UNIVERSITATEA POLITEHNICA BUCURESTI

FACULTATEA ELECTRONICA, TELECOMUNICATII SI TEHNOLOGIA INFORMATIEI

VERSIONAREA

Cadru didactic:

Conf. dr. ing. Stefan Stancescu

***Studenti*:**

Dima Adrian Claudiu - 443A

Serban Stefan - 443A

INTRODUCERE

Din punct de vedere al evolutei produsului software, versionarea are o stransa legatura cu ciclul de dezvoltare a acestuia. Astfel, aceasta lucrare trateaza deopotriva evolutia ciclului de dezvoltare al unui produs software, dar si versionarea acestuia.

Prima parte a acestei lucrari descrie momentele cheie in dezvoltarea aplicatiei software prin prisma celor 3 perioade foarte importante:

1. Testare si dezvoltare
2. Produsul final
3. Mentenanta

In tot acest timp, pot exista infinite versiuni intermediare si linii de dezvoltare suplimentara care se vor gestiona cu ajutorul unei metode de versionare. Acestea sunt reprezentate cu ajutorul a 3 modele:

1. Modelul bazat pe fisiere locale
2. Modelul client-server
3. Modelul bazat pe abordarea distribuita

CICLUL DE VIATA AL UNEI VERSIUNI SOFTWARE

1. Dezvoltare & Testare

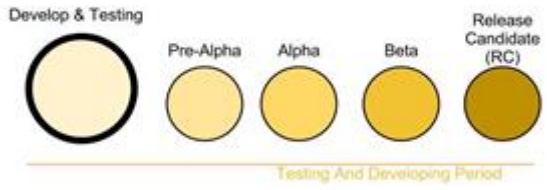

Figura 1 – Reprezentarea perioadei de dezvoltare si testare

***Pre-alpha*:** se refera la toate activitatiile desfasurate in timpul proiectului , activitati ce sunt prioritare pentru testare. Aceste activitati pot include: software design , software development si unit testing.

Versiunile *Milestone* (1.0, 2.0, etc.) includ seturi specifice de functii si sunt eliberate de indata ce functionalitatea lor este completa.

Alpha*:** este prima faza de incepere a testarii produsului software. In aceasta faza dezvoltatorii testeaza software-ul, utilizand anumite tehnici. Aceasta faza este de asemenea cunoscuta ca "*Private Beta*". Sunt realizate testari aditionale de catre alte echipe (sunt echipe de QA care testeaza aplicatia respectiva), iar acestea dau feedback echipei de dezvoltatori cu respectivele probleme aparute in urma testarii (bugs***).

Lansare acestei aplicatii in interiorul firmei este cunoscuta ca "*Alpha release"*. Aceasta versiune poate fi instabila si de asemenea poate cauza "crash-uri" sau "data loss". Faza alpha se termina de regula cu o inghetare de caracteristici (s-au realizat toate functionalitatile programului).

Beta*:** este faza de dezvoltare a produsului software urmata dupa alpha . Aceasta faza incepe in momentul in care caracteristicile aplicatiei sunt realizate. Principalul obiectiv pentru faza de testare ***beta este acela de a reduce impactul aplicatiie asupra utilizatorilor.

Utilizatorii versiunii beta sunt numiti "beta testers" si sunt de regula clienti sau posibili viitori client care doresc sa testeze gratuit aplicatia software.

Versiunea beta este utila pentru demonstratii interne (prezentarea aplicatiei in interiorul unei companii mare de dezvoltare) si de asemenea ca "*preview*" sau "*prototype*" pentru a selecta o gama de clienti.

- Figura 1: http://imgur.com/225UMpZ

Open/Closed beta: developerii elibereaza ori o versiune "*closed beta*" ori "*open beta*"(cunoscut si sub numele de CTP). "*Closed beta*" este o versiune lansata pentru un grup restrans de testare. "O*pen beta*" se adreseaza unui grup mare de testare, in general acest grup fiind publicul. Utilizatorii si/sau testerii anunta developerii de orice *bug* gasit in aplicatie si de asemenea pot sugera anumite viitoare functionalitati pentru aplicatie .

