3.

Egy játékszoftver fejlesztésén több programozó dolgozik. Feladata, hogy a projektben résztvevők számára kialakítsa a közös fejlesztési környezetet, valamint gondoskodjon az egyes programváltozatok és frissítések elkészítéséről.

- Határozza meg a közös fejlesztés során jelentkező problémákat!
- Ismertesse a verziókezelő rendszerek főbb jellemzőit!
- Határozza meg – egy tetszőlegesen választott verziókezelő rendszerre alapozva – a programozók számára a közös munkafolyamatokat és szabályokat!
- Jellemezze a szoftverek életciklusát!
- Ismertesse egy programfrissítés készítésének módját!

A tételhez segédeszköz nem használható.

A fejlesztés egy „valamire is való” rendszer esetében a ma már kizárólag csapatmunkában történik.

Ehhez elengedhetetlen a csapat tagjai közötti megfelelő információcsere biztosítása. Ennek megfelelően a követelményeket és az egységes kép kialakítása érdekében a modelleket is dokumentálni kell.

Szintén kulcsfontosságú, hogy a modell leírása kellőképpen egyértelmű legyen ahhoz, hogy mindenki számára ugyanazt jelentse, vagyis hogy a fejlesztésben résztvevők mindegyikében (a megrendelőt is beleértve) a feladatról ugyanaz a kép alakuljon ki. Ennek megfelelően a modelleket kellőképpen formális módon kell leírni. A modell kellően precíz leírásával elérhetjük azt is, hogy a modellben levő ellentmondások a felszínre kerüljenek.

UML, mint jelölésrendszer

Mint már volt róla szó a fejlesztés általában csapatmunkában történik, ennek megfelelően fontos a modellek dokumentálása. Ehhez szükségszerűen a modelleket a gondolati síkból egy mindenki számara megfogható formába kell hozni (következik mindez abból, hogy a gondolatolvasás ma még nem megvalósítható alternatívája az információ átvitelnek). Ennek megfelelően szükség van olyan modellező nyelvre, amely lehetővé teszi a modellek leírását olyan formában, amely mindenki számára érthető. Egy ilyen, szabványos modellező nyelv az UML (Unified Modeling Language), amelyet ma már minden magára valamit is adó szoftvercég a modelljeinek dokumentálására használ.

Megjegyzés

Az UML nem fejlesztési folyamat, ennek megfelelően nem mondja meg, hogy hogyan kell a szoftvert kifejleszteni. Az UML modell leíró nyelv, egy jelölésrendszer, amely modell elemeket definiál és meghatározza az egyes modell elemekhez szemantikai tartalmát és szintaktikáját.

Az UML, mint modell leíró nyelv szerepe

- Lehetővé teszi a modellek dokumentálását
  - később is emlékezni fogunk
  - segíti a csapat tagjai közötti kommunikációt
- Vizuális, grafikus nyelv, ezáltal nagymértékben segíti a probléma megértését
  - szemléletes
  - kifejező
- Szabvány
  - lehetővé teszi a cégek közötti kommunikációt
- Kellően flexibilis és könnyen kiterjeszthető ahhoz, hogy valamennyi szoftverfejlesztéssel foglalkozó cég illeszteni tudja a maga fejlesztési folyamatához
- Kellően formális ahhoz, hogy
  - A fejlesztők számára megfelelő pontossággal bírjon
  - A modellező eszközök számára is kellőképpen megfogható és kezelhető

A modellezéssel kapcsolatos problémákat a következőképpen foglalhatjuk össze:

1. Megoldást kell találni a bonyolultság kezelésére, vagyis hogy az elkészült modellek átláthatóak, kezelhetők maradjanak

2. Biztosítanunk kell a modellnek leírásának egyértelműségét, az ellentmondások ellenőrizhetőségét.

Verziókezelő rendszerek

Verziókezelés alatt több verzióval rendelkező adatok kezelését értjük. Leggyakrabban a mérnöki tudományokban és a szoftverfejlesztésben használnak verziókezelő rendszereket fejlesztés alatt álló dokumentumok, tervek, forráskódok és egyéb olyan adatok verzióinak kezelésére, amelyeken több ember dolgozik egyidejűleg. Az egyes változtatásokat verziószámokkal vagy verzióbetűkkel követik nyomon.

