Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava 4

# Manažment verzií zdrojového kódu pomocou Git

## Metodika (Téma č. 3)

**Bc. Róbert Móro**

Študijný program: Softvérové inžinierstvo
Ročník: 1.
Predmet: Manažment softvérových a informačných systémov
Ak. rok: 2010/11

# 1 Úvod

Účelom tejto metodiky je opis procesov manažmentu verzií zdrojového kódu v prostredí malého vývojového tímu (5-7 ľudí). Použitým nástrojom pre manažment verzií je Git[1]. Metodika je určená pre všetkých členov vývojového tímu, ich roly a prislúchajúce zodpovednosti sú uvedené v časti 3 dokumentu. Zoznam nadväzujúcich metodík a dokumentov je uvedený v časti 2.

Metodika zachytáva manažment verzií zdrojového kódu na dvoch úrovniach – vyššia úroveň je zachytená v častiach 2 až 4, nižšia úroveň, opisujúca kroky vybraných procesov v nástroji Git, je zachytená v častiach 5 a 6.

## 1.1 Pojmy a skratky

**Git**
Voľne dostupný distribuovaný systém kontroly verzií.

**Repozitár**
Úložisko dokumentov (zdrojových kódov), ktoré je spravované pomocou Git-u, t.j. adresár so svojimi podadresármi. Rozlišuje sa centrálne (vzdialené) a lokálne úložiská.

**Vetva**
Predstavuje odklonenie vo vývoji od hlavnej, tzv. *master* vetvy. Hlavná vetva spravidla obsahuje len otestovaný, funkčný a stabilný kód a každá nová funkcionalita sa vyvíja a testuje v samostatnej vetve.

**Commit**
Odoslanie zmien do lokálneho úložiska.

**Merge**
Zlúčenie rôznych vývojových vetiev, zahŕňa riešenie konfliktov medzi rôznymi verziami toho istého súboru.

**Rebase**
Proces získania zmien, ktoré boli vykonané na inej vývojovej vetve, za účelom aktualizácie danej vývojovej vetvy.

**Konflikt**
Vzniká, ak bol rovnaký súbor modifikovaný dvoma rôznymi spôsobmi.

# 2 Nadväzujúce metodiky a dokumenty

Manažment verzií v kontexte etáp životného cyklu softvéru súvisí s:

- manažmentom zmien a požiadaviek na zmenu aplikácie
- manažmentom chýb
- manažmentom konfigurácií softvérového systému

[1] http://git-scm.com/
http://book.git-scm.com/

Manažment zmien opisuje procesy a určuje formálne postupy zaevidovania požiadavky na zmenu. Výstupom je analýza a návrh, na základe ktorého sa zmena implementuje. Pri samotnej implementácii sa následne využívajú procesy manažmentu verzií zdrojového kódu definované v tejto metodike.

Manažment chýb opisuje životný cyklus chyby od jej nahlásenia, overenia, cez pridelenie až po jej vyriešenie. Manažment verzií vstupuje do tohto procesu v momente začatia riešenia chyby vývojárom, vstupom je protokol o zaevidovanej a overenej chybe.

Manažment konfigurácií softvérového systému riadi proces nasadzovania a zostavovania softvérového systému. Vstupom je produkčná verzia zdrojových kódov systému. Systém pre kontrolu verzií zabezpečuje správu viacerých verzií (konfigurácií) systému naraz.

Manažment verzií tiež podporuje manažment úloh v distribuovanom projekte, napomáha pri kontrole práce a vyťaženia jednotlivých členov tímu.

Spomínaným súvisiacim procesom sa venujú príslušné metodiky a samostatné dokumenty.

# 3 Roly a zodpovednosti účastníkov

V tab. 1 sú uvedené identifikované roly, ktoré vystupujú v procesoch manažmentu verzií spolu s ich definovanými zodpovednosťami.