Un exemplu de "*open beta*" test a fost "*Community Technology Previews*" - Microsoft.

***Release candidate*:** termenul de "*Release Candidate*" (RC) se refera la o versiune ce poate fi un produs final , ce este gata de lansare numai daca nu apare o eroare majora. Bugurile unei aplicatii sunt clasificate in: minor, majore, fatale.

In general un bug major se refera la o functinoalitate importanta precum transferul bancar in timp ce un bug minor se refera la o mica eroare grafica de design ce dureaza putin reabilitarea ei si evident timpul de testare este mult mai mic comparativ cu testul unui "*major bug*".

In aceasta faza , toate functinalitatile aplicatiei , tot ce tine de design a fost codat si testat in mai multe versiuni beta sau numite dupa alte litere grecesti precum gamma, delta sau versiuni aproape complete (aproape de lansare dar care sunt inca sub testare) numite si omega sau zenith.

O versiune "*Release*" se numeste "*code complete*" atunci cand echipa de dezvoltatori stabileste ca nu se va mai adauga un nou cod sursa aplicatiei. Exista insa o posibilitate sa se adauge coduri sursa mici pentru realizarea erorilor. Un cod nou se poate adauga intr-un nou "*release*".

2. Lansarea

Figura 2 – Reprezentarea perioadei de lansare

***RTM - Release to Marketing*:**

("*release to manufacturing*" sau "*release to marketing*"- cunoscute si ca "*going gold*")

Termenul RTM este folosit pentru a indica faptul ca aplicatia a cunoscut un nivel inalt de calitate si este gata pentru distribuirea in masa, fie prin mijloace electronice, fie prin mijloace mass-media.

- Figura 2: http://imgur.com/225UMpZ

Termenul nu defineste modalitatea de distribuire, ci spune doar ca nivelul calitatii este suficient de ridicat pentru distribuirea in masa. (exemplu: s-au realizat CD-ul de instalare si documentatia aferenta instalarii) .

RTM se intampla inainte de GA (General Availability).

***General Availability (GA)*:** este punctul in care toate activitatile necesare de comercializare au fost realizate si produsul software este pus la dispozitia publicului prin intermediul WEB sau suporturi fizice – CD , DVD , Stick , Hard Disk – pt un sistem de operare.

Activitatile comerciale pot include, insa nu sunt obligatorii, disponibilitatea aplicatiei la nivel mondial (centre in anumite tari) ceea ce implica faptul ca aplicatie trebuie sa fie implementata in cat mai multe limbi (acest lucru implica un timp in plus de testare si de asemenea costuri suplimentare).

Timpul dintre RTM si GA poate varia de la o saptamana pana la cateva luni in anumite cazuri, deoarece este nevoie de timp pentru a realiza toate activitatile cerute de GA.

Un alt termen cu un inteles aproape identic cu GA este FCS (First Customer Shipment). Unele companii precum Microsystems si Cisco folosesc FCS pentru a descrie versiuni de soft ce au fost lansate pentru beneficii de venituri. Este de asemenea stagiul in care aplicatia este considerata live ("*gone live*").

"*Live version*" este versiunea finala pentru un anumit produs software. O versiune live este considerata a fi foarte stabila si cu buguri foarte putine si mici si cu o calitate foarte mare pentru a putea fi distribuita global.

3. Mentenanta

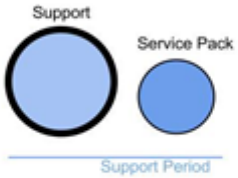

Figura 3 – Reprezentarea perioadei de mentenanta

***Service Pack*:** in timpul "vietii" unei aplicatii software, aceasta este supusa la anumite update-uri pentru imbunatatire si lansare a unor noi servicii sau rezolvari de buguri.

Spre exemplu, Microsoft Windows XP are 3 *Service Pack*-uri majore. Acestea sunt aduse utilizatorilor ca fisiere executabile ce se instaleaza pe sistem foarte usor.

