Dr. Mileff Péter

SZOFTVERFEJLESZTÉS

VERZIÓKÖVETÉS, VERZIÓKÖVETŐ RENDSZEREK

Miskolci Egyetem
Általános Informatikai Tanszék

Miről is lesz szó?

- Verziókezelés fogalmának tisztázása
- Miért van rá szükség? Kik használják? Hol és Hogyan?
- Verziókezelés megvalósításának modelljei
- Alapfogalmak a verziókövetésben
- Műveletek
- Főbb verziókövető rendszerek ismertetése
- Szoftvereszközök bemutatása
- Gyakorlati bemutató

A VERZIÓKÖVETÉSRŐL ÁLTALÁNOSAN…

Verziókövetés?

Fogalma:

- olyan eljárások összessége, amelyek lehetővé teszik egy adathalmaz változatainak (verzióinak) együttes kezelését.
- **Szoftverek esetében:**
  - A szoftver életciklusa során a forráskódban végzett módosítások tárolása
    - Leggyakrabban forráskód fájlok verziónak támogatása
  - Menedzselés. Pl. loggolás, history, verzióváltás, visszagörgetés, ki mikor mit csinált, stb
- **Elnevezések:**
  - Revision Control, Version Control, Source Control, Source Code Management (SCM)

Miért szükséges?

- A fejlesztés során mindig szükség van a historikus adatokra!
  - A forráskód sok iteráción megy keresztül
    - Jó tudni mikor mi történt
  - Probléma esetén vissza lehet térni a korábbi verziókra
  - Független a fejlesztők számától
- **Egyszemélyes fejlesztés:**
  - Nincs párhuzamos fejlesztés (egy projekten belül)
    - önmagunknak is célszerű az egyes verziókat menedzselni

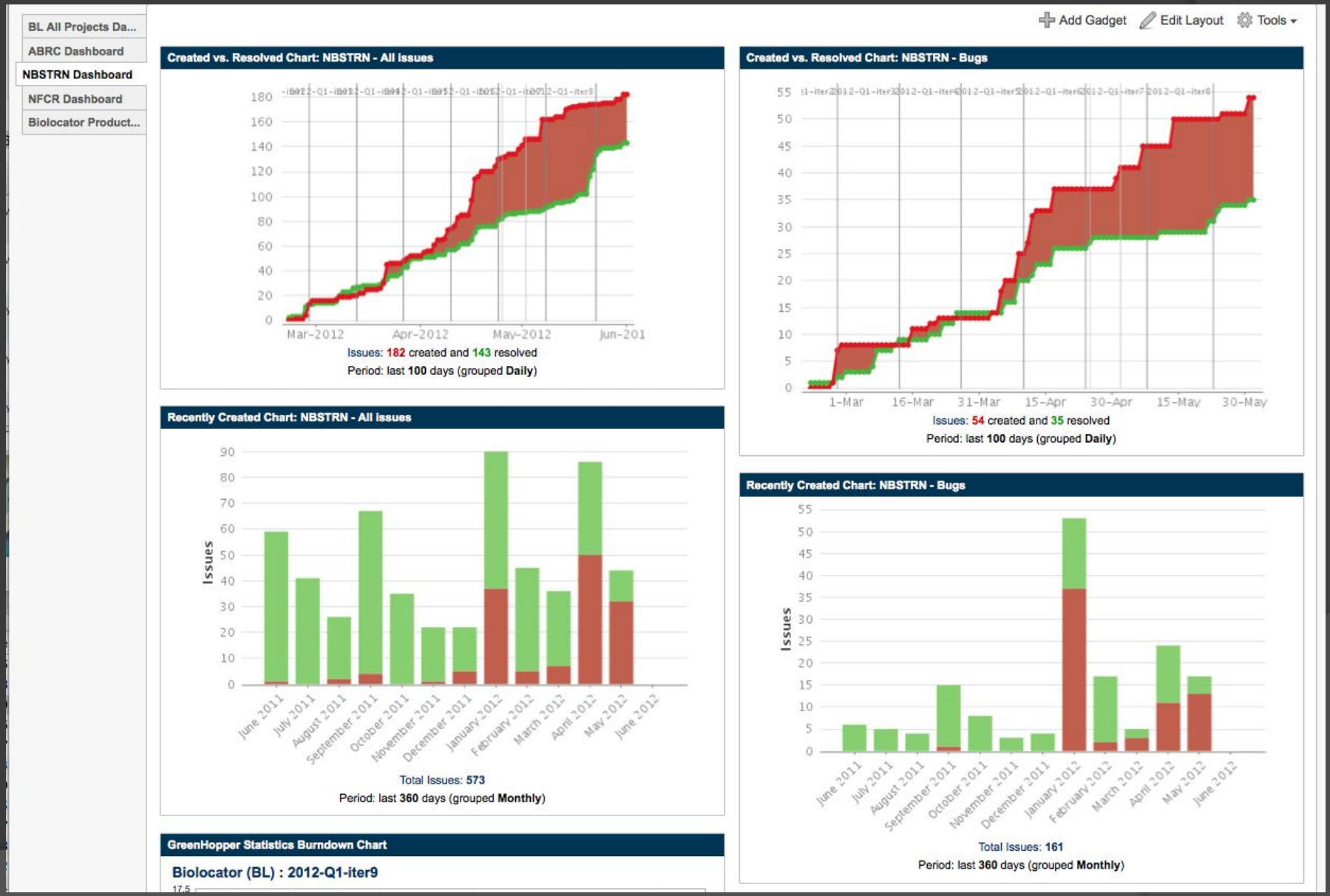
Miért szükséges?

- **Ma egy komolyabb szoftver fejlesztése több személyt igényel**
  - Feladatok tipikusan **csoport munkában** készülnek el
    - Folyamatos kommunikáció szükséges
    - A feladatok kiosztása párhuzamos
      - A csapat minden tagja dolgozik valamilyen feladaton
- Ezt a bonyolult kapcsolatot menedzselni kell
  - Látni kell, hogy ki, mikor, mit fejlesztett
  - Menedzselni kell a kód összefésülését azonos fájlokon való dolgozás esetén
  - **Egyéb:** speciális verziók jelölése, változatot összeolvasztása, stb

Miért szükséges?

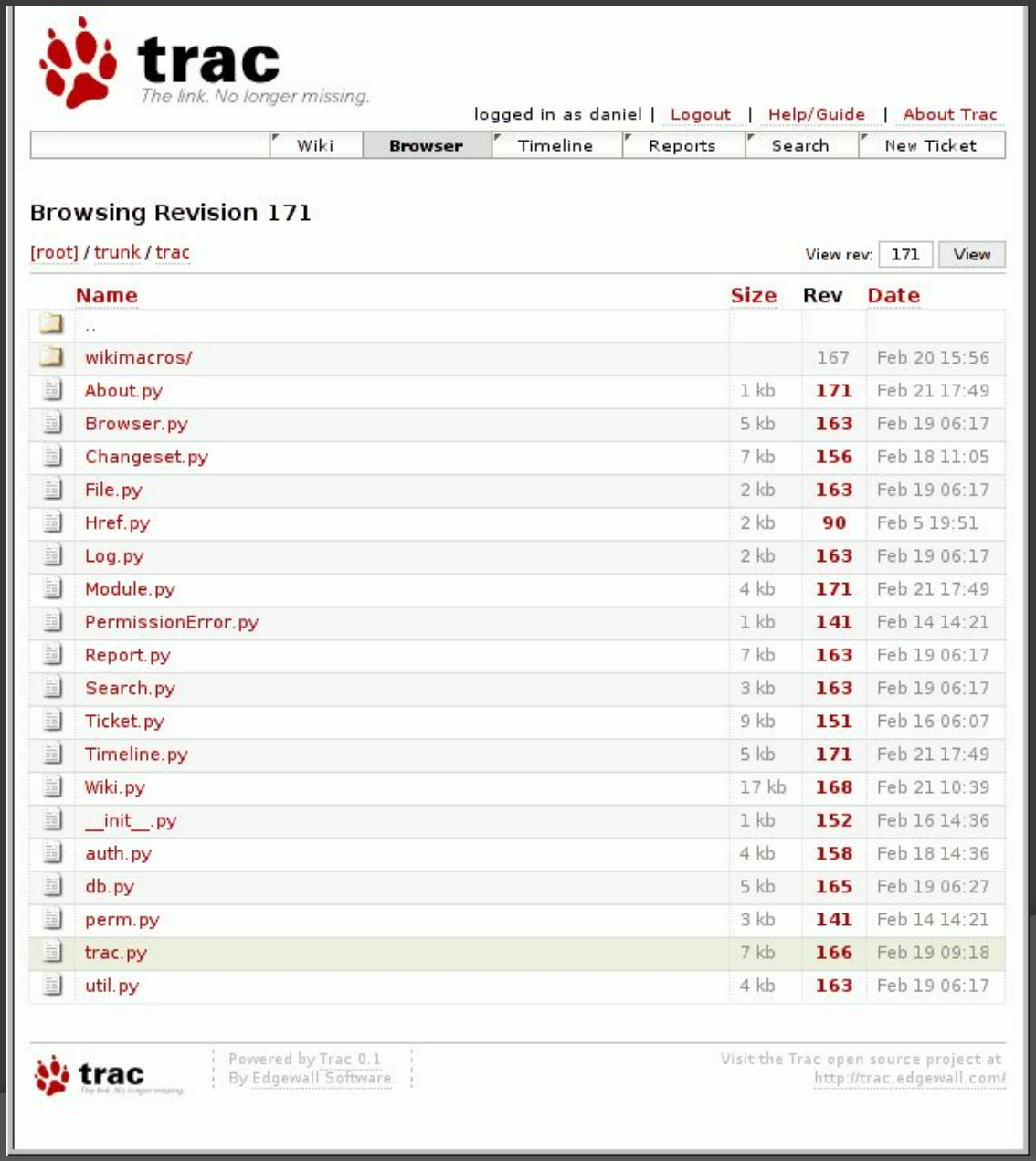
- A menedzsment számára biztosítható egy visszacsatolás
  - Jól látszik a fejlesztés menete
    - Ki min dolgozott és dolgozik éppen
- A verziókövető rendszerek sokszor összekapcsolhatók:
  - Feladatütemezővel, projekt manager eszközökkel (Pl. JIRA, TRAC)
  - Wiki-vel
  - Egyéb rendszerekkel (Pl. Bugzilla)
- Vizuális felületet nyújtanak a fejlesztés menetéről
  - Statisztikai adatok
  - Diagramok