Felhasználási területek

Elsősorban többszemélyes projektek esetében.

A legtöbb verziókezelő rendszert szoftverfejlesztési projektekben használták először, de egyes szövegszerkesztők, táblázatkezelők és egyes tartalomkezelő szoftverek is támogatják.

A beépített verziókezelés a wiki szoftvereknél is kulcsfontosságú. A wiki rendszerek integrált verziókezelői teszik lehetővé, hogy a felhasználók nyomon követhessék egymás szerkesztéseit, és visszaállíthassanak oldalakat azok korábbi verzióira, ezzel védekezve a vandalizmus és a spam ellen.

Alapjellemzők

Definíció: a szoftver megépítéséhez szükséges források és más fájlok tárolása és megosztása

- Biztonságosan
- Ellenőrzött hozzáféréssel
- A fájlok legújabb illetve korábbi változatainak megőrzésével

Alapjellemzők: biztonság

- Megbízható hardver (pl. RAID) – diszkhibák ellen
- Rendszeres mentés, archiválás (Disaster recovery)
- A fejlesztés történetének teljes rögzítése
- Verziók azonosítása
- Fejlesztői/kooperációs hibák ellen
- Új verziók és javítások párhuzamos fejlesztéséhez

Alapjellemzők: ellenőrzött hozzáférés

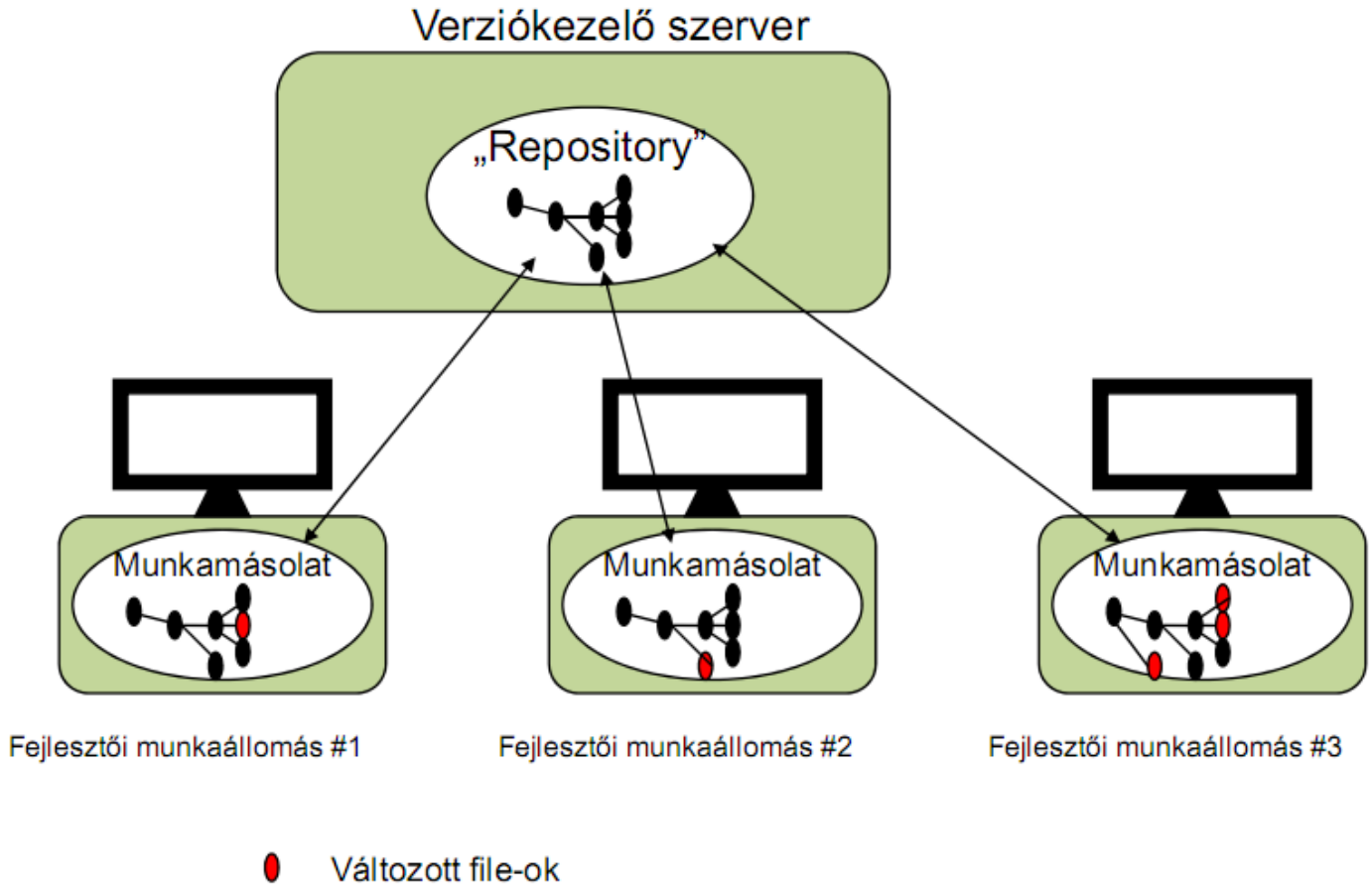
- Felhasználó-azonosítás
- A változtatások névhez köthetők legyenek
- A vétett hibák névhez köthetők legyenek
- Jogosultságok bevezetése, védelem
- Konkurencia-kezelés
- Fájl szinten (párhuzamos módosítások)
- Alkalmazás logikai szintjén (konzisztens állapotok, mérföldkövek megkülönböztetése)