***End of life*:** atunci cand aplicatia nu mai este vanduta si/sau sustinuta din punct de vedere al mentenantei ("*supported*"). In acest caz suportul pentru rezolvarea bugurilor sau lansarea unor noi versiuni s-a terminat.

- Figura 3: http://imgur.com/225UMpZ

Modul de denumire a versiunilor unei aplicatii

PRODUCT NAME (LIFECYCLE STAGE – LIFECYCLE STAGE NUMBER) MAJOR.MINOR.REVISION

PRODUCT (BETA-2) 1.2.2345

LIFECYCLE STAGE

Valorile posibile sunt: alpha, beta, RC, RTM, GA, SP.

LIFECYCLE STAGE NUMBER

Se reprezinta cu un singur digit pornind de la 1.

Valoarea minima este egala cu 1 iar maxima cu 5.

Chiar daca dupa a 5-a iteratie produsul nu a fost in stare sa atinga nivelul de calitate dorit, LIFECYCLE STAGE NUMBER nu ar trebui crescut ci tinut sub nivelul de 5.

MAJOR

Este reprezentat de doi digiti cu prefixul diferit de 0. Valorile posibile 1-9. Numarul versiunii majore este crescut in cazul in care cantitati importante de informatii sunt aduse in urma update-ului.

MINOR

Este reprezentat de doi digiti cu prefixul diferit de 0. Valorile posibile 1-9. Numarul versiunii minore este crescut in cazul in care cantitati mici de informatii sunt aduse in urma update-ului.

REVISION

Este reprezentat de patru digiti. Este modificat dupa fiecare build intern, build cu scopul de testare .

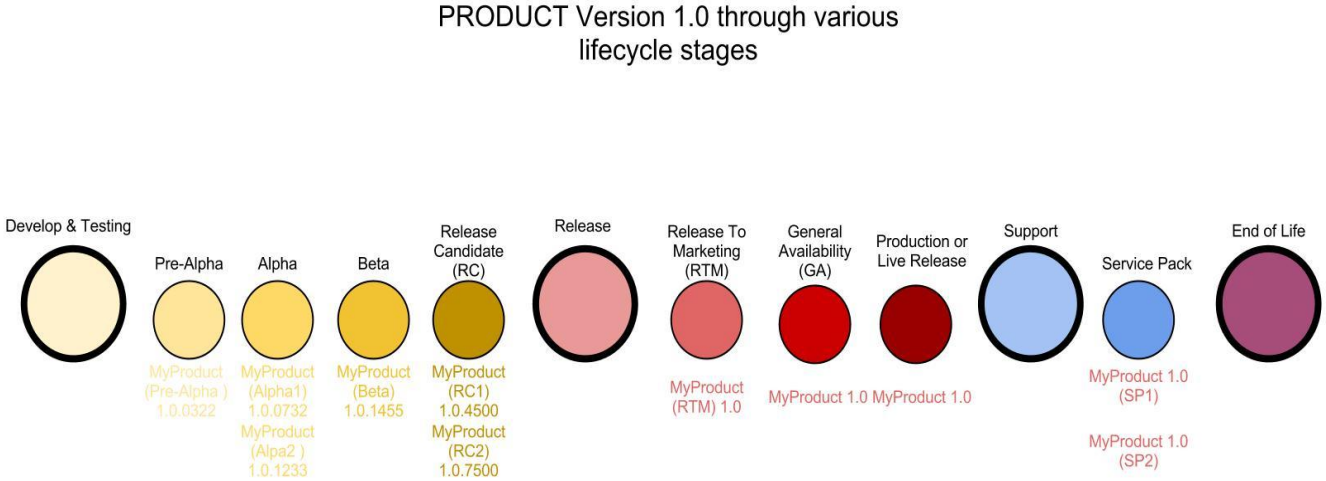

Figura 4 – Ciclul de viata al versiunii 1.0 a aplicatiei software

- Figura 4: http://imgur.com/225UMpZ

SISTEME DE VERSIONARE

Sistemele de versionare au un rol crucial in procesul de dezvoltare in timp a unei aplicatii. Acestea sunt folosite pentru a evidentia versiuni stabile, pentru a testa si a incerca repararea bug-urilor din versiuni instabile sau pentru a implementa functionalitati noi in cadrul aplicatiei. Pentru asta, ele folosesc o colectie din cele mai importante date, documentatie si o ierarhizare a fisierelor si directoarelor ce formeaza un "***repository***".