JIRA – Fisheye kiegészítő



JIRA – Fisheye kiegészítő

Trac


trac
The link. No longer missing.
logged in as daniel | Logout | Help/Guide | About Trac
Wiki
Browser
Timeline
Reports
Search
New Ticket
Browsing Revision 171
[root] / trunk / trac
View rev: 171
View
Name
Size
Rev
Date
..
wikimacros/ 167 Feb 20 15:56
About.py 1 kb 171 Feb 21 17:49
Browser.py 5 kb 163 Feb 19 06:17
Changeset.py 7 kb 156 Feb 18 11:05
File.py 2 kb 163 Feb 19 06:17
Href.py 2 kb 90 Feb 5 19:51
Log.py 2 kb 163 Feb 19 06:17
Module.py 4 kb 171 Feb 21 17:49
PermissionError.py 1 kb 141 Feb 14 14:21
Report.py 7 kb 163 Feb 19 06:17
Search.py 3 kb 163 Feb 19 06:17
Ticket.py 9 kb 151 Feb 16 06:07
Timeline.py 5 kb 171 Feb 21 17:49
Wiki.py 17 kb 168 Feb 21 10:39
__init__.py 1 kb 152 Feb 16 14:36
auth.py 4 kb 158 Feb 18 14:36
db.py 5 kb 165 Feb 19 06:27
perm.py 3 kb 141 Feb 14 14:21
trac.py 7 kb 166 Feb 19 09:18
util.py 4 kb 163 Feb 19 06:17
trac
Powered by Trac 0.1
By Edgewall Software.
Visit the Trac open source project at
http://trac.edgewall.com/

Egyszerű verziókövetés

- A kódot minden nagyobb változtatás előtt egy-egy külön mappába mentjük
  - ezeket próbáljuk megfelelően megkülönböztetni
    - Pl. a külön könyvtárakat dátumokkal/verziószámokkal látjuk el
- Működőképes!
  - De a legkevésbé hatékony verziókezelési technika
  - Egyszemélyes fejlesztésnél csak
- **Probléma:**
  - idővel, nehézkessé válik megjegyezni a tartalmi különbségeket az egyes verziók között,
  - sok tárterület foglalhat
  - Nincs szoftver eszköz, ami extra funkciókat, segítséget nyújt
    - Pl. Diff - összehasonlítás

A VERZIÓKÖVETŐ RENDSZEREK FELÉPÍTÉSE…

Verziókezelési modellek

- Központosított modell (hagyományos)
- Elosztott verziókezelő rendszerek

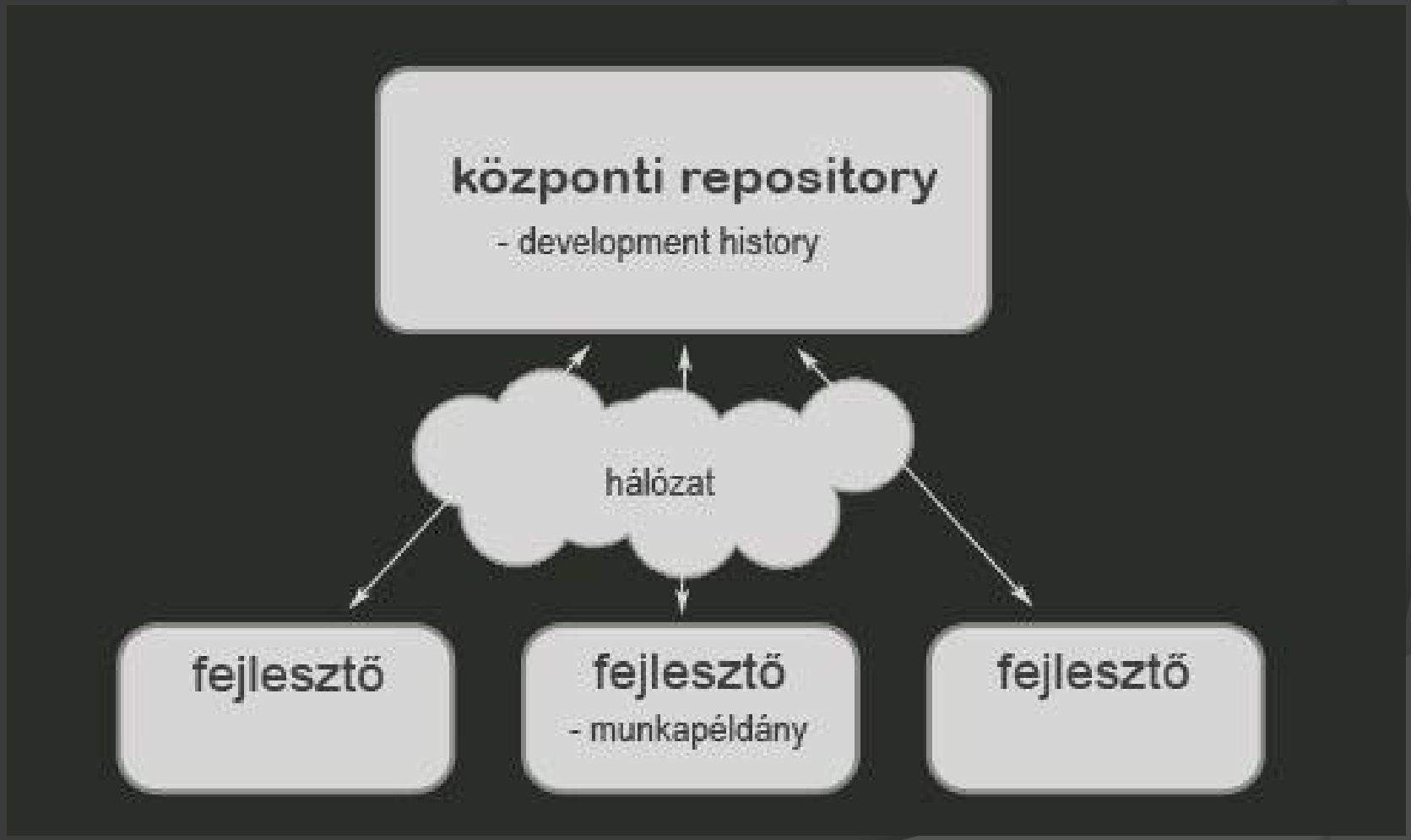
Központosított rendszerek

- Minden fejlesztő egy közös repository-t használ
  - Minden művelet ezen a szerveren hajtódik végre
  - Az adatbázis lehet egy külön gépen vagy akár ugyanazon is
- **A munkamenet:**
  - gyakorlatilag egy commit - update folyamatból áll.
  - Minden commit után ajánlatos frissíteni a working directoryt, hogy a mások által írt változtatások megjelenjenek.
- Probléma: egyidejű módosítás
  - el kell kerülni azt, hogy a felhasználók felülírják egymás munkáját

