Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4

Askalot meets Harvard Courses at edX

[Askalot2edX]

Metodika pre podporu vývoja a integrácie

Tím: číslo 6, AskEd
Vedúci tímu: Ing. Ivan Srba
Členovia tímu: Černák Martin, Gallay Ladislav, Hnilicová Eva, Huňa Adrián, Jandura Filip, Žuffa Tibor
Akademický rok: 2015/2016
Autor: Ladislav Gallay
Verzia číslo: 5
Dátum poslednej zmeny: 12.12.2015

Úvod

Cieľom tejto metodiky je definovanie základných pojmov a postupov pri manažovaní zdrojového kódu a verzií projektu. Metodika presne definuje prácu s nástrojmi na správu revízií zdrojového kódu, konvencie pre odovzdávanie a vetvenie zdrojového kódu a riešenie konfliktov. Metodika opisuje prácu s nástrojom Git a službou GitHub.

Metodika je určená, pre každého člena tímu, ktorý potrebuje pristupovať k zdrojovému kódu a pracovať s ním, najmä však pre vývojárov.

Metodika vychádza z klasických prístupov k manažmentu zdrojového kódu rovnako ako aj z prístupov používaných pri populárnych projektoch s otvoreným zdrojovým kódom. Jednotlivé postupy sú uvedené v chronologickom poradí v akom ich programátor pri vývoji spravidla použije, t.j. od inicializácie repozitára, cez odovzdávanie zdrojového kódu a pridávanie funkcionality, až po opravu chýb v softvéri.

1. Použité skratky a značky

RCS	Systém na správu revízií (angl. revision control system) zdrojového kódu a iných dokumentov a súborov projektu
ITS	Systém na sledovanie úloh (angl. issue tracking system) v rámci projektu
Git	Distribuovaný systém na správu revízií zdrojového kódu
GitHub	Služba a sociálna sieť na zdieľanie a vývoj zdrojového kódu využívajúca Git
Repozitár	Repozitár (angl. repository). Úložisko zdrojového kódu a iných dokumentov a súborov projektu v rámci systému na správu revízií. Rozlišujeme vzdialené (na serveri) a lokálne repozitáre
Odovzdanie	Odovzdanie (angl. commit) je množina zmien v súboroch projektu v určitom čase podpísaná autorom.
Vetva	Vetva (angl. branch) je postupnosť odovzdaní. Vetvenie umožňuje spravovať súbory v jednej vetvy repozitáru nezávisle od iných vetiev
Feature	Funkcionalita, ktorá je podporená používateľským príbehom

2. Postupy

2.1. Nastavenie nástroja Git a prepojenia so službou GitHub

Predpokladá sa nainštalovaný nástroj Git a vytvorený účet v sieti Github. Používateľské meno v sieti GitHub sa odporúča zadať v tvare „[iniciála-mena][priezvisko]“ alebo „[meno][priezvisko]“ (s malými písmenami), napr. „lgallay“ alebo „ladislavgallay“. Pokiaľ už používateľ má existujúci účet, nie je potrebné meniť jeho názov alebo vytvárať nový.

Postup:

1. Nastaviť používateľské meno a e-mail v nástroji Git. Používateľské meno musí byť v tvare „[Meno] [Priezvisko]“ (s diakritikou), napr. „Ladislav Gallay“ alebo „Tibor Žuffa“. Používateľké meno sa dá nastaviť iba pre konkrétny repozitár bez použitia prepínaša --global.

   ```
   git config --global user.name "username"
   git config --global user.email "user@example.com"
   ```

2. Nastaviť rovnaký primárny e-mail účtu v sieti GitHub ako bol nastavený v nástroji Git.
3. Vygenerovať SReEESH kľúč a priradiť ho k účtu v sieti GitHub.

   ```
   ssh-keygen -t rsa -C "user@example.com"
   ```

   Otvoriť vygenerovaný súbor id_rsa.pub a skopírovať obsah medzi kľúče SSH účtu v sieti GitHub a potvrdiť pridanie kľúča.
4. Pridať fotku k účtu v sieti GitHub pomocou služby Gravatar.
5. Požiadať správcu organizácie v sieti GitHub o členstvo (platí iba v prípade ak má tím vytvorenú organizáciu).
6. Informovať správcu vzdialených repozitárov o úspešnom nastavení a požiadať o pridelenie práv na odovzdávanie zdrojového kódu.

2.2. Inicializácia vzdialeného repozitára

Najskôr sa vytvorí nový lokálny repozitár a následne sa vykoná prvé odovzdanie do vzdialeného repozitára v rámci služby GitHub. Názov repozitára musí byť malými písmenami a ako oddeľovač sa smie použiť iba krátka pomlčka, napr. „project-site“.