Cateodata, procesul de dezvoltare se imparte in ramuri. Astfel, o parte a echipei de dezvoltatori poate urmari implementarea unor noi functionalitati in timp ce alta echipa incearca rezolvarea bug-urilor curente ale aplicatiei.

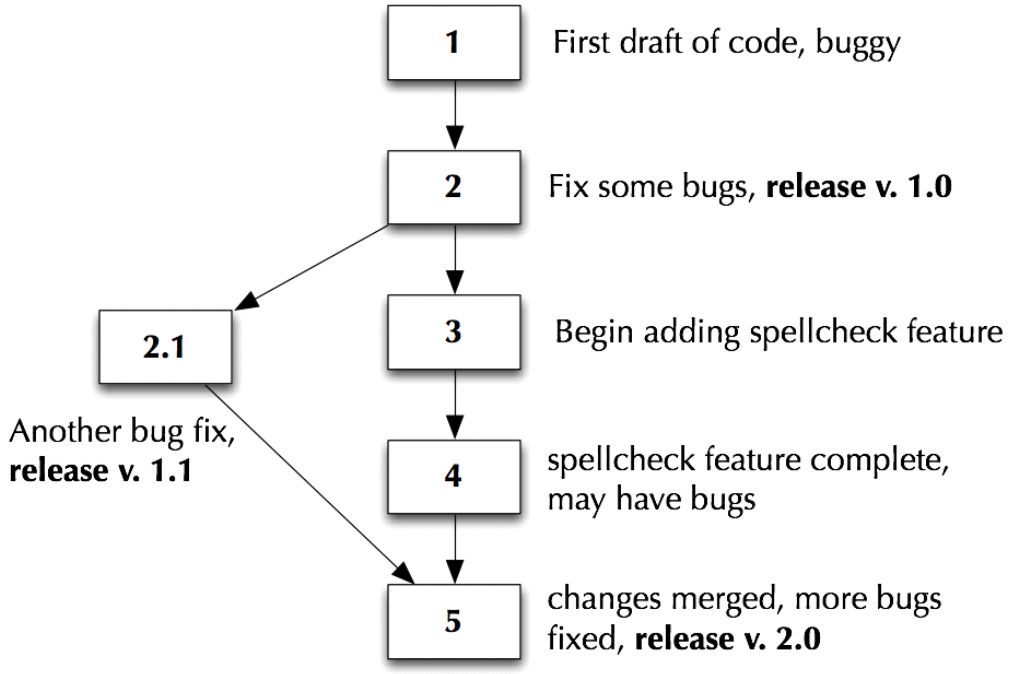

Figura 5 - Rezolvarea unui *bug*

Sistemului de versionare i se cere sa poata sustine toate acestea si sa ofere de asemenea si posibilitatea a reveni la o versiune anterioara, de a modifica, de a suprascrie sau de a recompila parti din codul folosit pentru aplicatie.

Dupa cum am mentionat, sistemele de versionare folosesc unul dintre aceste modele:

- modelul datelor locale
- modelul client-server (date centralizate)
- modelul datelor distribuite

- Figura 5: http://www.cs.colorado.edu/~kena/classes/5828/s07/lectures/06/tracking-changes-cm.png

1. MODELUL DATELOR LOCALE

Aceasta abordare locala necesita ca toti dezvoltatorii sa foloseasca acelasi sistem. Acest model analizeaza fisierele individual si are ca rezultat inlocuirea sau implementarea acestora in noul software.

RCS – Revision Control System

Este o aplicatie software care se foloseste doar pentru fisiere text individuale, nu pentru proiecte. Desi exista posibilitatea impartirii in ramuri, cei mai multi utilizatori ai aceste metode se folosesc doar de mecanismul de zavoare si lucreaza pe o singura ramura.