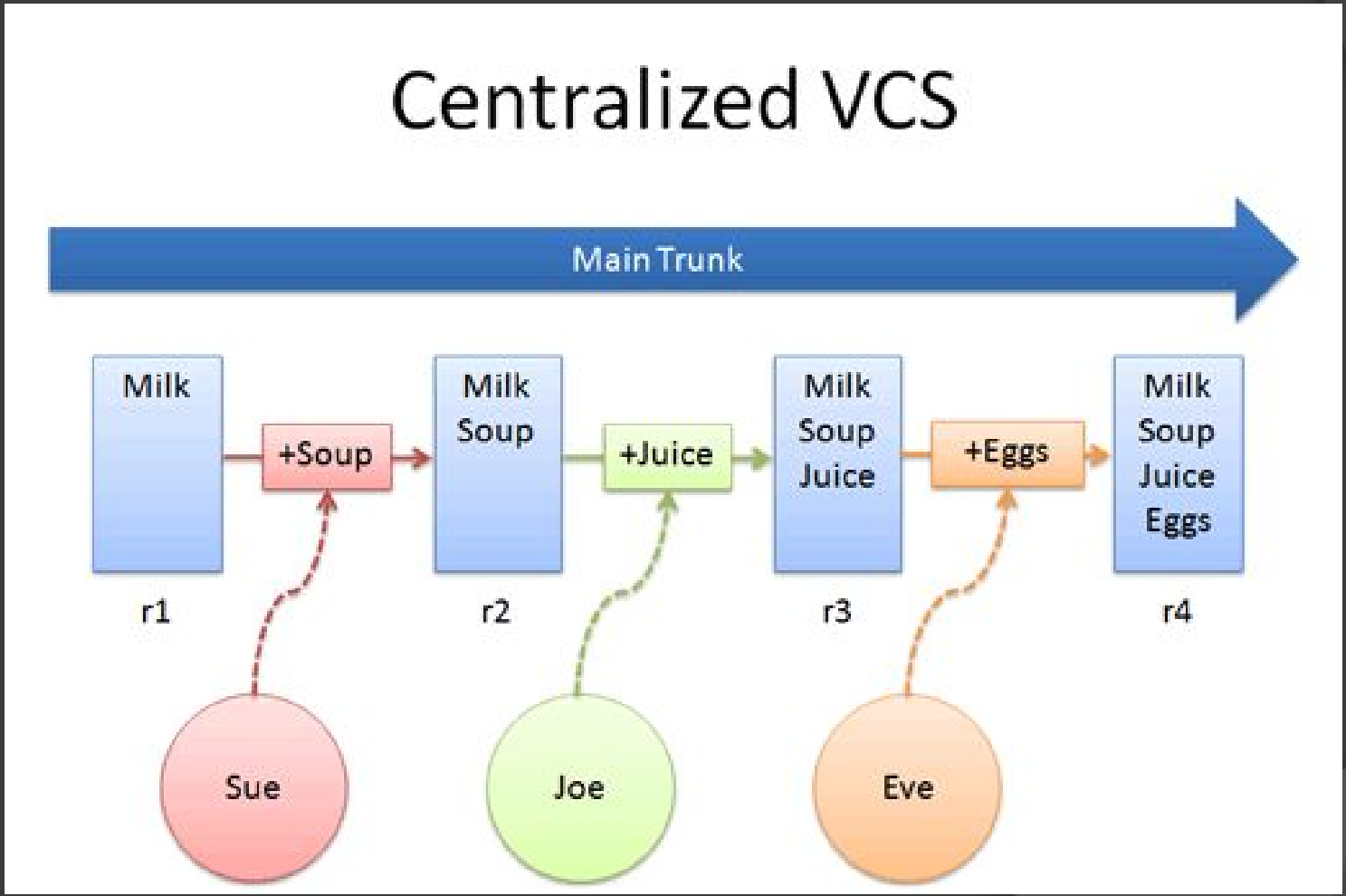
Központosított rendszerek

- **Az ütközések kezelésének két módja:**
  - **Lock (zárolás):** tilos a konkurens hozzáférés
    - ha valaki elkezd módosítani egy fájlt, akkor azt más felhasználó nem nyithatja meg írásra
    - Nagyobb vagy sok fájlt érintő változtatásoknál elkerülhető a bonyolult összefésülési művelet
    - Túl sokáig zárolt fájl problémát okozhat a többi felhasználónak
  - **Merge (összefésülés):** több felhasználó dolgozhat egyidejűleg ugyanazon a fájlon
    - Az elsőként módosítást küldő felhasználó sikerrel fog járni,
    - a rendszer a többi felhasználónak pedig összefésülési lehetőséget ad,
    - lehet automatikus vagy kézi

Központosított rendszerek



Központosított rendszerek

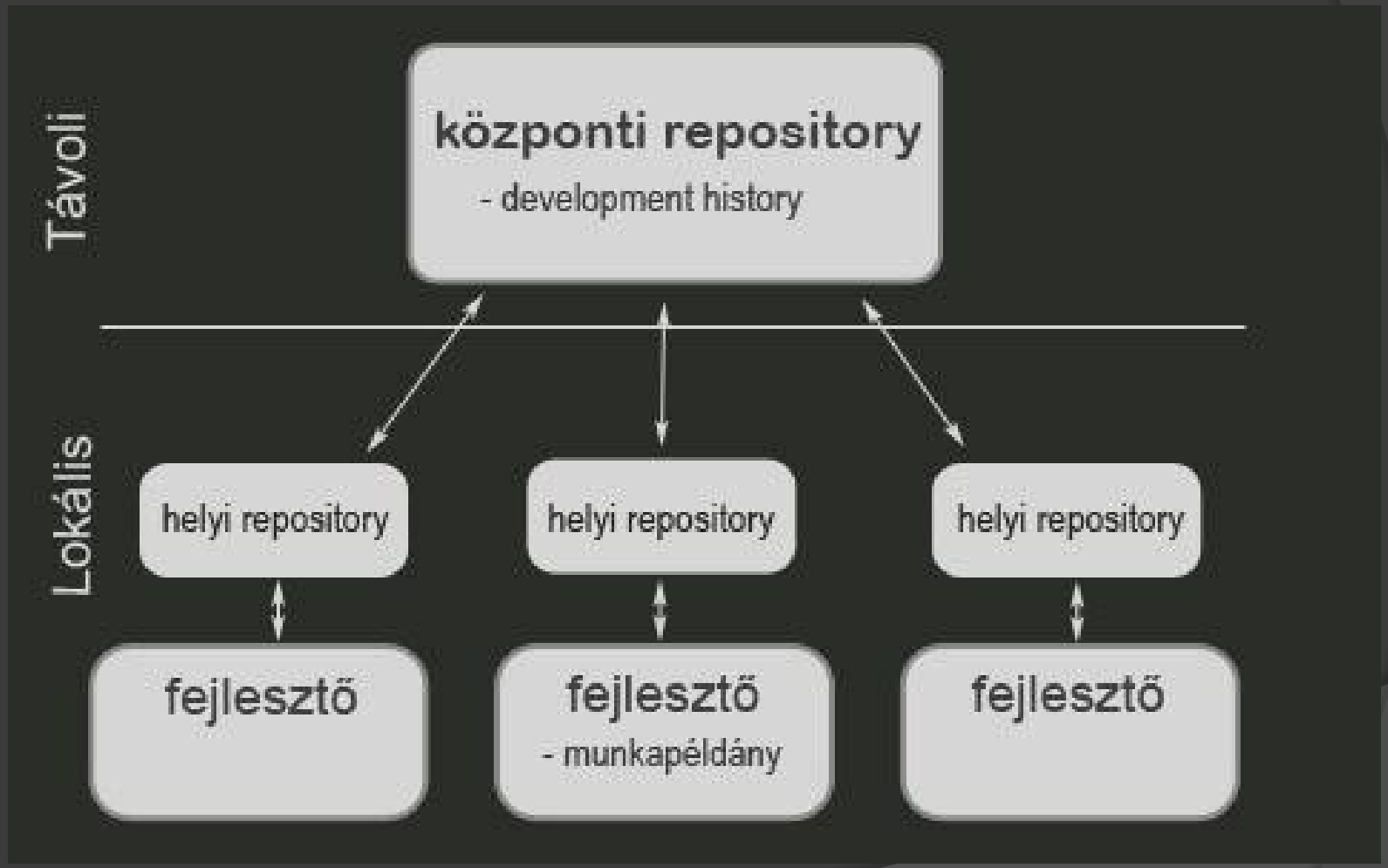


Mindenki szinkronizál és „becsekkeli” a változásait.