Postup:

1. Vytvoriť vzdialený repozitár projektu službou GitHub. Pri tvorbe repozitáru cez rozhranie služby nepridať automaticky žiadne súbory ani nevykonať žiadne odovzdanie. Výstupom tohto procesu sú aktívne HTTPS a SSH adresy vzdialeného repozitáru, napr.:

   ```
   https://github.com/username/project.git
   git@github.com:username/project.git
   ```

2. Vytvoriť lokálny repozitár projektu v novom adresári:

   ```
   mkdir project
   cd project
   git init
   ```

3. Vytvoriť súbor .gitignore. Obsah súboru musí aspoň základne pokrývať potreby projektu vzhľadom na použitý programovací jazyk, knižnice alebo aplikačné rámce.
4. Vytvoriť súbor `README.md`, so základnými informáciami o projekte.

5. Vytvoriť súbor `LICENSE.md`, ktorý obsahuje text MIT licencie.
6. Vykonať prvé odovzdanie so správou „Initial commit“:
```
git add .
git commit -m "Initial commit"
```
7. Pridať referenciu na vzdialený repozitár projektu.
```
git remote add origin git@github.com:username/project.git
```
8. Odovzdať vykonané zmeny do vzdialeného repozitára projektu.
```
git push origin branch-feature
```
kde *branch-feature* je názov vetvy.
9. V sieti GitHub prideliť práva členom tímu na odovzdávanie zdrojového kódu a podľa potreby informovať členov tímu o vzniku repozitára.

2.3. Inicializácia lokálneho repozitára

Inicializácia lokálneho repozitára na základe existujúceho vzdialeného repozitára sa vykonáva príkazom:

```
git clone git@github.com:username/project.git
```

2.4. Pravidlá pre odovzdávanie zdrojového kódu

Je žiadúce aby odovzdania mali jemnú granularitu zmien a zároveň aby zmeny v odovzdaní medzi sebou logicky súviseli.

Pre správu odovzdania (angl. commit message) platí:

- píše sa v angličtine,
- prvé písmeno je vždy veľké,
- na konci správy nie je bodka,
- správa začína slovesom v prítomnom jednoduchom čase, ktoré opisuje typ danej zmeny, napríklad `Add`, `Edit`, `Fix`, `Change`, `Rename`, `Remove`, `Update`, `Refactor`, `Implement`, `Resolve`, `Close`.
- na delenie správy na menšie časti sa používa čiarka (správa nesmie obsahovať žiadnu inú interpunkciu),
- má maximálne 50 znakov,
- skratky sa píšu korektne (nepísať „xml“, ani „Xml“ ale správne „XML“).

V kontexte jazyka Ruby pre správu odovzdania ďalej platí:

- názvy modulov a tried sa píšu iba v tvare „XmlParser“ alebo „XML Parser“,
- neuvádza sa menný priestor modulov alebo tried (nepísať „Core::XmlParser“).

V kontexte ITS pre správu odovzdania ďalej platí:

- má tvar „[akcia-nad-stavom-úlohy] #[identifikátor-úlohy] [zvyšok-správy]“,
- akcia nad stavom úlohy v ITS a je buď Fix, Implement, Resolve alebo Close,
- referencia na úlohu v ITS sa uvádza v tvare „#[identifikátor-úlohy]“.

Príklady správ odovzdania:

```
Disable user registration via email
Refactor user model, fix user helpers
Add XmlParser, XmlNode and XmlHelpers
Update XML library, add XML Parser
```

```
Fix #31 correct URL to documents
```