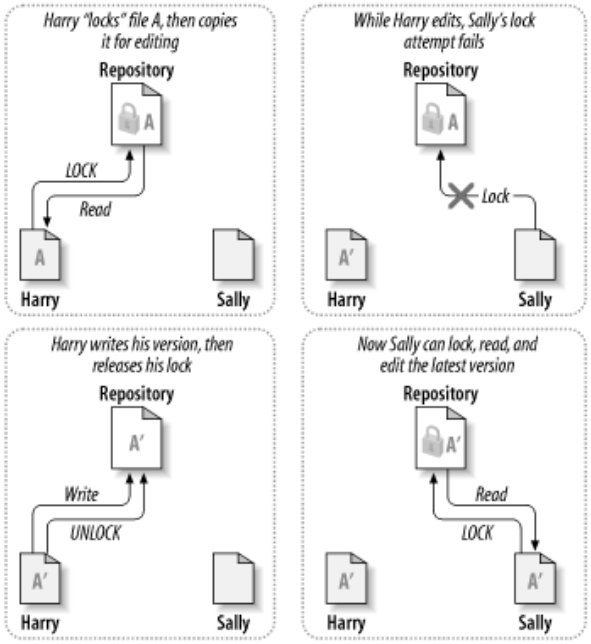

Figura 6 – Reprezentarea conditionarii accesului la fisier cu ajutorul zavorului

Acest sistem este folositor pentru fisiere care sunt versionate frecvent, cum ar fi programele, documentatiile, grafica procedurala etc. RCS poate fi folosit si pentru fisiere binare, desi eficienta in acest caz nu este foarte buna.

Versionarea se face cu ajutorul utilitarului *diff* care face diferenta dintre versiunea anterioara si versiunea curenta a fisierului in cauza.

In cazul script-urilor automate sau a fisierelor de configurare, RCS este preferat datorita simplitatii sale. In zilele noastre, unele motoare de cautare precum "*Twiki*" si "*Foswiki*" folosesc RCS pentru stocarea versiunilor paginilor

- Figura 6: http://svnbook.red-bean.com/en/1.7/svn.basic.version-control-basics.html#svn.basic.repository

2. MODELUL CLIENT-SERVER

Acest model presupune existenta unui repository central ("*central repository*") ce va fi folosit de toti dezvoltatorii proiectului.

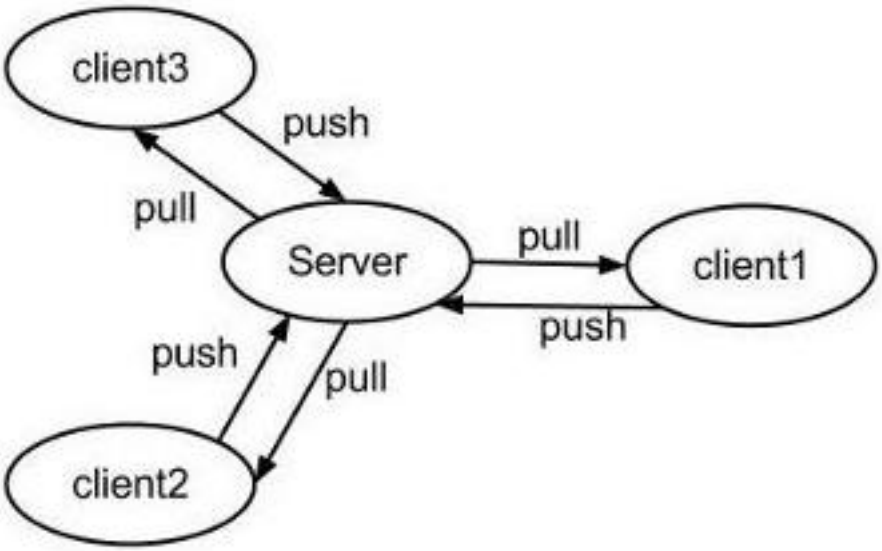

Figura 7 – Modelul Client-Server folosit in versionare

CVS – Concurrent Versions System

Este o aplicatie software care documenteaza schimbarile unui anumit set de fisiere ("*repository*") si permite dezvoltatorilor sa colaboreze intr-o maniera mai usoara.