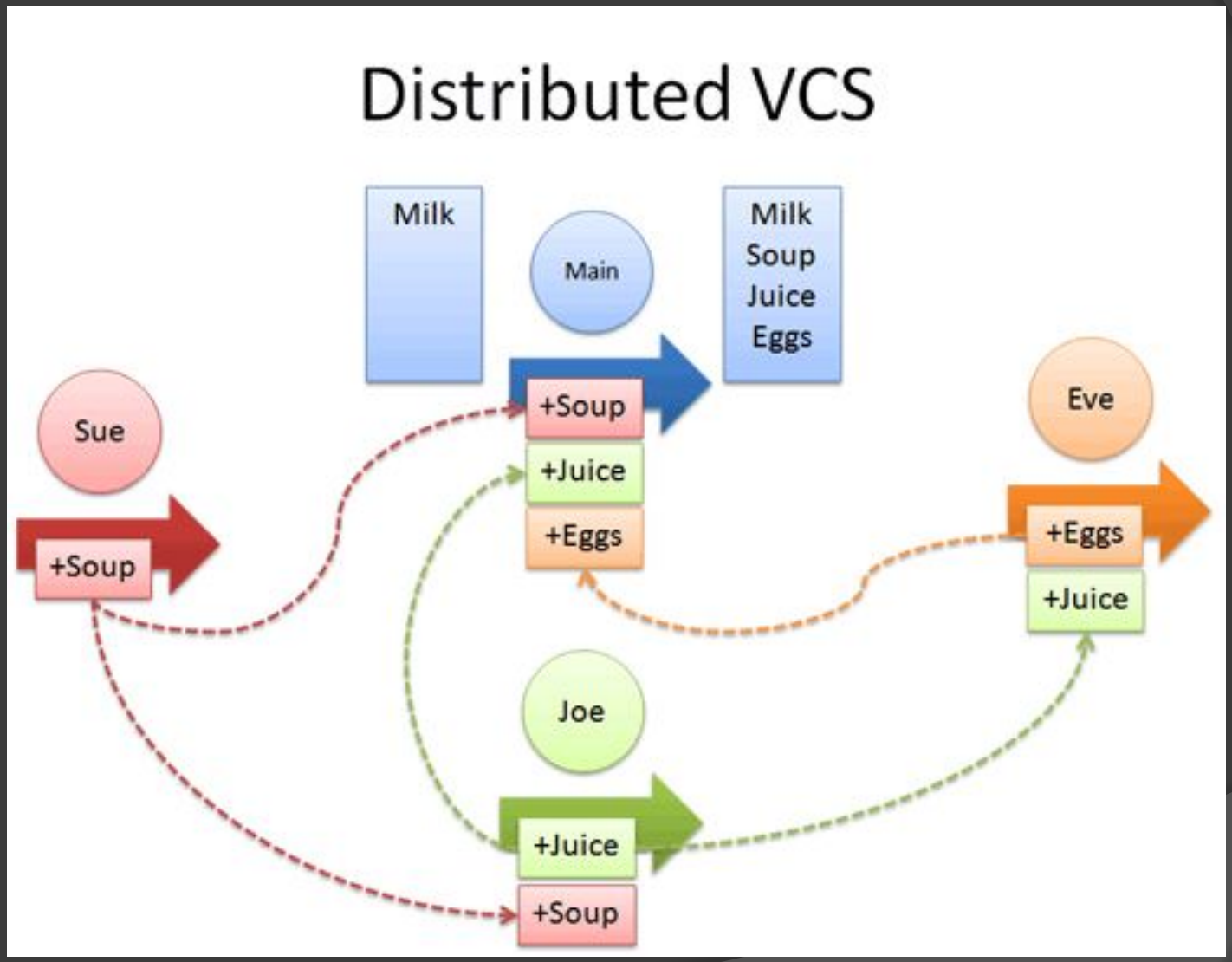
Elosztott v. rendszerek

- Egy központi tároló helyett minden felhasználó gépe egy-egy külön tárolóként jelenik meg,
  - csak munkamásolatok vannak
- A szinkronizáció az egyes gépek között küldött patch-ek (módosításcsomagok) által valósul meg
- **A modell jelentős változásokat okoz:**
  - **Offline is működik!**
  - A gyakori műveletek gyorsak, mert nem kell központi szerverrel kommunikálni
  - Egyszerű, rugalmas, gyors elágaztatás és összefésülés
  - Nem feltétlenül nyújt védelmet az adatvesztés ellen
  - Ütközések kezelése: elágaztatás, majd összefésülés (általában automatikusan)

Elosztott v. rendszerek



Elosztott v. rendszerek



Mindenki a saját repo-ba comitál, majd push-olja a központiba

ALAPFOGALMAK…

Fontosabb fogalmak

- A verziókezelő szoftverek logikai működése eltér(het) egymástól
  - Az alkalmazott fogalmak azonosak!

- **Repository:** röviden csak repo. Maga a tárolónk.
  - tárolja az összes projekthez tartozó fájlokat és azok verzióit
  - általában egy speciális könyvtárszerkezet speciális fájlokkal
  - minden projektet tehát külön repo-ban kell tárolni!

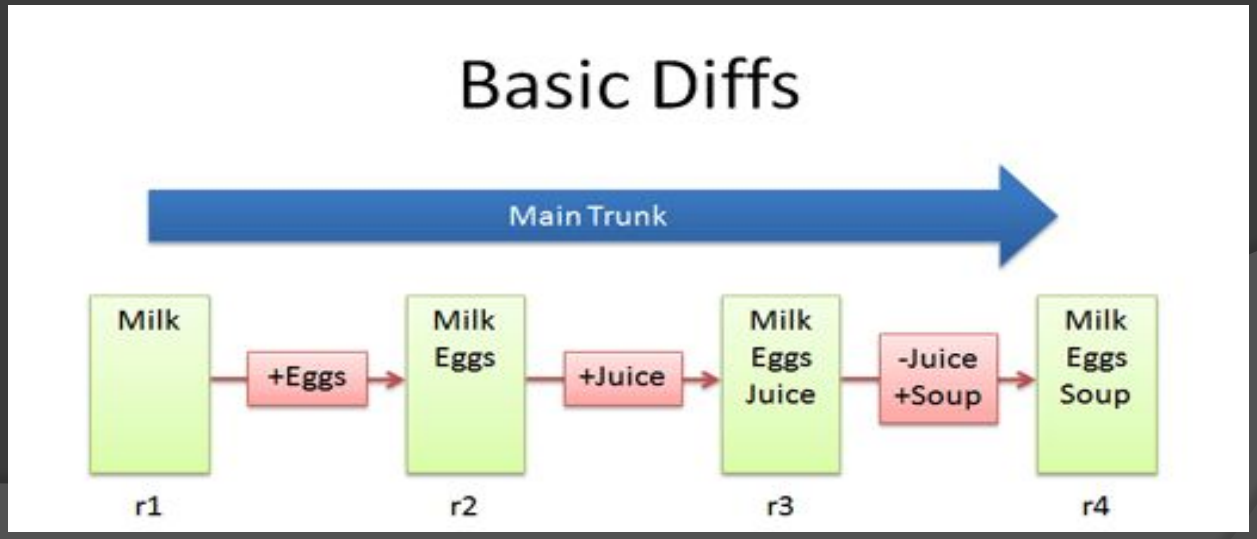
- **Working copy:**
  - A kód egy részének egy példánya, amelyen a fejlesztő éppen dolgozik a saját gépén.
  - A munka befejeztével ennek állapota kommitok formájában tárolásra kerül a repositoryban

Fontosabb fogalmak

- **Commit:**
  - A kódon eszközölt változtatásokat úgynevezett kommitok formájában érvényesíthetjük a tárolókon belül,
  - A tárolók mintegy pillanatképként tartalmazzák azokat, illetve projektünk aktuális állapotát.
  - Ha zsákutcába futnánk fejlesztés közben, akkor ezek alapján kereshetjük vissza kódunk korábbi verzióját.
  - Ezért célszerű minden nagyobb módosítást követően kommitolnunk.
- **Revision:** verzió
  - Minden kommit után egyel növekszik az repoban a revision értéke, azaz a verziószám
- **Checkout:**
  - Lokális másolat készítése valamely verziókezelt fájlról.
  - Alapértelmezésben ilyenkor a legfrissebb verziót kapja a felhasználó,
    - de van lehetőség konkrét verzió kikérésére is verziószám alapján.

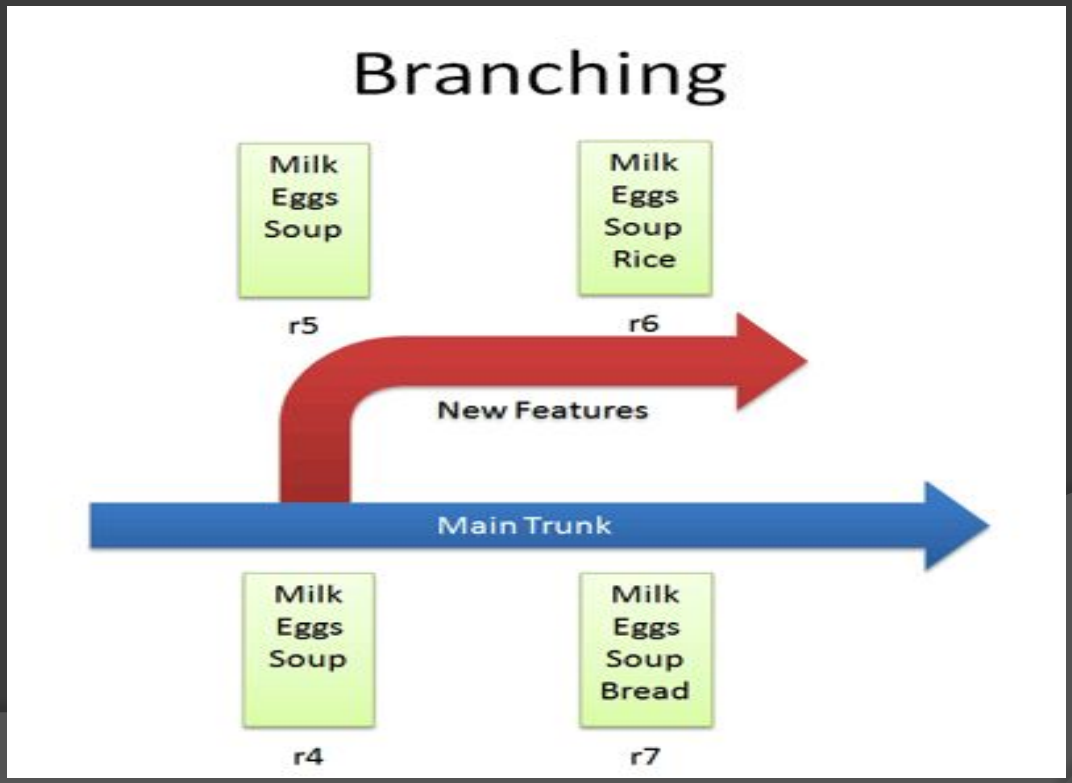
Fontosabb fogalmak

- **Head:** a legfrissebb kommitot (verziót) jelöli, az aktuális ág teteje.
- **Pushing:** adatok feltöltése a központi repoba (Pl. git, mercurial)
- **Trunk:** A fejlesztés fő ágát képviseli. Lényegében egy speciálisan elnevezett „branch”
- **Update:** a repoban lévő változásokat dolgozza be a felhasználó working copy-jába, tehát a lokális verzióba.
- **Diff/Change/Delta:** két file között változás megtalálása/mutatása.