Kezelési modellek

Központosított modell (hagyományos): minden verziókezelési művelet egy közösen használt szerveren történik .

Elosztott verziókezelő rendszerek: minden felhasználó gépe egy-egy külön tárolóként jelenik meg.

Példa: Központi verziókezelő



Alapműveletek

- Lokális munkamásolat készítése (checkout)
- Lokális másolat frissítése (update)
- A lokális példány is már változhatott addigra!
- Változások megtekintése, elemzése (log, diff, status)
- Változások visszaírása a repositoryba (commit, checkin) - ez az egyetlen írási művelet a repository felé!

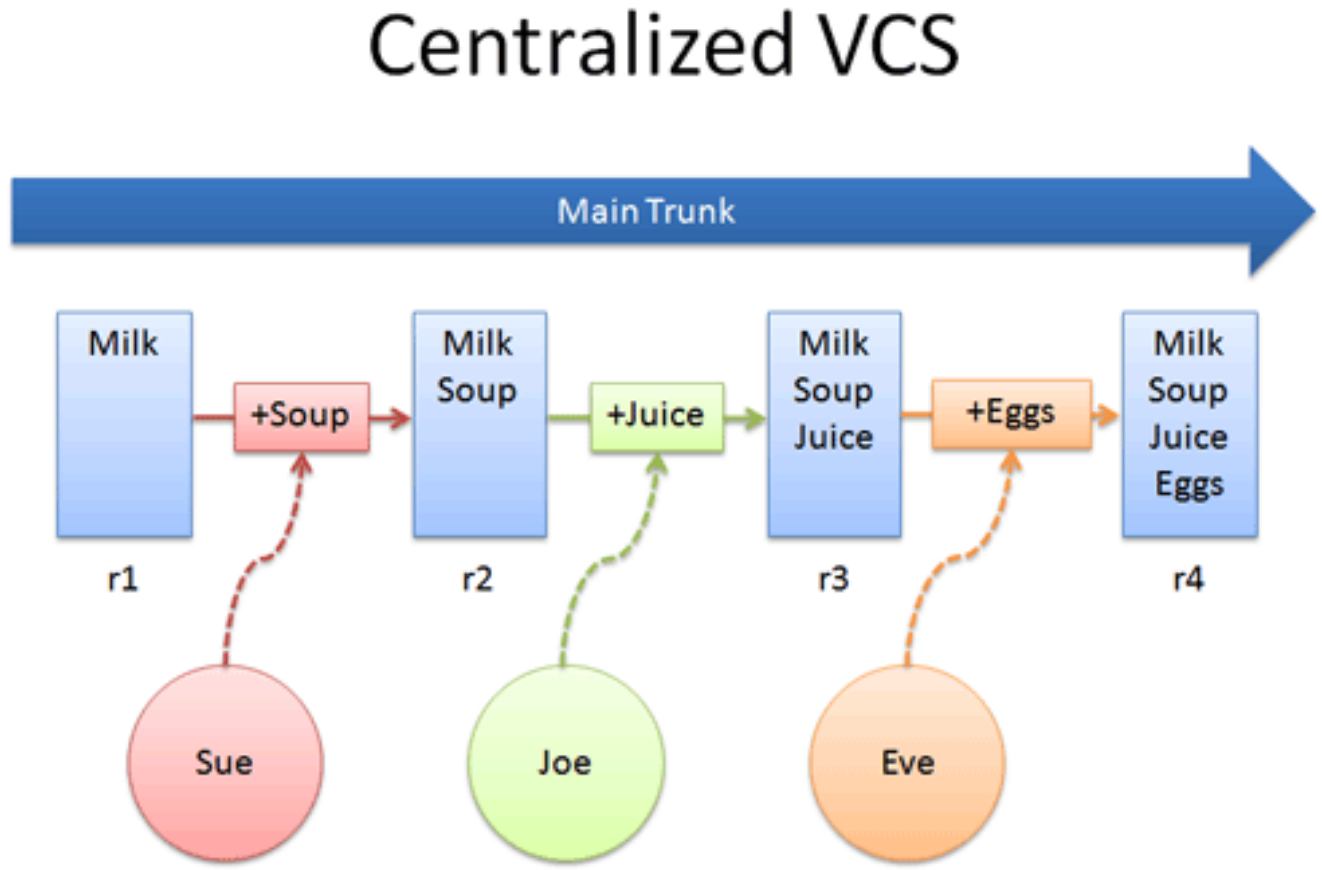
Központosított modellek

Probléma: ha két, vagy több fejlesztő egyidejűleg próbálja módosítani ugyanazt a fájlt. Az ilyen (centralizált) rendszerek kétféleképpen oldják meg ezt a problémát (concurrency modell): lock-olással és merge-eléssel.

Lock modell (zárolás): egyszerre csak egyvalaki dolgozhat a fájlon, a többiek addig ki vannak zárva. Ennek a módszernek előnyei és hátrányai is vannak. A nagyobb vagy sok fájlt érintő változtatásoknál célszerű ezt választani, mert bonyolult összefésülési műveleteket lehet megtakarítani vele. Ha azonban egy fájl túl sokáig zárolt állapotban marad, akkor a többi fejlesztő esetleg arra vetemedhet, hogy a verziókezelést megkerülve a fájl lokális másolatát módosítsák, ami nagyobb bonyodalmakhoz vezethet.

Merge modell (összefésülés): a fájlt egyidejűleg többen is használhatják, a fájl a módosítások összefésülése révén alakul, fejlődik. Ekkor a saját változtatását elsőként elhelyező felhasználó mindenképpen sikerrel fog járni. A rendszer a többi felhasználónak összefésülési lehetőséget ad, mellyel a különböző módosítások összeolvaszthatóak, így a felhasználók nem írják felül egymás munkáját. Az összefésülés lehet automatikus vagy kézi. Általában az összefésülésre képes verziókezelők is adnak lehetőséget fájlok egy felhasználós, kizárólagos szerkesztésére reserved edit néven.