**Tab. 1** Roly a zodpovednosti.

| Rola | Zodpovednosť |
|---|---|
| Vývojár | • Vývoj nových funkcionalít<br>• Oprava chýb |
| Manažér vývoja | • Riešenie konfliktov |
| Manažér podpory vývoja | • Inštalácia<br>• Konfigurácia úložiska<br>• Správa úložiska |
| Manažér kvality | • Testovanie<br>• Hodnotenie kvality<br>• Rozhodnutie o nasadení v produkčnej verzii |

# 4 Procesy manažmentu verzií zdrojového kódu

Manažment verzií zdrojového kódu zahŕňa tieto procesy:

1. Proces inštalácie (kap. 4.1)
2. Proces konfigurácie (kap. 4.2)
3. Proces prepojenia s nástrojom na manažment úloh (kap. 4.3)
4. Proces vývoja novej funkcionality (kap. 4.4)
5. Proces opravy chyby (kap. 4.5)
6. Proces riešenia konfliktov (kap. 4.6)
7. Proces nasadzovania (kap. 4.7)

## 4.1 Proces inštalácie

| | |
|---|---|
| *Vstup:* | Nakonfigurované vývojové prostredie na serveri |
| *Výstup:* | • Nainštalovaný nástroj na manažment verzií<br>• Zápis z inštalácie |
| *Zodpovednosť:* | Manažér podpory vývoja, vývojár |
| *Súvisiace dokumenty:* | Postup inštalácie a konfigurácie vývojového prostredia |

Primárnu zodpovednosť za proces inštalácie má manažér podpory vývoja, ktorý inštaluje nástroj na manažment verzií na serveri. Vývojári zodpovedajú za inštaláciu nástroja na ich lokálnom počítači, splnenie úlohy kontroluje opäť manažér podpory vývoja.

Proces teda pozostáva z týchto krokov:

1. Inštalácia nástroja na serveri
2. Inštalácia nástroja na lokálnom počítači
3. Kontrola a zdokumentovanie inštalácie

## 4.2 Proces konfigurácie

| | |
|---|---|
| *Vstup:* | Nainštalovaný nástroj na manažment verzií |
| *Výstup:* | Nakonfigurované centrálne úložisko na serveri |
| *Zodpovednosť:* | Manažér podpory vývoja, vývojár |
| *Súvisiace dokumenty:* | Žiadne |

Primárnu zodpovednosť za proces konfigurácie má manažér podpory vývoja, ktorý konfiguruje centrálne úložisko a prístup k nemu. Vývojár konfiguruje lokálne úložisko.

Proces konfigurácie pozostáva z týchto krokov:

1. Vytvorenie centrálneho úložiska
2. Vytvorenie prípravnej (*staging*) vetvy
3. Nastavenie prístupových práv používateľov k centrálnemu úložisku
4. Konfigurácia lokálneho úložiska

## 4.3 Proces prepojenia s nástrojom na manažment úloh

| | |
|---|---|
| *Vstup:* | • Nainštalovaný a nakonfigurovaný nástroj na manažment verzií<br>• Nainštalovaný a nakonfigurovaný nástroj na manažment úloh |
| *Výstup:* | Prístup k centrálnemu úložisku pomocou nástroja na manažment úloh |
| *Zodpovednosť:* | Manažér podpory vývoja |
| *Súvisiace dokumenty:* | Metodika manažmentu úloh |

Proces prepojenia s nástrojom na manažment úloh zabezpečuje manažér podpory vývoja. Prepojenie umožní:

- Prístup k centrálnemu úložisku pomocou nástroja na manažment úloh

- Sledovanie zmien (commitov) aj s komentármi od vývojárov a následné vyhodnotenie ich činnosti
- Prepojenie zmien s úlohami pridelenými vývojárom v nástroji na manažment úloh (automatické uzatvorenie úlohy pri jej vyriešení)

## 4.4 Proces vývoja novej funkcionality

| | |
|---|---|
| *Vstup:* | Analýza a návrh novej funkcionality |
| *Výstup:* | • Implementovaná a otestovaná nová funkcionalita nasadená v produkčnej verzii systému<br>• Popis implementácie<br>• Zápis z testovania |
| *Zodpovednosť:* | Vývojár, manažér vývoja |
| *Súvisiace dokumenty:* | • Metodika testovania<br>• Metodika technickej dokumentácie |

Primárnu zodpovednosť za vývoj novej funkcionality má vývojár, manažér vývoja len koordinuje vývoj a rieši problémy vývojárov.

Proces vývoja novej funkcionality má za cieľ implementovať novú funkcionalitu podľa požiadaviek zákazníka zachytených v analýze. Ide o proces v širšom kontexte vývoja, testovania a nasadzovania, z pohľadu manažmentu verzií pozostáva z týchto krokov:

1. Vytvorenie novej vetvy na vývoj funkcionality (kap. 5.1)
2. Vývoj novej funkcionality v samostatnej vetve (kap. 5.2)
3. Zlúčenie vetvy funkcionality s prípravnou vetvou (kap. 5.3)

Podrobný opis krokov je uvedený v časti 5.