Acest model functioneaza pe principiul prezentat in Figura 7 – un dezvoltator se conecteaza la server pentru a descarca versiunea centralizata a proiectului careia ii aduc imbunatatirile sau modificarile necesare. Versiunea este apoi incarcata inapoi pe server pentru ca toti ceilalti dezvoltatori sa beneficieze de modificari.

Astfel, dezvoltatorii sunt obligati sa isi descare mereu ultima versiune "centrala", lucru care se realizeza de obicei prin intermediului clientului de CVS.

Pentru a evita posibilitatea in care mai multe versiuni diferite sunt uploadate pe server, CVS accepta doar modificarile aplicate celei mai recente versiuni a fisierului.

In momentul incarcarii cu succes pe server a unei versiuni modificate, versiunile tuturor fisierelor implicate sunt incrementate iar serverul noteaza in fisierul de log ora, data si numele utilizatorului care a realizat upload-ul.

- *Figura 7:* http://help.cs.umn.edu/sites/all/files/images/rcs/rcs_centralized_and_distributed.jpg

Exemplu:

Pentru vesiunea 1.0 a unui produs se poate lua decizia de continuare a rezolvarii erorilor pe un front si de implementare a unor noi trasaturi pe celalalt front. Sa presupunem ca dupa un timp cele 2 subversiuni vor fi reunite pentru a forma versiunea 3.0 . Acest lucru este facilitat de catre sistemele VCS si totodata usureaza atat munca dezvoltatorilor de programe cat si munca celor care ii supervizeaza.

Cel mai usor mod de reprezentare a evolutiei unui produs software se face cu ajutorul arborilor.

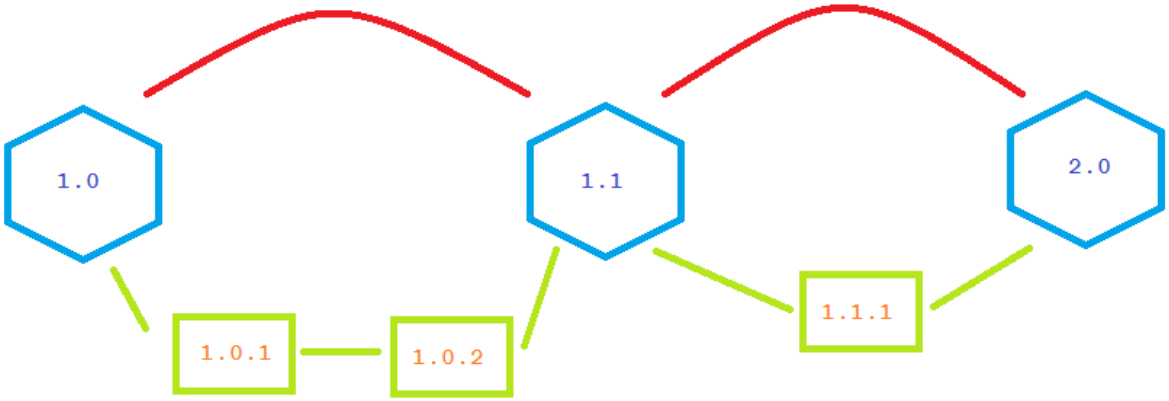

Figura 8 – Reprezentarea cu ajutorul arborilor a evolutiei unui produs software

Astfel, sunt reprezentate linii de dezvoltare ce pleaca de la un nod (*trunks*), apoi se impart in mai multe ramuri(*branches*) iar in final reuniunea intr-un alt nod (o noua versiune). Arborele are de regula o radacina (*head*) de la care pleaca. Un tag (*5,13*) reprezinta o salvare importanta in timp a progresului facut(“*snapshot*”). De regula, acestea pot reprezenta versiuni publicate.

Din cauza posibilitatii de reunire in diverse alte noduri, figura de mai jos nu reprezinta un arbore classic, ci un graf direct aciclic.