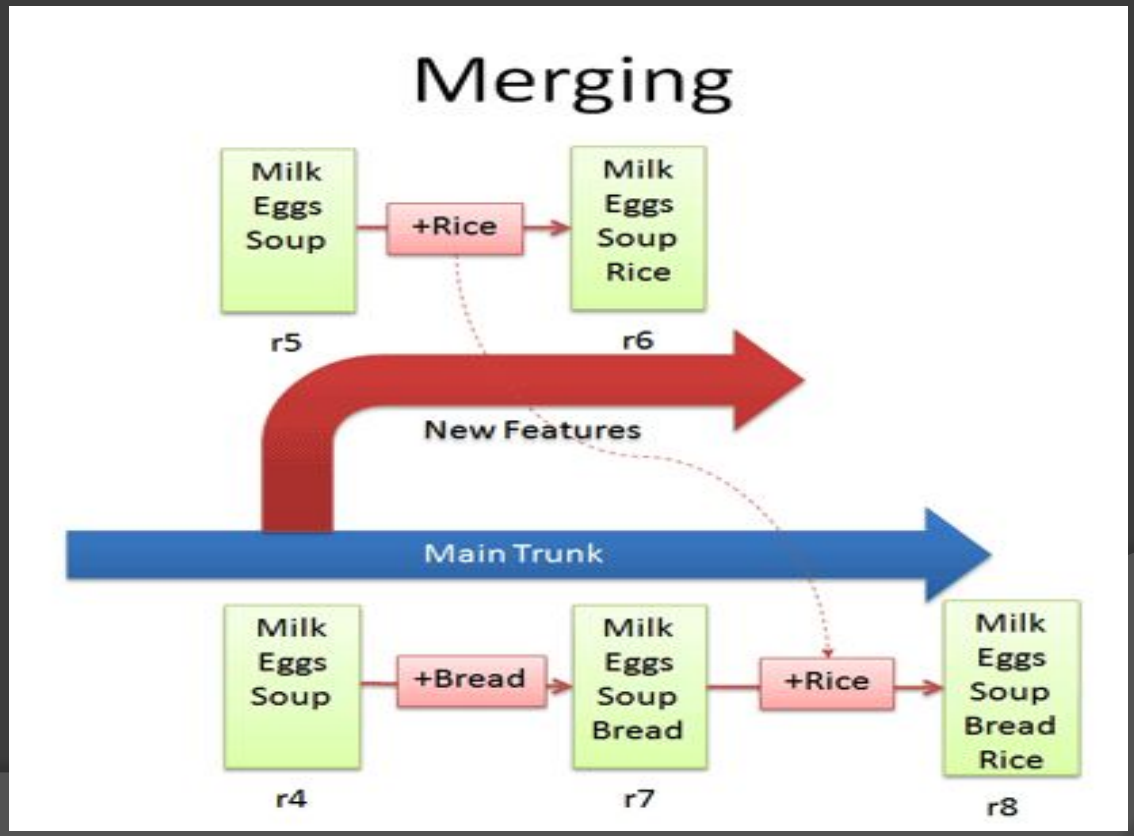
Fontosabb fogalmak

- **Branch:** fejlesztési ág
  - Egy alternatív fejlesztési útvonalat képvisel
    - Pl. ha projektünket esetleg a tervezettől eltérő irányban is szeretnénk továbbfejleszteni.
  - Az eredeti verziót érintetlenül hagyva tudunk kísérletezni.
  - Optimalizált helyfoglalás, nem egyszerű másolás

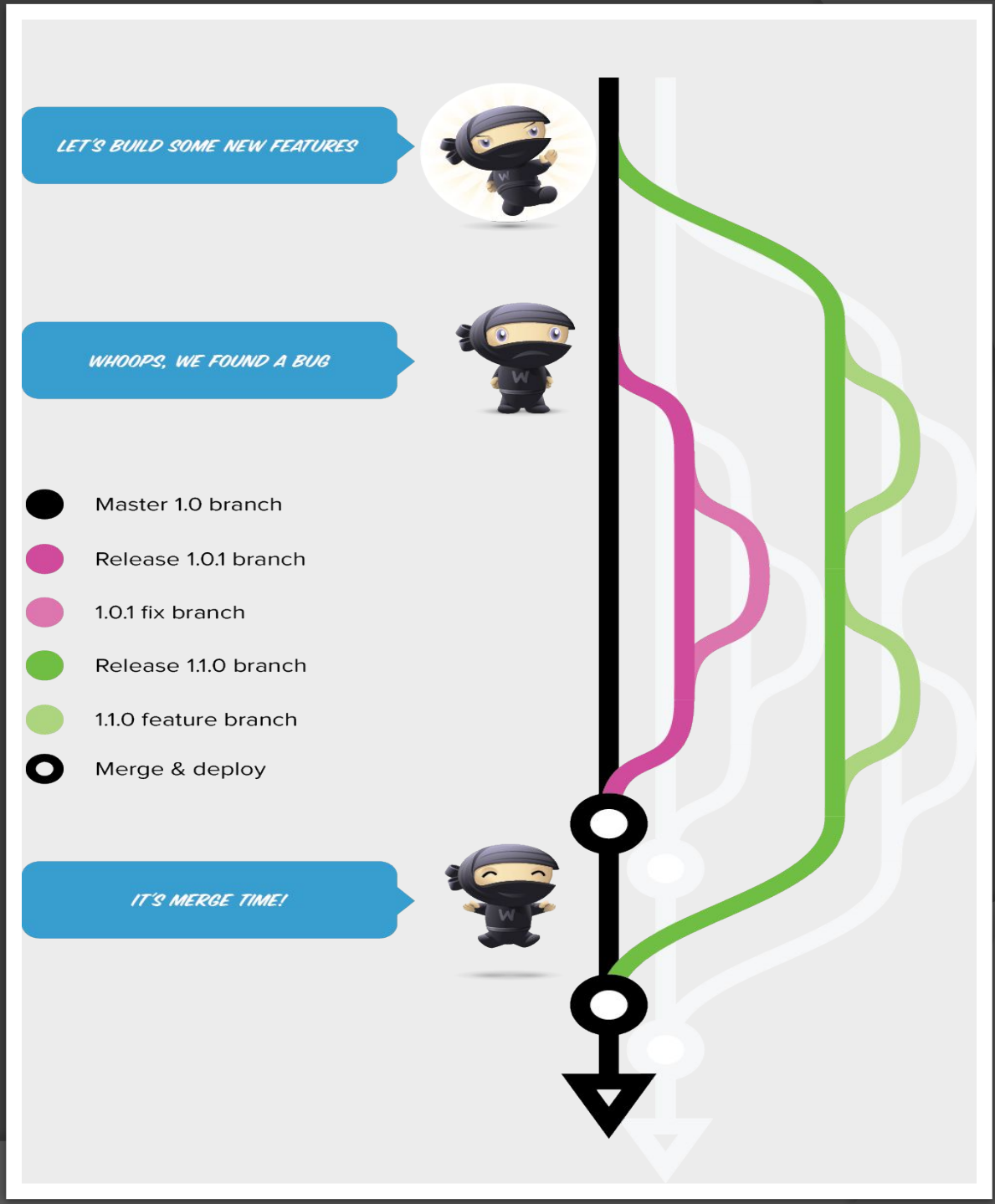


Fontosabb fogalmak

- **Merge:** összefésülés
  - A fejlesztési ágak létrehozása mellett lehetőségünk van ezek egyesítésére is.
  - Ennek folyamata az összefésülés - **merging**



Merge


LET'S BUILD SOME NEW FEATURES
WHOOPS, WE FOUND A BUG
IT'S MERGE TIME!
Master 1.0 branch
Release 1.0.1 branch
1.0.1 fix branch
Release 1.1.0 branch
1.1.0 feature branch
Merge & deploy