## 4.5 Proces opravy chyby

| | |
|---|---|
| *Vstup:* | Nahlásenie, overenie a pridelenie chyby na riešenie |
| *Výstup:* | • Opravená chyba<br>• Zápis o riešení |
| *Zodpovednosť:* | Vývojár, manažér vývoja |
| *Súvisiace dokumenty:* | Metodika manažmentu chýb |

Primárnu zodpovednosť za opravu chyby má vývojár, manažér vývoja len koordinuje vývoj a rieši problémy vývojárov.

Proces opravy chyby má za cieľ opraviť chybu nájdenú v procese testovania alebo pri prevádzke systému v prípravnej (*staging*) alebo produkčnej verzii. Ide o proces v širšom kontexte vývoja, testovania a manažmentu chýb, z pohľadu manažmentu verzií pozostáva z týchto krokov:

1. Vytvorenie novej vetvy na opravu chyby
2. Oprava chyby v samostatnej vetve
3. Zlúčenie vetvy opravy chyby s prípravnou vetvou

Kroky procesu sú podobné krokom procesu vývoja novej funkcionality, ktoré sa opisujú v časti 5, preto sú v podrobnom opise týchto krokov v časti 6 uvádzané len rozdiely oproti nim.

## 4.6 Proces riešenia konfliktov

| | |
|---|---|
| *Vstup:* | Konflikt medzi dvomi rôznymi zmenami toho istého zdrojového súboru |
| *Výstup:* | Vyriešený konflikt – jeden zmenený zdrojový súbor |
| *Zodpovednosť:* | Manažér vývoja, vývojár |
| *Súvisiace dokumenty:* | Žiadne |

Konflikty môžu vznikať na dvoch úrovniach:

- Pri aktualizácii vetvy vývoja z hlavnej vetvy centrálneho úložiska vývojárom (*rebase*, viď kap. 5.2)
- Pri zlučovaní vetiev

Konflikty na prvej úrovni rieši vývojár, ktorý zodpovedá za to, že vyvíja v aktuálnej verzii systému, čím sa minimalizujú konflikty. Konflikty na druhej úrovni môžu nastať v prípade neodborného alebo nepozorného zásahu vývojára, presnejšie v prípade nevhodne vyriešeného konfliktu prvej úrovne, ktorý má za následok následnú nefunkčnosť niektorej časti systému vyvíjanej iným vývojárom. Tieto druhy konfliktov rieši manažér vývoja.

Riešenie konfliktov prvej úrovne je opísané v kap. 5.2. Riešenie konfliktov druhej úrovne súvisí s manažmentom prehliadok kódu a testovania, v kontexte manažmentu verzií pozostáva z týchto krokov:

1. Sledovanie histórie zmien
2. Porovnávanie obsahu zmien
3. Vyriešenie konfliktu

Posledný krok zahŕňa návrat k predchádzajúcej verzii alebo nahlásenie chyby.

## 4.7 Proces nasadzovania

| | |
|---|---|
| *Vstup:* | • Vývojová verzia systému<br>• Akceptačné testy |
| *Výstup:* | • Produkčná verzia systému<br>• Zápis z akceptačného testovania |
| *Zodpovednosť:* | Manažér vývoja, manažér kvality |
| *Súvisiace dokumenty:* | Metodika testovania |

Proces nasadzovania súvisí s manažmentom kvality a testovania a predovšetkým s manažmentom nasadzovania, preto pre podrobný opis viď príslušné metodiky. Z pohľadu manažmentu verzií je dôležité rozlíšiť medzi rôznymi verziami systému. Preto sa rozlišujú tieto vetvy:

- Hlavná (*master*)
- Prípravná (*staging*)

*Prípravná verzia* systému slúži na odskúšanie a integráciu otestovaných funkcionalít pred ich nasadením do produkčnej verzie, nasadzuje sa z prípravnej (*staging*) vetvy. *Produkčná verzia* systému je prezentovaná zákazníkovi, preto zahŕňa len odladené (otestované) funkcionality, ktoré prešli integračnými a akceptačnými testami. Táto verzia sa aktualizuje po ukončení každého vývojového cyklu, nasadzuje sa z hlavnej (*master*) vetvy. O nasadení do prípravnej verzie rozhoduje manažér vývoja, do produkčnej verzie manažér kvality.

