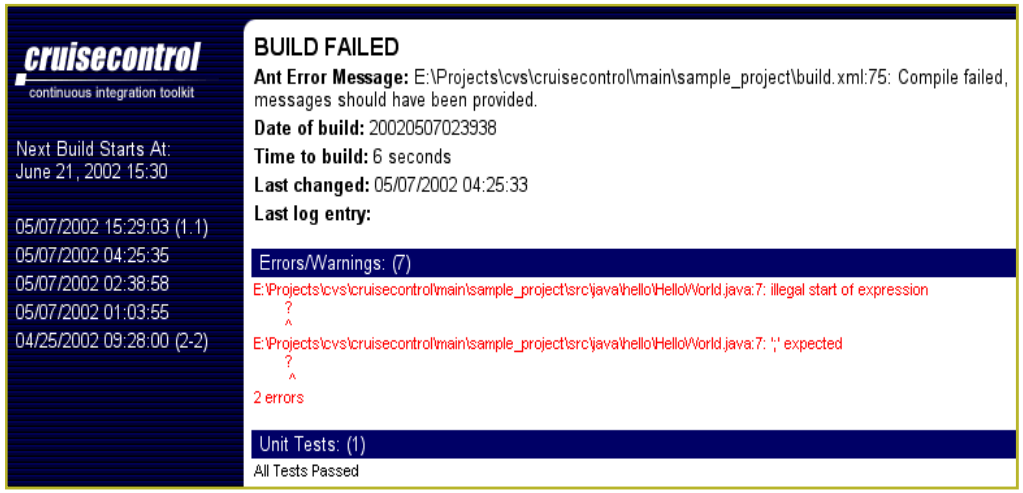
Daily Build and Smoke test



- Integrační sestavení + zkouška těsnosti
  - *pravidelně* 1x denně (někdy nočně)
  - výsledky okamžitě známy a reflektovány
    - » nová hlášení problémů
    - » opravy *ihned* zapracovány do kódu
  - check-in kódu, který vede k chybám, je neslušné chování
    - » lehká (nebo i vážnější) sankce vhodná
- Výhody
  - zvladatelné množství změn během denních check-in
    - ⇒ zvladatelné množství oprav, detekce problémů „vždyť včera to fungovalo“, analýza změn kódu místo ladění – viz diskuse o sestavení
  - pravidelný, obecně známý rytmus projektu
  - lepší morálka týmu („to nám to roste“)
- Cena: trocha discipliny, trocha automatizace

Continuous Integration

- Dotažení do dokonalosti (nebo extrému ;-)
  „1x denně“ → „neustále“
- Klíč: automatizace
  - co/ci, sestavení, testování, oznamování
  - robot na spouštění



Far from being a nuisance, the NT team attributed much of its success on that huge project to their daily builds. Those of us who work on projects of less staggering proportions will have a hard time explaining why we aren't also reaping the benefits of this practice.

– Steve McConnell, 1996

Nástroje pro podporu sestavení

- Scriptovací
  - shell, perl, python, php, …
  - make
  - ant, maven
  - CruiseControl
- Verifikace sestavení
  - xUnit (JUnit apod., Cactus)
  - testovací roboti

make

- Nástroj pro automatizaci *build* (překladu a sestavení) projektu
  - na základě popisu závislostí typu zdrojový-odvozený
- Pojmy:
  - pravidlo (*rule*)
  - cíl (*target*)
  - závislost (*dependency*)
  - příkaz (*command*)
- Makefile: definice pravidel
  - spouští se `make`
- Základní podoba makefile

```
CC = gcc    # pro DOS: tcc
OBJ = main.o soubory.o

all: program
program: $(OBJ)
        $(CC) -o $@ $(OBJ)
main.o: main.c main.h soubory.h
        $(CC) -c main.c
soubory.o: soubory.c soubory.h
        $(CC) -c soubory.c

clean:
        rm *.o core
```

make: Podrobnosti a konvence

- Způsob zpracování:
  - default cíl = první v pořadí
  - spouští se příkazy pravidel, v nichž jsou závislosti „mladší" než cíl
  - na akce každého pravidla se spouští sub-shell
    - ⇒ nepřenáší se proměnné
- Falešné (*phony*) cíle:
  - „návěští podprogramu" nikoli výsledný soubor
    - např. pro úklidové akce
  - nemají závislosti, pouze akce
  - `.PHONY all clean` (prevence, kdyby existoval soubor `clean`)
- Zpracování chyb:
  - implicitně ukončení make při libovolné chybě v příkazu akce
  - `-rm *.o core` – ignorování chyby v příkazu
- Konvence:
  - `Makefile` nebo `makefile`
  - standardně cíle all, clean, install, test
  - vhodné definovat proměnné SHELL, CC, CFLAGS, OBJ

ant

http://www.linuxzone.cz/index.phtml?ids=2&idc=390

- Rozšiřitelný nástroj pro automatizaci sestavení
  - Java
- Buildfile (build.xml)
- 3 hlavní typy elementů
  - Project - root element dokumentu.
  - Target - činnosti, které je potřeba vykonat pro dosažení cíle
    - » cílů je v buildfile většinou více a mohou na sobě záviset
  - Task - příkaz, který vykonává určitou činnost
    - » mnoho předdefinovaných tasků
    - » možnost napsat si vlastní => rozšiřitelnost

Příklad antfile

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="compile" name="aplikace">
  <target name="compile"> <!-- spouští kompilátor javy. -->
    <javac debug="true" deprecation="true" destdir="." srcdir="."> </javac>
  </target>

  <target name="run" depends="compile"> <!-- spouští zkompilovaný program. -->
    <java classname="com.patrny.ant.Aplikace" failonerror="true" fork="true">
      <classpath> <pathelement location="."/> </classpath>
      <!-- argumenty programu -->
      <arg value="test 1 2 3"/>
    </java>
  </target>

  <target name="clean"> <!-- Smaže soubory. -->
    <delete> <!-- Výběr souborů -->
      <fileset dir="."> <include name="**/*.class"/> </fileset>
    </delete>
    <delete file="aplikace.jar"/>
    <delete dir="apidoc"/>
  </target>
</project>
```

Maven

- Správa projektů
  - „project object model“
  - komponenty, závislosti
- Velmi snadný build
  - pro většinu jazyků a typů projektů předdefinované tasky
  - možnost pluginů
  - centrální úložiště + lokální cache pro knihovny