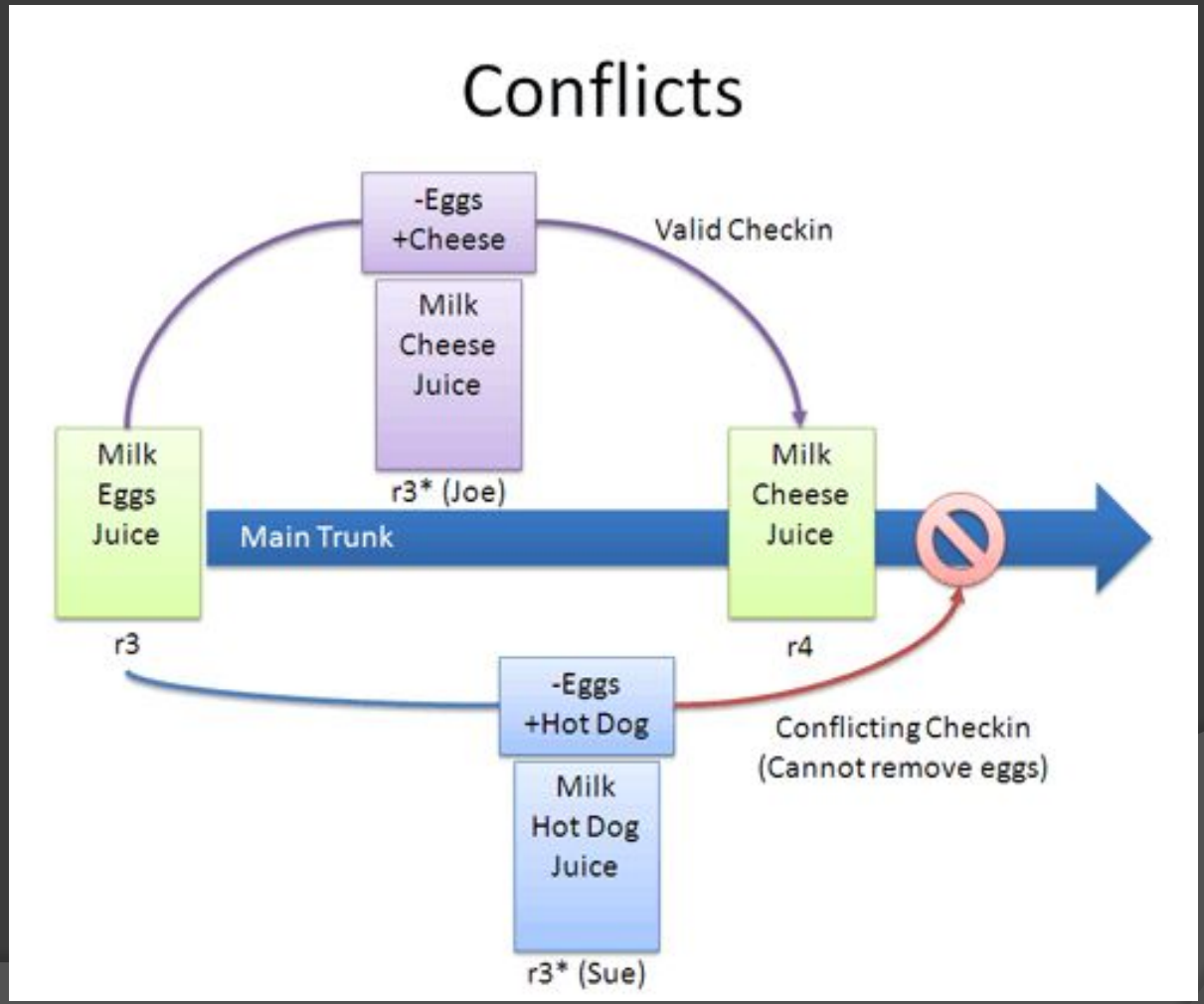
Fontosabb fogalmak

- **Conflict:**
  - Ágak összefésülése során keletkező jelenség
  - A két ág verziója olyan kódot tartalmaz, amit nem lehet automatikusan összefésülni
  - Manuálisan kell elvégezni az összefésülést
  - A modern IDE-k grafikus felületet biztosítanak erre
- **Példa:**

```
<<<<<<< .mine
This is fun stuff!
=======
This is a documentation file
>>>>>>> .r6
```

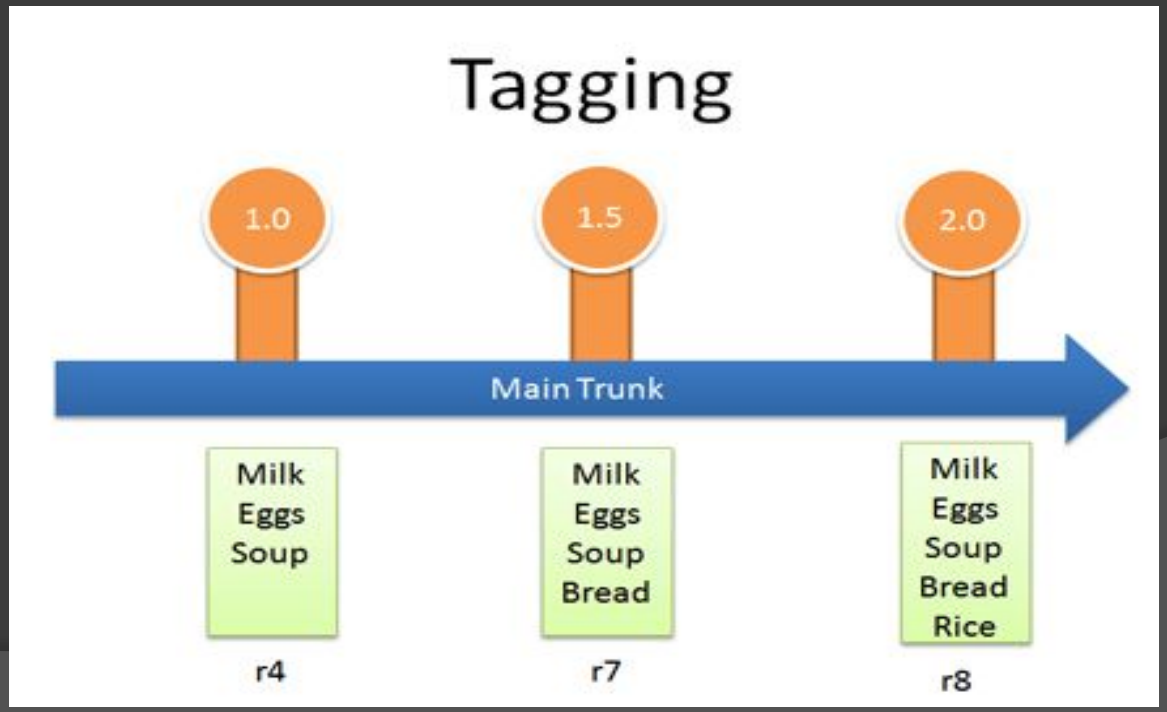
Fontosabb fogalmak

- **Conflict:**



Fontosabb fogalmak

- **Verzió tag-elés (tagging):**
  - olyan branch, amely megállapodás alapján nem kerül szerkesztésre
    - Lényegében egy mentés az adott verzióról
  - Speciális verziókat jelölhetünk vele névvel:
    - Pl. Webshop 1.0, Super Mario 1.3



VERZIÓKEZELŐ RENDSZEREK A GYAKORLATBAN…

Verziókezelő rendszerek csoportosítási szempontjai

- Repository modell szerint (központi, elosztott)
- Támogatott platformok (Linux, Windows,..)
- Költsége (ingyenes, fizetős, illetve licensze)
- History modell (changeset, patch, snapshot)
- Verzió-azonosító (Revision ID: namespace, sequence, pseudorandom)
- Hálózati protokoll (http, https, ftp,sftp,ssh)
- Nyílt vs. Zárt forráskód

Ismertebb verziókezelő rendszerek

- Concurrent Versions System(CVS): ingyenes, központi, egyik legrégebbi
- Subversion(SVN): ingyenes, központi, modern CVS
- Git: ingyenes, elosztott. Pl. Linux forráskód
- Mercurial: ingyenes, elosztott
- Bazaar: ingyenes, elosztott
- Bitkeeper: fizetős, elosztott
- Visual SourceSafe: Microsoft, shared folder alapú, fizetős

SVN áttekintés

- Nyílt forrású verziókezelő rendszer
  - Unix, Linux, Windows, OSX, BSD, Solaris, BeOS, Haiku, stb
- Használatával a fájlok és könyvtárak időbeli változásait kezelhetjük.
- A tároló hasonlít egy átlagos fájlkiszolgálóra,
  - kivéve hogy a fájlok és könyvtárak minden módosítását feljegyzi.
- **Mit nyújt:**
  - teljes körű verziómenedzsment parancssorból

http://subversion.apache.org/

SVN áttekintés

- **SVN kiszolgáló:**

  svnserve - Linux

  svnserve.exe - Windows

- Beépített pehelysúlyú szerver
  - Installációs csomaggal együtt települ
  - TCP/IP protokollon keresztül kommunikál
  - Saját protokoll - svn://
  - Képes ssh tunnelen kommunikálni

Démonként való futtatása:

svnserve.exe -d -r c:/MySVNRepo

SVN hozzáférési módok

Séma	Hozzáférési mód
file://	közvetlen tárolóelérés (helyi lemezen)
http://	A Subversion-t ismerő Apache webkiszolgáló elérése WebDAV protokollon keresztül
https://	Ugyanaz, mint a http://, de SSL titkosítással
svn://	svnserve kiszolgáló egyedi protokollon való elérése
svn+ssh://	Ugyanaz, mint az svn://, de SSH alagúton keresztül

SVN használat

- Repo létrehozása (szerver):létrejön az alapvető file struktúra

svnadmin create MyRepo

- Working copy létrehozása (svn checkout):
  - kliens oldalon egy munkamásolat létrehozása

svn checkout repohelye hovátegye

Pl.

svn checkout http://example.org/svn/MyRepo C:/LocalRepo

Példa ssh tunnelre:

svn co svn+ssh://example.org/svn/MyRepo C:/LocalRepo

SVN használat

- Új file hozzádása a working copy-hoz:

  svn add akarmi.txt

- Fájl törlése:

  svn del akarmi.txt

- Változások komittálása a repo-ba:
  - Minden változás bekerül a repository-ba

  svn commit –m `Kommit szoveg ide`

- Változások letöltése a repo-ból:

  svn update

SVN használat

- File verzió visszagörgetése:

```
svn  revert  test.c
```

- Branch létrehozása:

```
svn copy svn+ssh://example.com/svn/MyRepo/trunk
svn+ssh://example.com/svn/MyRepo /NAME_OF_BRANCH
    -m "Creating a branch of project
```

- Összefésülés:
  - ág visszafésülése a fő ág 250-es revision-jébe

```
svn merge -r 250:HEAD
http://example.com/svn/MyRepo/branches/my-branch
```

SVN használat

- Verzió tag-elés:

```
svn copy http://path/to/revision http://path/to/tag
```

Ismertebb SVN kliensek

- Tortoise SVN, RapidSVN
- A fejlettebb verziókezelők lehetővé teszik az integrációt más eszközökkel
- Különböző IDE-khez gyakran letölthetők verziókezeléssel kapcsolatos kiegészítők
  - Grafikus diff, merge, commit, revert
  - Szinkronizációs nézet, history, stb
- **Eclipse/Netbeans alapú kliensek:**
  - Subversive
  - Subclipse

Ismertebb SVN szerverek

- Számtalan online szolgáltatás verziókövetésre
  - Ingyenes és pénzes
- **Integrált eszközök:**
  - Modern web-es felület
  - Több verziókövető rendszer támogatása
  - Repository létrehozása, menedzselése online
  - Wiki oldalak
  - Felhasználók menedzselése
  - Bugtracker rendszer
  - Agilis fejlesztési kiegészítők
  - Egyéb lehetőségek: Pl. diagramok rajzolása, stb

Google, Assembla, Bitbucket, SourceForge, BerliOS, DevGuard, stb

Hogyan szervezzük a repository-t?