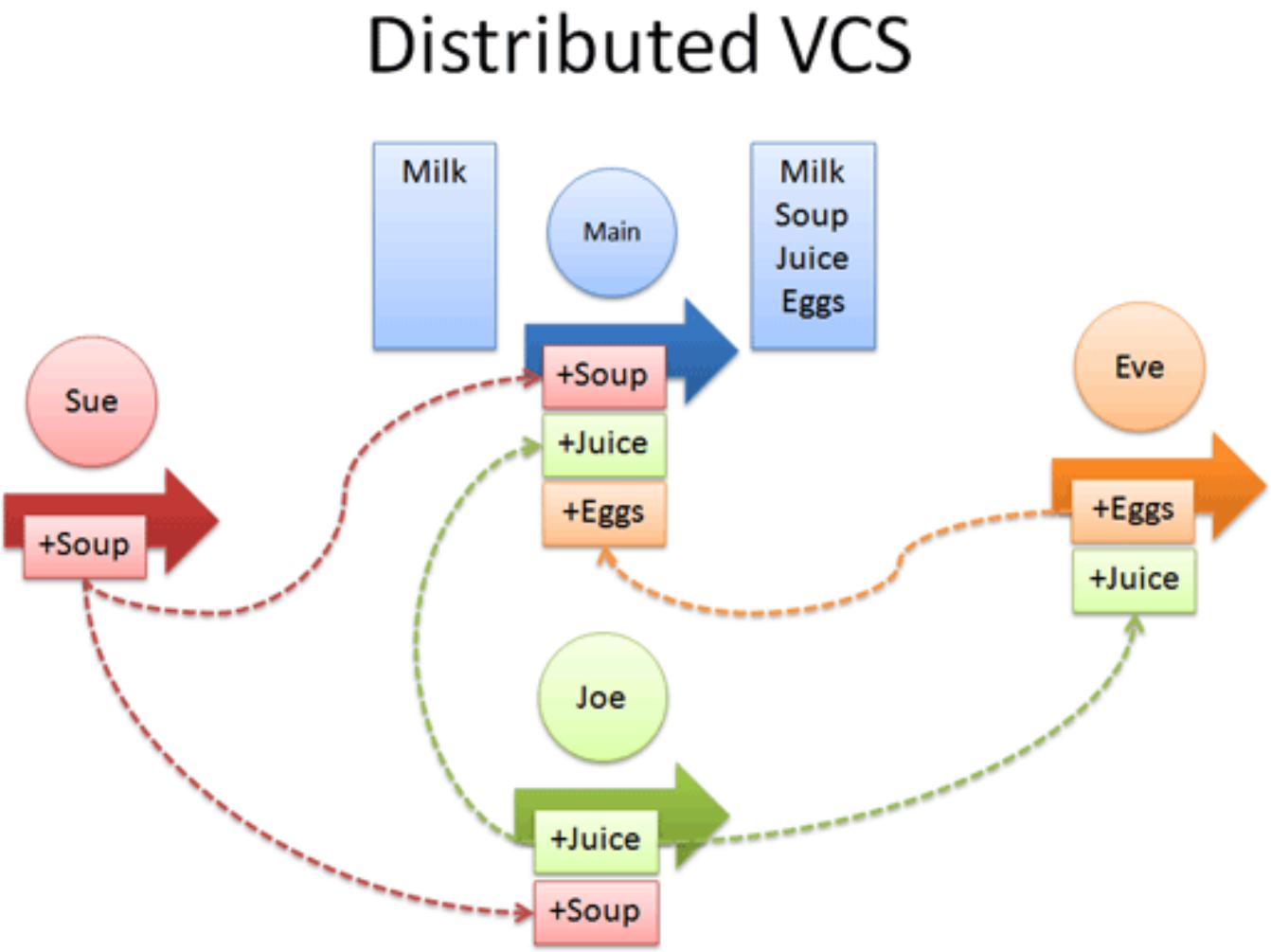
-



Elosztott rendszerek

- Nincs nagy központi adatbázis, csak munkamásolatok (working copies).
- A gyakori műveletek gyorsak, mert nem kell központi szerverrel kommunikálni.
- Minden munkamásolat egy-egy távoli backup, ami természetes védelmet ad az adatvesztés ellen.