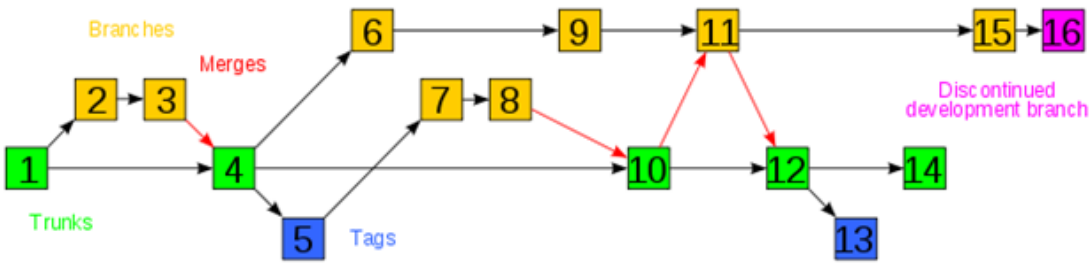

Figura 9 – Graf direct aciclic pentru reprezentarea versionarii

- Figura 9: http://en.wikipedia.org/wiki/Revision_control

3. *MODELUL DATELOR DISTRIBUITE*

Acest model presupune lucrul fiecarui dezvoltator cu propriul sau *repository.* Un alt pas a fost adaugat pentru corelarea acestora. Se recomanda folosirea lui pentru proiectele mari in care este nevoie de coordonarea multor dezvoltatori.

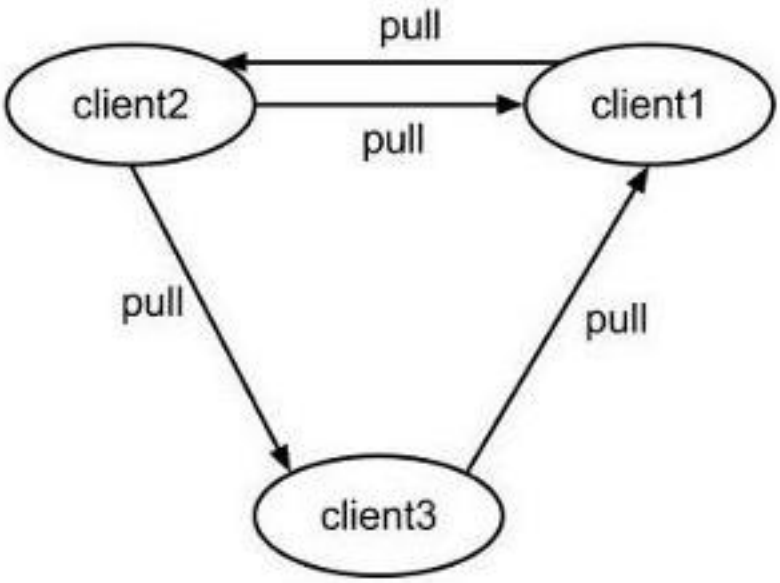

Figure 10 – Reprezentarea modelului datelor distribuite

Distributed revision control system – DRCS

Reprezinta tot o modalitate de versionare a produsului software, cu avantajul ca ofera dezvoltatorilor posibilitatea de lucru fara nevoia de a fi conectati la o retea comuna.

DRCS adopta o abordare peer-to-peer (client-client) care implica existenta a nenumarate repository-uri cu care clientii se sincronizeaza bazandu-se pe determinarea setului se schimbari aplicate repository-ului respectiv.

Unul din avanjajele acestui sistem este rapiditatea cu care se pot face operatii specifice de “commit”(aplicare schimbari) sau “ revert changes”(anularea ultimelor schimbari). Comunicarea dintre dezvoltatori este necesara doar in cazul schimbarii de informatii cu alt dezvoltator.

Fiecare dintre *repository*-urile salvate sunt efectiv folosite ca solutie de back-up, lucru ce duce la o protectie naturala impotriva pierderii datelor. Un alt avantaj este dat de faptul ca accesul la retea nu este necesar in majoritatea operatiilor.