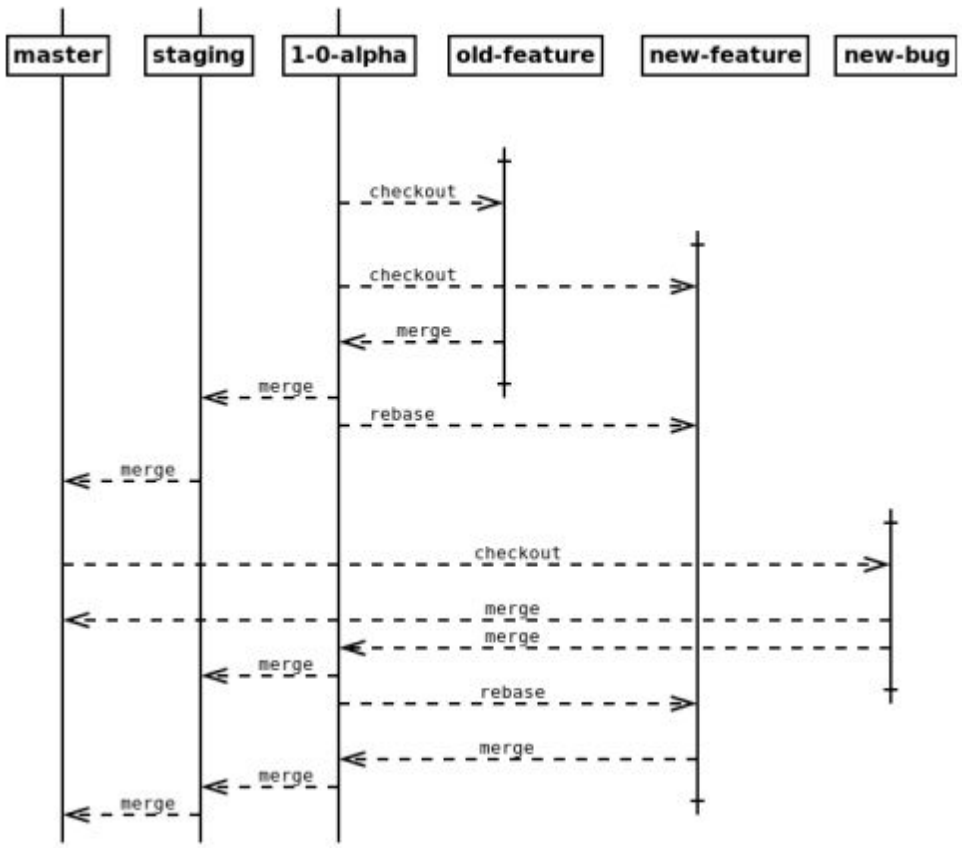
2.5. Pravidlá pre vetvenie zdrojového kódu

Pre názov vetvy platí:

- píše sa v angličtine,
- obsahuje iba malé písmená,
- používa krátku pomlčku ako oddeľovač slov (neobsahuje žiadnu inú interpunkciu),
- je stručný a presný.

V rámci hlavného repozitára projektu, v ktorom je aplikácia dodávaná zákazníkovi. Rozlišujeme tieto vetvy:

- Produkčná vetva (angl. production branch), je distribuovaná zákazníkovi a obsahuje funkcionalitu schválenú zákazníkom. Názov vetvy je „master“.
- Testovacia vetva (angl. staging branch), obsahuje novú a otestovanú funkcionalitu pripravenú na schválenie zákazníkom. Vetva sa spája s produkčnou vetvou a do nej sa spája aktuálna vývojová vetva. Názov vetvy je „staging“.
- Vetvy pre novú funkcionalitu (angl. feature branch), v každej sa implementuje práve jedna nová funkcionalita podporená používateľským príbehom. Názvy vetiev majú sufix „-feature“, napr. „user-authorization-feature“.
- Vetvy pre opravy chýb (angl. bug fix branch), v každej sa implementuje oprava práve jednej chyby nahlásenej v systéme ITS. Názvy vetiev majú sufix „-bug“, napr. „login-bypass-bug“.
- Ostatné vetvy sú spravidla tie, v ktorých prebieha migrácia na nové verzie použitých technológií alebo refaktorovanie zdrojového kódu.



Obr. 1: *Príklad vývoju vetiev repozitáru v čase.*

Na diagrame vyobrazenom na Obr. 1. je načrtnutý vývoj vetiev repozitáru s projektom v čase. Z nadradenej vetvy sa do vlastnej vetvy používa príkaz `rebase` a do nadradenej príkaz `merge`.

2.6. Pravidlá pre riešenie konfliktov

Ak nastane konflikt v paralelne editovaných súboroch pri spájaní vetiev, tak platí, že:

- pred nasledujúcim odovzdaním dotknutých súborov musia byť všetky konflikty vyriešené,
- sa riešením konfliktov nesmie narušiť existujúca funkcionalita softvéru.

2.7. Pridanie novej funkcionality podporenej používateľským príbehom

Pridanie novej funkcionality predpokladá existenciu zodpovedajúcej úlohy v ITS.