-



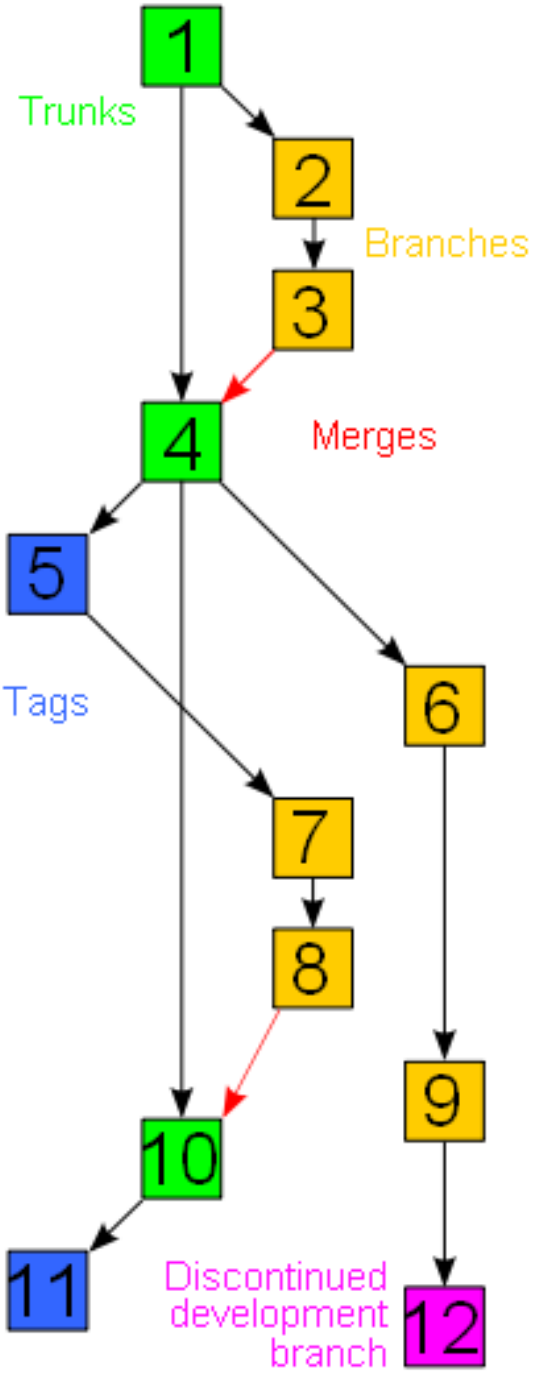
Központi vs. elosztott konklúzió

A központi: egyszerű, kiváló backup, undo, szinkronizáló műveletekre.

Elosztott: a valós élethez közelebb áll, összefésülés, elágazásra alkalmasabb, nem igényel folyamatos online jelenlétet, csak a változások megosztásánál. Sokkal gyorsabb nagy projekteknél, sok változtatás esetében, a teljes verziótörténet offline is elérhető minden egyes kliensen, cserébe nehezebb menedzselni központilag.

Nyitott rendszer (open system)

- Minden munkamásolat gyakorlatilag egy ág. (branch)
- Minden ág egy-egy munkamásolatként implementálódik. Az ágak összefésülés patch-ek küldözgetésével történik.
- Lehet válogatni az egyes változtatások között, nem kell feltétlenül minden változtatást letölteni.
- Új tagok bármikor csatlakozhatnak a rendszerhez, nincs szükség szerveroldali regisztrációra.



Trunk: a fejlesztés fő vonala (baseline, mainline), jóváhagyott változata

Branch: elágazás, párhuzamosan fejlesztett változat

Tag: a fájlokhoz adott időpillanatban, adott ponton rendelt címke (label), ami a verziószámot jelöli (beszédesen, vagy számokkal).

