Universitatea Politehnică Bucureşti

Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

Studenți implicați în redactare:

- ***Enache Marinel (442A)*** *coord.*
- ***Mălăescu Alexandru (441A)***
- ***Pană Radu (441A)***

..:: 2011 ::..

CUPRINS:

--Pană Radu--

1. Cicluri de viaţă ale software-ului

2.3 Ciclul în cascadă

2.4 Ciclul în V

2.5 Modelul incremental

2.6 Modelul spiralei

2.7 Modelul cu prototipuri

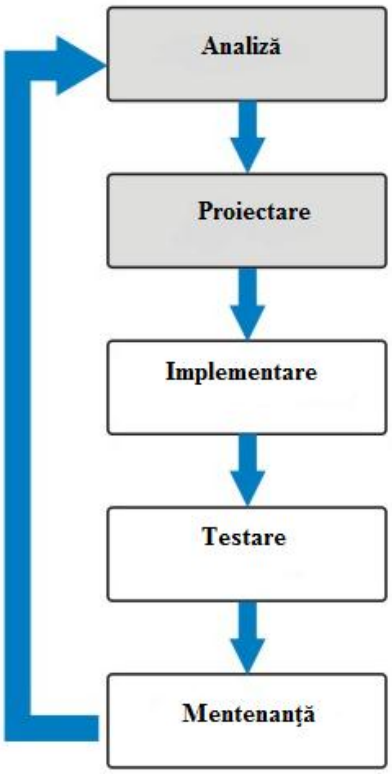
2.7.2 Prototipuri cu aruncare

2.7.3 Prototipuri evolutive

2.8 Modelul dezvoltării rapide a aplicaţiilor

1. *Cicluri de viaţă a software-ului*

Modelele de ciclu de viaţă a software-ului descriu fazele ciclului software şi ordinea în care aceste faze au loc. Există numeroase modele şi multe companii adoptă modele proprii dar toate sunt foarte similare. Modelul de bază este următorul:



Fiecare model produce rezultate folosite de faza următoare a ciclului de viaţă. Cerinţele sunt specificate în proiectare. Codul este produs în timpul implementării care este posibilă datorită proiectării. Testarea verifică rezultatele implementării, comparându-le cu cerinţele.

Cerinţele:

Această fază este importantă pentru managerii de proiect şi acţionari. Întâlniri între aceştia au loc pentru a determina cerinţele. Cine va folosi sistemul? Cum va fi folosit sistemul? Ce informaţii vor intra în sistem şi ce informaţii vor ieşi din acesta? Astfel de întrebări îşi găsesc răspunsul în timpul fazei de adunare a cerinţelor. Se produce o listă consistentă de funcţionalităţi pe care le oferă sistemul, care descriu funcţii pe care sistemul le poate îndeplini, se decide logica de prelucrare a datelor, ce date sunt stocate şi folosite şi cum va funcţiona interfaţa cu consumatorul. Rezultatul final este o descriere a funcţionalităţii sistemului şi nu se specifică modul în care sarcinile vor fi îndeplinite.

Proiectarea:

Proiectul sistemului software este produs din rezultatele fazei anterioare. Arhitecţii sunt cei care contribuie la îndeplinirea acestei faze. Aici sunt produse detaliile despre cum va acţiona sistemul. Arhitectura, software şi hardware, comunicaţia, proiectarea software (diagramele UML sunt produse) sunt toate ieşiri ale acestei faze.

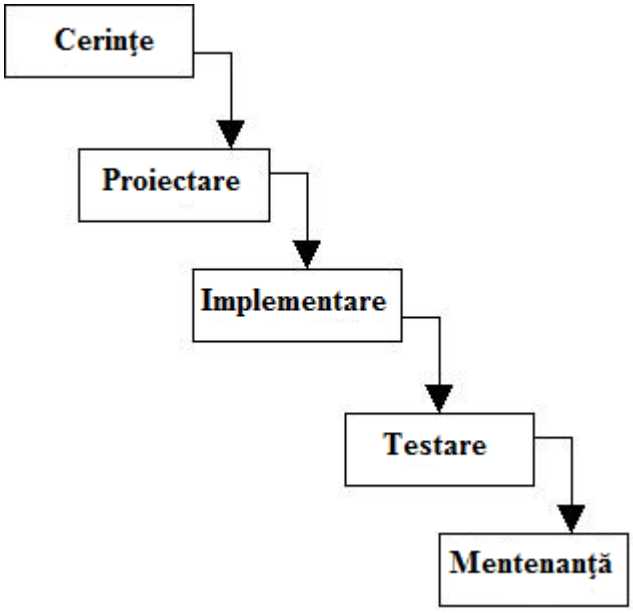
Implementarea:

Codul este produs din rezultatele fazei de proiectare. Aceasta este faza cea mai de durată. Implementarea poate să se suprapună atât cu faza de proiectare cât şi cu cea de testare. Multe tool-uri automatizează producerea codului folosind informaţia adunată şi produsă în timpul fazei de proiectare.

Testarea:

În timpul testării, implementarea este testată pentru a se verifica îndeplinirea cerinţelor, văzând astfel dacă produsul rezolvă într-adevăr nevoile adresate şi adunate în timpul fazei de preluare a cerinţelor. Se realizează teste unitare şi de acceptare. Cele unitare acţionează asupra unei singure componente a sistemului, pe când cele de acceptare se efectuează asupra întregului sistem.

1.1 Ciclul în cascadă:



Este cel mai comun model de ciclu de viaţă. Mai este numit modelul liniar-secvenţial. Este foarte simplu de înţeles şi folosit. Fiecare fază trebuie completată în întregime înainte ca următoarea fază să poată începe. La sfârşitul fiecărei etape are loc o revizuire pentru a se determina dacă procesul este pe calea cea bună şi pentru a se stabili dacă se continuă sau se renunţă la acesta. Fazele nu se suprapun la acest model.

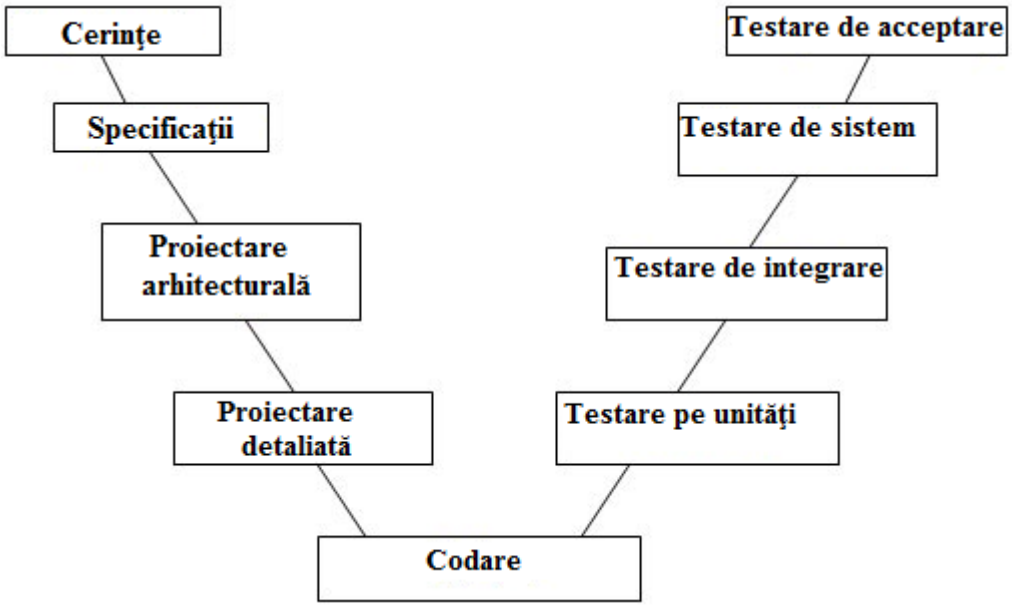
Avantaje:

- simplu şi uşor de folosit;
- uşor de gestionat datorită rigidităţii modelului: fiecare fază produce rezultate specifice şi are un proces de revizuire;
- etapele sunt procesate şi completate una câte una;
- funcţionează bine pentru proiecte mici unde cerinţele sunt foarte bine înţelese;