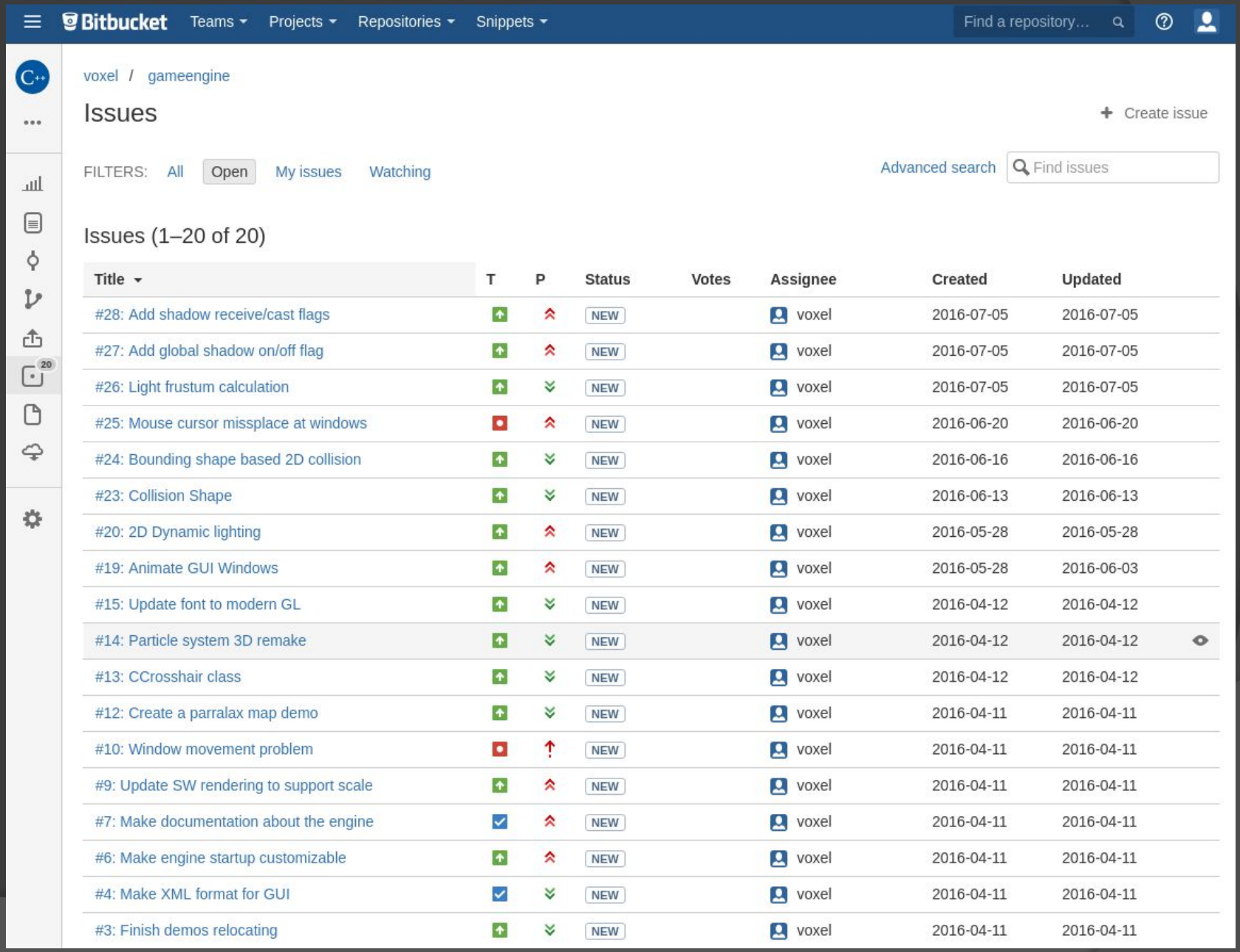
Repository szervezése

- Ma minden komolyabb projektet verziókövetnek
- Ez megfelelő repository szervezést igényel
- **Miért?**
  - Tudni kell ki mit csinált és mikor
  - A kód biztonsága mindennél fontosabb
  - Menedzselni kell a
    - kiadásokat - névvel / számmal ellátva
    - verziókat - megfelelő számozást igényel
    - fejlesztői ágakat
    - egyéb részeket / elágazásokat
  - Commit-ok összekapcsolása az Issue Tracking rendszerekkel

Repository szervezése

- **Egy tipikus fejlesztés menete**
  - A fejlesztő csapat Issue Tracking rendszert használ
  - A csapat hetente legalább 1x
    - rendszerezi a problémákat
      - bug, issue
    - felveszi az új fejlesztési taszkokat
    - A csapat priorizálja a feladatok
      - Fejlesztési modelltől függőben új fejlesztési ciklust indít
        - pl. Sprint
  - Az Issue Tracking beli bugok, taszkok számmal és névvel azonosítottak
    - pl. ISSUE 1357 - Login ablak nem működik Opera böngészőn

Bitbucket példa



Repository szervezése

- A fejlesztés során több ág (branch) használata indokolt
  - fejlesztésre, kiadásokra, egyéb területekre
- A fő fejlesztési ág minden esetben a **trunk**
  - mindenki ide fejleszt, commit-ol
    - új funkciók
    - hibajavítások, egyebek
- **A commit-ok formája:**
  - Egy commit komment részének specifikusnak kell lennie
    - Az elnevezés össze kell kapcsolja az Issue Rendszer taszkjaival
    - Szabály (nem kőbe vésett):
      - a komment tartalmazza az issue számát és cím szövegét
      - Pl. Issue - 125. Fix login window Opera browser problem
    - **Issue szám nélküli kommentek nem kívánatosak, de előfordulhatnak!**

Repository szervezése

Mire valók a többi branch-ek?

- **1. Program kiadások (pl. Play Store):** a szoftverből élete során több kiadás készülhet. Pl. 1.0, 1.2, 2.0, stb
  - A kiadásokat is a verziókövető rendszernek kell menedzselni!
  - Hogyan?
    - Kiadás esetén az aktuális trunkból egy branch-et készítünk
    - minden release egy megfelelő névvel ellátott branch lesz
      - Pl. RELEASE_1, RELEASE_1_1

Repository szervezése

- Miért jó a külön branch a kiadásoknak?
  - a kiadásokban felfedezett hibák is javíthatók
    - mivel a trunk már egyéb funkciókat is tartalmazhat (messzebb jár), így az nem használható erre
- Hiba javítása mindig a branchben történik:
    - a) Átállás az aktuális release branch-re
    - b) Hiba javítása
    - c) Javítás commitolása a branch-be
    - d) esetleg új kiadás készítése
  - Ha a hiba trunk-ban lévő verziót is érinti, akkor ott is javítani kell
    - vagy a release branch-ben lévő módosítást vissza merge-elni a trunk-ba

Repository szervezése

Mire valók még a branch-ek?

- **2. Új, nagy horderejű változás:**
  - Bizonyos új fejlesztések külön ágat igényelnek
  - **Oka:**
    - Ne zavarja meg a trunk fejlesztését, mert nagy horderejű változás
      - a szoftver főbb részei nem fognak működni
      - gátolja a többi fejlesztő tevékenységét
    - Sokszor csak kísérleti fejlesztés
      - Esetleg új API-k tesztelése
      - Bizonyos részek lecserélése, stb
  - Sikeres fejlesztés után a változásokat visszavezetik a trunk-ba

Szoftver verziók számozása...

Verzió számozása

- A szoftver életciklusa során számos változáson esik át
  - több verzió és kiadás is elkészülhet
  - **ezeket menedzselni kell**
- A megfelelő verziószámozás és értelmezése fontos!
  - Historikus jelentőssége van
    - A kérdéses verzióra mindig vissza lehessen menni, az
    - Az állapot megmaradjon
      - hibajavítás, egyéb okok miatt
- Primitív verziózás és kiadás:
  - a szoftver kiadása a trunk ág head-jéből készül
  - számozás inkrementálisan történik, de “hasraütésre”

Verzió számozása

- Számos verziószámozási séma létezik
  - Nincs legjobb,
  - Bármelyik testre szabható a további igényeinknek megfelelően
    - A lényeg a szoftver kiadásaiba vitt rendszer
- Már a szoftver fejlesztési ciklus elején dönteni kell valamilyen séma mellett
  - logikussá teszi a fejlesztési és kiadást
  - nem zavarja össze a user-eket sem

Szemantikus verziószámozás

- **Semantic versioning** - http://semver.org/

- Egy széles körben elfogadott szabályrendszer
  - Definiálja verzió számozását
    - részletes, precíz
  - Főként olyan rendszereknél
    - ahol sok az iteráció,
    - gyakoriak a kiadások,
    - sok a függőség (dependency)
  - Tipikus példa az egyes library-k
    - Pl.: LibreOffice_5.2.0_Linux_x86-64_rpm.tar.gz

Szemantikus verziószámozás

- **Miért van rá szükség?**
  - elkerüljük a “dependency hell”-t
- Példa
  - Legyen egy library, neve “Tűzoltó”
  - A Tűzoltó lib számára szükséges a “Létra” szemantikusan verziószámozott komponens
  - Amikor a tűzoltó lib-et létrehozták, akkor a létra verziószáma 3.1.0 volt.
  - A tűzoltó lib számára függőségként definiálhatjuk, hogy a szükséges létra komponens verziószáma 3.1.0 <= XX < 4.0.0 kell legyen
  - Ha a létra komponens verziót lép, pl. 3.2.0, akkor beengedhető tűzoltó build rendszerébe
  - A szemantikus számozással garantálható a függőségek kompatibilitása

Szemantikus számozás

- **Egy szoftver verziója:**

Major.minor.patch

M.m.p

Major: olyan verzióváltást, API változást jelöl, amelyek inkompatibilisek egymással

- Azaz nem cserélhető ki egymással, nem frissíthetők kódjavítás nélkül
- Pl.: SDL_1.2 < - > SDL_2.0

Minor: olyan hozzáadott plusz funkció változások az API-ban, amelyek visszafelé is kompatibilisek egymással.