Merge (összefésülés) szükségessége

- Ha egy felhasználó módosítja a saját munkamásolatát, majd letölt a szerverről egy másik módosított változatot. Ekkor a szerveren lévő változásokat össze kell fésülni a lokális munkapéldány változásaival a kliensen.
- Ha a fejlesztésben elágazás történt, majd egy hibát kijavítottak valamely ágban, s a javítást alkalmazni kell a másik ágra is.
- Ha a fejlesztésben elágazás történt, majd az ágakat különböző irányba fejlesztettek tovább, s a különböző fejlesztéseket össze kell vonni egy közös változatba (trunk-ba).

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)

Korszerű verziókövető rendszerek:

- Git (ingyenes, ma a legnépszerűbb rendszer)
- Subversion (ingyenes)
- Microsoft TFS
- IBM Rational ClearCase
- Mercurial (ingyenes)

Verziókezelő rendszerek általános fogalmai – ezek a minden programozó által a verziókezelő rendszerben végzendő munkafolyamatok, melyeket elolvasva a betartandó szabályok is világossá válnak. Az, hogy ez a gyakorlatban hogy fest, a verziókezelő rendszer típusától függ: központosított / elosztott / nyitott modell (lásd feljebb).

A terminológia rendszerenként változik, de vannak általánosan használt szakkifejezések.

Baseline: Egy dokumentum vagy fájl jóváhagyott verziója, melyhez az azt követő változtatásokat viszonyítják.

Branch (ág): A verziókezelt fájlok egy részhalmaza elágazhat, így azoknak több aktuális változatuk lesz egyidejűleg, melyeket akár különböző sebességgel és különböző irányokba is fejleszthetnek.

Check-out: Lokális másolat készítése valamely verziókezelt fájlról. Alapértelmezésben ilyenkor a legfrissebb verziót kapja a felhasználó, de általában van lehetőség konkrét verzió kikérésére is verziószám alapján.

Check-in vagy Commit: Az a művelet, amikor a lokális példány változtatásai beíródnak (vagy egyszerű másolás vagy összefésülés eredményeként) a szerveren tárolt változatba.

Conflict: Konfliktusról akkor beszélünk, ha ketten akarnak megváltoztatni egy dokumentumot vagy fájlt és a rendszer nem képes összeépíteni a változásokat. A felhasználónak ekkor fel kell oldania a konfliktust, amit

vagy úgy tehet meg, hogy a változtatásokat összekombinálja vagy úgy, hogy kiválasztja az egyik változtatást és csak azt juttatja érvényre.

Change: Egy változtatás (change, diff vagy delta) mindig egy verziókezelt dokumentumon vagy fájlon tett változtatást jelenti. Rendszerfüggő, hogy milyen mértékű módosítások számítanak change-nek.

Change list: Egy change list vagy change set egy check-in művelet során bevitt változtatások listája, olyan rendszereken, melyek támogatják atomi műveletként több változás egyidejű becsekkelését.

Dynamic stream: Egy olyan adatszerkezet, amely egy adott tárolón lévő elemek konfigurációját reprezentálja, és időben változik.

Export: Az export a checkout-hoz hasonlít azzal a különbséggel, hogy tiszta könyvtárat csinál a verziókezeléshez szükséges meta adatok nélkül. Ezt a műveletet általában közvetlenül a tartalom publikálása előtt szokták használni.

Head: A legutóbbi checkin.

Import: Az import művelettel lehet egy lokálisan tárolt adathalmazt, amely még nem munkamásolat, felmásolni a tárolóra és verziókontroll alá helyezni.

Mainline: Hasonlít a trunk-hoz, de minden ágnak lehet saját mainline-ja.

Merge: A merge művelettel két változtatáslistát lehet összefésülni, s ezáltal egy közös verziót létrehozni. Erre a következő esetekben lehet szükség:

- Ha egy felhasználó módosítja a saját munkamásolatát, majd letölt a szerverről egy másik módosított változatot. Ekkor a szerveren lévő változásokat össze kell fésülni a lokális munkapéldány változásaival a kliensen.
- Ha a fejlesztésben elágazás történt, majd egy hibát kijavítottak valamely ágban, s a javítást alkalmazni kell a másik ágra is.
- Ha a fejlesztésben elágazás történt, majd az ágakat különböző irányba fejlesztettek tovább, s a különböző fejlesztéseket össze kell vonni egy közös változatba (trunk-ba).