Postup:

1. Vytvoriť novú lokálnu vetvu s funkcionalitou.
   ```
   git checkout -b new-feature
   ```
2. Ak už bola vetva s funkcionalitou nahraná do vzdialeného repozitára je potrebné vykonať aktualizovanie lokálneho zdrojového kódu:
   ```
   git pull --rebase origin new-feature
   ```
3. Ak pribudli odovzdania do príslušnej vývojovej vetvy je potrebné aktualizovať vetvu s funkcionalitou o tieto odovzdania:
   ```
   git rebase gama
   ```
4. Implementovať novú funkcionalitu, po častiach lokálne odovzdávať zdrojový kód a podľa potreby sa vracať ku krokom číslo 2 a 3.
5. Lokálne skontrolovať odovzdaný zdrojový kód, skontrolovať a spustiť testy.
6. Nahrať zdrojový kód vetvy s funkcionalitou do vzdialeného repozitára.
   ```
   git push origin new-feature
   ```
7. Požiadať poverenú osobu o prehliadku zdrojového kódu.
   - Ak poverená osoba implementáciu akceptuje, tak sa pokračuje krokom číslo 8,
   - inak poverená osoba určí čo treba opraviť a pokračuje sa krokom číslo 2.
8. Spojiť vetvu s príslušnou vývojovou vetvou.
   ```
   git checkout gama
   git merge new-feature
   ```
9. Vyriešiť konflikty, ak nastali.
10. Nahrať zdrojový kód príslušnej vývojovej vetvy do vzdialeného repozitára.
    ```
    git push origin gama
    ```
11. Požiadať poverenú osobu o spojenie vetvy s testovacou vetvou.

2.8. Riešenie kritických úloh pre produkčnú vetvu

Riešenie rýchlej opravy funkcionality alebo iného vážneho problému v produkčnej vetve smie vykonávať iba poverená osoba pričom sa predpokladá existencia zodpovedajúcej úlohy v ITS.

Postup:

1. Vytvoriť novú lokálnu vetvu s opravou z produkčnej vetvy.
   ```
   git checkout master
   git pull origin master
   git branch new-bug
   ```
2. Implementovať opravu a po častiach lokálne odovzdávať zdrojový kód.

3. Lokálne skontrolovať odovzdaný zdrojový kód, skontrolovať a spustiť testy.
4. Nahrať zdrojový kód vetvy s opravou do vzdialeného repozitára. git push origin new-bug
5. Spojiť vetvu s produkčnou vetvou a príslušnou vývojovou vetvou.

```
git checkout master
git merge new-bug

git checkout gama
git merge new-bug
```

6. Vyriešiť konflikty, ak nastali (predpokladajú sa konflikty s príslušnou vývojovou vetvou).
7. Nahrať zdrojový kód produkčnej vetvy a príslušnej vývojovej vetvy do vzdialeného repozitára.

```
git push origin master
git push origin gama
```

8. Požiadať poverenú osobu o nasadenie produkčnej vetvy do prevádzky

2.9. Prehliadka zdrojového kódu

Kontrolu kvality kódu je nutné vykonať pri každom odovzdaní kódu vývojara zo svojej súkromnej vetvy do jednej zo spoločných vetiev, napríklad po každej ukončenej úlohe, ktorá v plnej miere plní svoju funkcionalitu. Pri prehliadaní zdrojového kódu používame funkciu porovnania vetiev v sytéme GitHub. Programátor nahrá zdrojový kód do vzdialeného repozitára v systéme GitHub a v ITS zmení zodpovednú osobu na člena tímu, ktorý má vykonať prehliadku zdrojového kódu.

V systéme GitHub sa vytvorí *pull request* zo svojej vetvy do požadovanej vetvy (*master, staging, gama,* prípadne iná nadradená vetva). Do komentára sa uvedie stručný zoznam zmien a ak je to nutné aj ich odôvodnenie. V opise sa musí nachádzať odkaz na úlohu v ITS, pokiaľ takáto úloha existuje. Spravidla sa do hlavnej vetvy *master* nikdy neposúvajú zmeny priamo z programátorskej vetvy. Kód by mal ísť najprv do jednej z testovacích vetiev a až odtiaľ môže byť postúpený ďalej do hlavnej vývojovej vetvy. To neplatí pre urgentné zmeny, ako napríklad opravy v zabezpečení systému a opravy iných dôležitých chýb.

Osoba vykonávajúca prehliadku zistené nedostatky v zdrojovom kóde komunikuje autorovi podľa Metodiky pre komunikáciu. Po dokončení prehliadky zdrojového kódu sa v ITS zaznačia prípadné zmeny vo funkcionalite. V prípade závažného nedostatku, ktorý by sa mohol objaviť aj u ďalších tvorcov zdrojového kódu, sú členovia tímu okamžite upozornení podľa platných princípov komunikácie v tíme.