- Pl.: SDL_1.1 és SDL_1.2

Patch: visszafelé kompatibilis bugfix-ek egymással.

- Pl.: Facebook Android API: 4.14.0, 4.14.1

Szemantikus verziószámozás

- A verziószámokat mindig növeljük:
  - ha M-et növeljük, akkor m.p 0.0 lesz,
  - ha m-et növeltük akkor p lesz 0, M marad ami volt

- A verziók sorrendje értelemszerűen balról jobbra történik
  - tehát 1.2.3 korábbi verzió, mint 2.1.1, ami korábbi, mint 2.2.0 ami viszont későbbi, mint 2.1.6789

Szemantikus verziószámozás

- A növelés mértéke általában 1
  - Ha egy release készítés elromlik valamiért, ugorhatunk több verziót is.
    - Pl. készül 1.5.3 verziójú release / branch stb
      - de valamiért hibás
    - Kiadunk egy új verziót, ami az 1.5.4 lesz
      - dokumentáljuk, hogy az 1.5.2 után az 1.5.4 jön,
      - az 1.5.3 pedig mintha nem is lenne,
      - még akkor is, ha ezzel a verzióval látszik is valami valahol

Szemantikus verziószámozás

- A szemantikus verziózás megengedi az egyedi elnevezéseket is:
  - Pl. A “-” jel után olyan pre-release verzió jelöléseket adjunk meg, mint alpha1, alpha2, beta9
  - A “-” utáni részben lehetnek pontok is
  - A sorrendiség szempontjából ilyenkor is balról jobbra történik az összehasonlítás ASCII sorrendben
    - Azaz 1.1.0-1 korábbi, mint 1.1.0-alpha
  - A verzió szám végére “+” után oda lehet tenni build információkat is,
    - de két verzió nem szabad, hogy csak ebben különbözzön

Amit nehéz feloldani

- Egy projektben egy régebbi verzióban, pl. **3.2.0** hiba van
  - Az XY ügyfél kér egy hibajavítást, és lesz 3.2.1.
  - De közben kiderül még egy hiba is, egy másik, QZ ügyfélnél és így azt is ki kell javítani a 3.2.0-ban.
- **Mi legyen a verziószámokkal:** lesz egy 3.2.1a meg egy 3.2.1b ?
  - Mert az ‘a’ hiba a QZ ügyfélnél nem okoz gondot.
    - Náluk nem jött elő. Oly módon használják a szoftvert, hogy az ‘a’ hiba semmi gondot nem okozhat.
    - A javítása, viszont potenciálisan hibaforrás, ezért ők nem akarnak egy 3.2.1-re épülő 3.2.2 verziót használni.
    - Csak arra a hibára akarnak egy javítást, amelyik a ‘b’ hibát javítja a 3.2.0 verzióban. Hasonlóan van ezzel az XY ügyfél is.

Amit nehéz feloldani

- Egy lehetséges megoldás:
  - a szemantikus verzió által megengedett “mínusz és valami” jelölés

Példa

- 3.2.1-XY
- 3.2.1-QZ

Mercurial röviden...

Mercurial (HG) használata

- **Mercurial repo létrehozása:**
  1) mkdir project
  2) cd project
  3) **hg init**
- **Fájlok hozzáadása:**
  1) create hello.txt
  2) **hg add hello.txt**
- Commit:
  hg commit -m "adding initial version of hello.txt"
- Változások elmentése a szerverre
  hg push

Mercurial (HG) használata

- **Létező repo klónozása**

  hg clone http://example.com/repo/hello my-hello

- **Változások letöltése a repo-ból**

  hg pull

- **A lekért változások alkalmazása a helyi repo-ra:**

  hg update

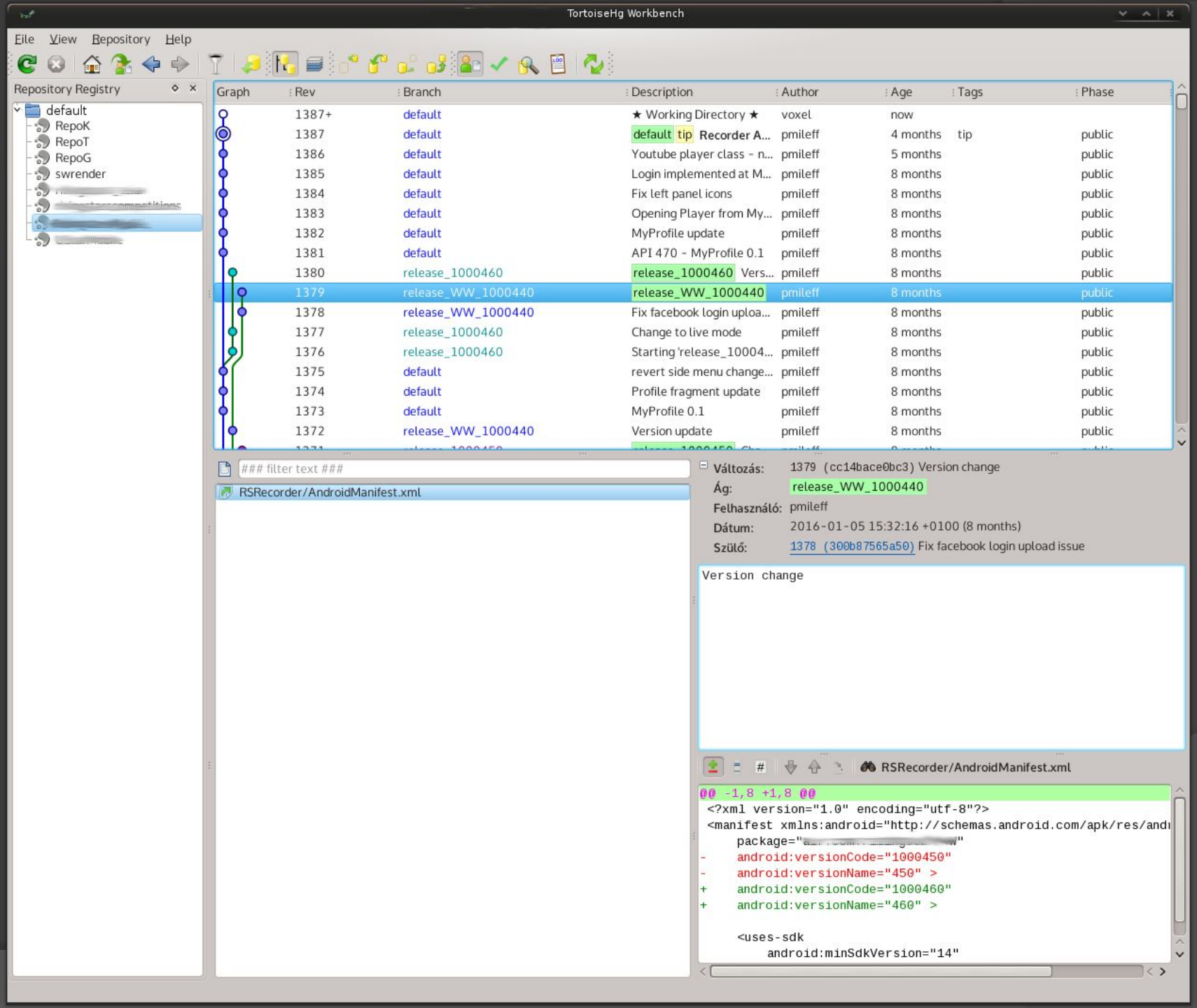
- **Merge:**

  hg merge

- **Repo státusz:**

  hg summary

  hg log


TortoiseHg Workbench
File View Repository Help
Repository Registry
default
RepoK
RepoT
RepoG
swrender
Graph Rev Branch Description Author Age Tags Phase
1387+ default ★ Working Directory ★ voxel now
1387 default default tip Recorder A... pmileff 4 months tip public
1386 default Youtube player class - n... pmileff 5 months public
1385 default Login implemented at M... pmileff 8 months public
1384 default Fix left panel icons pmileff 8 months public
1383 default Opening Player from My... pmileff 8 months public
1382 default MyProfile update pmileff 8 months public
1381 default API 470 - MyProfile 0.1 pmileff 8 months public
1380 release_1000460 release_1000460 Vers... pmileff 8 months public
1379 release_WW_1000440 release_WW_1000440 pmileff 8 months public
1378 release_WW_1000440 Fix facebook login uploa... pmileff 8 months public
1377 release_1000460 Change to live mode pmileff 8 months public
1376 release_1000460 Starting 'release_10004... pmileff 8 months public
1375 default revert side menu change... pmileff 8 months public
1374 default Profile fragment update pmileff 8 months public
1373 default MyProfile 0.1 pmileff 8 months public
1372 release_WW_1000440 Version update pmileff 8 months public
filter text
RSRecorder/AndroidManifest.xml
Változás: 1379 (cc14bace0bc3) Version change
Ág: release_WW_1000440
Felhasználó: pmileff
Dátum: 2016-01-05 15:32:16 +0100 (8 months)
Szülő: 1378 (300b87565a50) Fix facebook login upload issue
Version change
RSRecorder/AndroidManifest.xml
@@ -1,8 +1,8 @@
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/andr
package="
- android:versionCode="1000450"
- android:versionName="450" >
+ android:versionCode="1000460"
+ android:versionName="460" >
<uses-sdk
android:minSdkVersion="14"

GAME OVER