Repository: A repository, depot vagy tároló az a hely (tipikusan egy szerver), ahol az aktuális és a korábbi verziók tárolódnak.

Reverse integration: Az egyes ágak összedolgozása és bedolgozása a verziókezelő fő trunk-jába.

Revision: A revision szó ugyanazt jelenti, mint a version. Egy verzió.

Tag: A tag, label vagy címke egy fontos időpillanatot jelöl. Egy adott fájlcsoporthoz hozzárendelhető egy címke, amely beszédes, felhasználóbarát nevet vagy verziószámot adhat a csoportnak.

Trunk: A fejlesztés egyik olyan vonala, amely nem branch.

Resolve: Változási konfliktusok feloldására irányuló felhasználói tevékenység.

Update: Az update vagy sync a repository-ban lévő változtatásokat dolgozza bele a felhasználó munkamásolatába.

Working copy: Magyarul munkamásolat. A repository fájljainak másolata a felhasználó lokális gépén. Minden olyan munka, ami bekerül a repository-ba, először mindig egy munkamásolatban történik meg, innen a neve. Fogalmilag a munkamásolat egy homokozó.

Atomi szintű commit: A részben végrehajtott commit-ok korruptálhatják az adatbázist, ezért bevezették az atomi szintű commit fogalmát, amely azt jelenti, hogy egy commit által küldött változtatások csak akkor érvényesülnek, ha minden rendben zajlott az átvitel folyamán. Nem minden rendszer ismeri, de az újabbakra már jellemző a technika megléte.

Cherry-Picking: Algoritmus mely teljesen különböző commitok (patchek vagy changesetek) esetén kiválaszt egyet, és azt alkalmazza a branchre.

Changeset: Fájlok különböző verzióinak tárolása helyett egyes rendszerek ún. changeset –t használnak. A changeset csak a változásokat tárolja le két tree között, ezáltal elő lehet állítani egy verzióból a rákövetkező verziót.

Snapshot: Fájlok és könyvtárak egy csoportja, amely a jelenlegi verziót megelőző állapotot írja le.

A szoftverek életciklusa

A programkészítés lépései:
1.) A feladat megfogalmazása
2.) A feladat modellezése
3.) Az algoritmus meghatározása
4.) A szükséges hardver és szoftver környezet megállapítása
5.) Az algoritmus kódolása
6.) A program tesztelése, hibakeresése, javítása
7.) A hatékonyság növelése, optimalizálás
8.) A dokumentációk elkészítése
9.) A működő program karbantartása, felügyelete

Programfrissítés készítése

Ehhez a részhez a Git-et, mint a legnépszerűbb verziókezelő rendszert vesszük alapul.

Mielőtt programfrissítést készítenénk, a Git-ben egy külön branch-et célszerű létrehozni neki, így az addig elkészült részek érintetlenek maradnak és később az összefésüléskor eldönthetjük mit teszünk bele a végleges verzióba, és mit nem. Ehhez természetesen az addigi repository másolatát kell használni.

Ez után a frissítésbe tervezett funkciók megvalósítását kell elvégezni, közben rendszeres commit-ekkel pedig rögzíteni az ágon elvégzett módosításokat a repository-ba. Mivel külön branch-ben dolgozunk, a módosításaink együtt lekérhetőek és a Git képes őket egy fájlban összefoglalni. A frissítés így minden elvégzett változtatást és kiegészítést magába foglal.

A frissítés tesztelését a Git képes úgy elvégezni, hogy nem kell a merge-t elvégezni, ezzel a teljes szoftverünket módosítani. Ha ez rendben megy, akkor már végrehajtható a patch és ezzel a fő ág frissítése. Az így létrejövő programot már tesztelhetjük és ha beválik, kiadható a frissítés a Git-ben létrehozott águnk commit-jei alapján.