Okrem toho sa podľa potreby vytvárajú ďalšie vetvy systému, zachytávajúce ho v istom stave.

# 5 Detailný opis krokov procesu vývoja novej funkcionality

## 5.1 Vytvorenie novej vetvy na vývoj funkcionality

Každá funkcionalita sa vyvíja v samostatnej vetve. Vývojár si pred vytvorením novej vetvy aktualizuje hlavnú a prípravnú vetvu na svojom lokálnom repozitári pomocou príkazov:

```
$ git checkout master
$ git pull origin master
$ git checkout staging
$ git pull origin staging
```

Následne vývojár vytvorí novú vetvu. Názov vetvy sa skladá z troch častí – identifikátora typu úlohy (*feature* alebo *bug*) , identifikátora úlohy v nástroji na manažment úloh a stručného popisu. Napr. ak ide vývojár vyvíjať funkcionalitu k úlohe *137 Vytvorenie projektu*, názov vetvy bude vo formáte *feature-137-novy-projekt*.

```
$ git checkout –b <názov vetvy>
```

## 5.2 Vývoj novej funkcionality v samostatnej vetve

Vývojár vyvíja novú funkcionalitu vo vetve vytvorenej v predchádzajúcom kroku. Prehľad vykonaných zmien (a zmien, ktoré sa budú odosielať pri zadaní príkazu *commit*) sa získa príkazom:

```
$ git status
```

Pravidelne sa pridávajú a odosielajú zmeny do lokálneho úložiska:

```
$ git add –i
$ git commit –m "[#<id-úlohy>] <Vysvetľujúca správa>"
```

Prvý príkaz pridáva súbory interaktívne, teda je možné pridať len vybrané súbory. Keď to nie je potrebné, nasledujúcimi príkazmi sa pridajú a odošlú všetky zmeny:

```
$ git add .
$ git commit –am "[#<id-úlohy>] <Vysvetľujúca správa>"
```

Správa každého *commit* príkazu musí byť dostatočne vysvetľujúca, aby bolo zrejmé, k akej zmene došlo, píše sa po slovensky bez diakritiky v trpnom tvare, napr. *"[#137] Pridane vytvaranie noveho projektu"*.

Vetva, v ktorej sa vyvíja nová funkcionalita, sa pravidelne aktualizuje obsahom z prípravnej vetvy, aby sa vyvíjalo nad aktuálnou verziou systému:

```
$ git checkout staging
$ git pull origin staging
$ git checkout <názov-vetvy>
$ git rebase staging
```

Pri vykonávaní príkazu *rebase* môžu nastať konflikty, v takom prípade sa vykonávanie zastaví a treba vyriešiť konflikt. Po jeho vyriešení sa pridajú zmeny a pokračuje sa v ďalej v *rebase*:

```
$ git add –i
$ git rebase --continue
```

Vetva vývoja novej funkcionality sa odosiela do centrálneho úložiska:

```
$ git push origin <názov-vetvy>
```

## 5.3 Zlúčenie vetvy funkcionality s prípravnou vetvou

Po ukončení vývoja novej funkcionality a jej otestovaní sa vykoná interaktívny *rebase*, aby sa kvôli prehľadnosti zredukoval počet zaznamenaných *commit* príkazov, ktoré sa odošlú do centrálneho úložiska.

```
$ git rebase –i origin/staging
```

Napokon sa po schválení funkcionality manažérom vývoja vetva zlúči s prípravnou (*staging*) vetvou v lokálnom úložisku a odošle sa do príslušnej vetvy centrálneho úložiska:

```
$ git checkout staging
$ git merge <názov-vetvy>
$ git push origin staging
```

# 6 Detailný opis krokov procesu opravy chyby

Všetky kroky sú rovnaké ako kroky v časti 5, až na tieto rozdiely:

- ako prefix názvu vetvy sa použije *bug*, a nie *feature*
- pri interaktívnom *rebase* sa zlúčia všetky *commit* príkazy do jedného a v komentári sa za identifikátorom úlohy pred vysvetľujúcou správou uvedie *"FIX:"*