- *Figura 10:* http://help.cs.umn.edu/sites/all/files/images/rcs/rcs_centralized_and_distributed.jpg

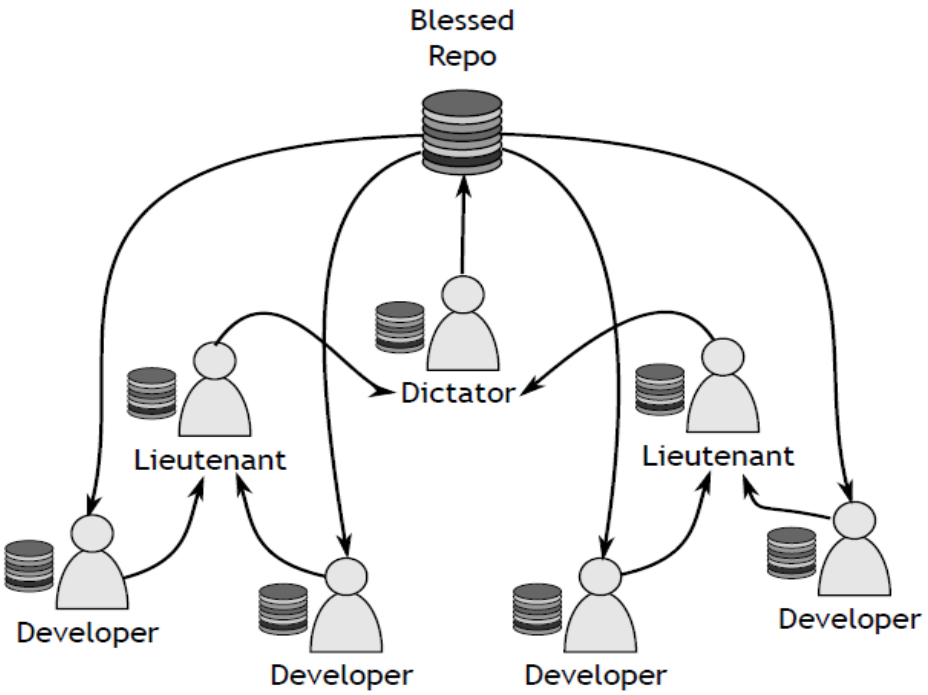

Figura 11 – Utilizarea unei versiuni distribuite pentru evolutia produsului software [model folosit de ***GIT***]

Este important de subliniat ca acest sistem ofera o infinitate de repository-uri ce pot fi "centrale" si ca sunt aplicabile diferite modele folositoare in proiectele foarte mari, precum: "development / release" si/sau "Commander / Lieutenant"

Un mare dezavantaj in cazul conexiunilor lente este dat de necesitatea copierii repository-ului intial de catre fiecare dezvoltator in parte.

Sistemele **DRCS** se impart in 2:

1. Sisteme deschide ("*Open systems*")

Acest sistem este caracterizat prin suportul oferit fiecarei ramuri independente si prin dependenta mare de operatia de imbinare – "*merge*". Fiecare ramura este implementata ca o copie funtionala – "*working copy*" – in care imbinarile se realizeaza prin schimbul normal din ramura in ramura a patch-urilor.

2. Sisteme replicate ("*Replicated systems*")

Asemanatoare cu modelul bazelor de date replicate. O uploadare de *repository* este echivalenta cu un *commit* distribuit. Cand acestea sunt realizate cu success, se creaza o noua singura noua ramura principala, motiv pentru care scade nevoia folosirii operatiei de imbinare. Un exemplu de software care foloseste acest model este "*Code Co-op*" realizat de Reliable Software.

BIBLIOGRAFIE

1. http://en.wikipedia.org/wiki/Comparison_of_revision_control_software
2. http://en.wikipedia.org/wiki/Distributed_revision_control
3. http://en.wikipedia.org/wiki/Concurrent_Versions_System
4. http://en.wikipedia.org/wiki/Revision_control
5. http://www.infoq.com/articles/dvcs-guide
6. http://svnbook.red-bean.com/en/1.7/svn.basic.version-control-basics.html#svn.basic.repository
7. http://en.wikipedia.org/wiki/Git_%28software%29
8. http://en.wikipedia.org/wiki/Software_versioning
9. http://www.avangate.com/community/resources/article/software-versioning.htm