Dezavantaje:

- nu este produs software funcţional decât târziu în ciclul de viaţă;
- risc mare şi incertitudini numeroase;
- model slab pentru proiecte complexe şi obiect orientate;
- model slab pentru proiecte lungi;
- model slab acolo unde cerinţele se pot schimba;

1.2 Ciclul în V:



Asemănător cu modelul în cascadă, cel în V reprezintă o cale secvenţială de execuţie a proceselor. Fiecare fază trebuie îndeplinită înainte ca următoarea să poată începe. Testarea este mai accentuată decât în modelul cascadă. Procedurile de testare sunt dezvoltate devreme, înaintea codării, în timpul fiecărei faze anterioare implementării.

Cerinţele încep ciclul de viaţă, la fel ca la modelul în cascadă. Înainte de începerea dezvoltării, planul de test al sistemului este creat. Acesta se concentrează pe îndeplinirea funcţionalităţii specificate de cerinţe.

Faza de proiectare de nivel înalt se concentrează pe arhitectura sistemului. Un test de integrare este creat pentru a testa capacitatea pieselor sistemului de a funcţiona împreună.

Proiectarea de nivel scăzut este faza în care componentele software sunt proiectate şi sunt folosite teste unitare (individuale).

Faza de implementare este cea în care codarea are loc. Odată terminată, execuţia continuă cu partea dreaptă a V-ului unde testele create anterior sunt folosite.

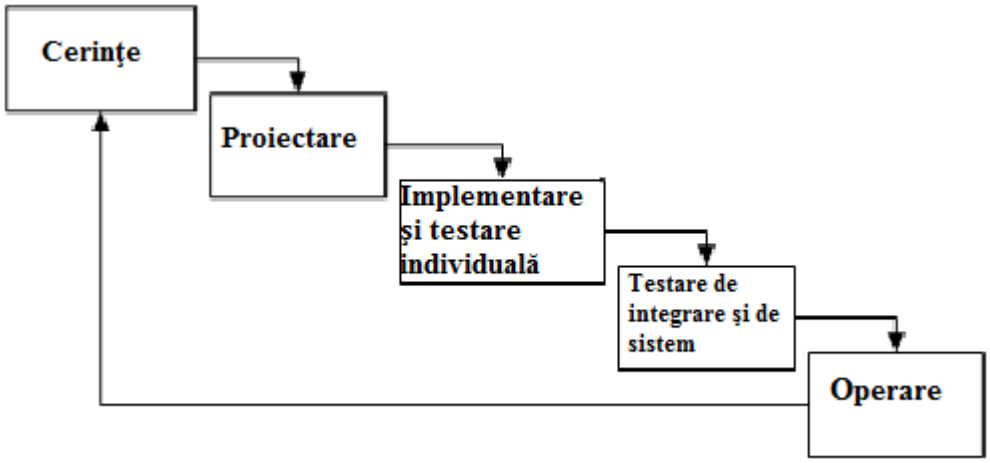
Avantaje:

- simplu şi uşor de folosit;
- fiecare fază are rezultate specifice;
- rată mai mare de succes faţă de modelul în cascadă datorită dezvoltării planurilor de test devreme în ciclul de viaţă;
- funcţionează bine pentru proiecte mici unde cerinţele sunt uşor de înţeles;

Dezavantaje:

- foarte rigid, asemănător modelului în cascadă;
- flexibilitate redusă, iar ajustările sunt dificile şi scumpe;
- nu sunt produse prototipuri ale produsului software;
- nu se specifică o cale de rezolvare a problemelor descoperite în fazele de testare;

1.3 *Modelul incremental:*



Modelul incremental este o abordare intuitivă a modelului în cascadă. Multiple cicluri de dezvoltare au loc, dând naştere unui ciclu „multi-cascadă”. Ciclurile sunt împărţite în iteraţii mai mici şi mai uşor de gestionat. Fiecare iteraţie trece prin cerinţe, proiectare, implementare şi testare.

O versiune funcţională a software-ului este produsă în prima interaţie. Următoarele iteraţii dezvoltă produsul software iniţial.

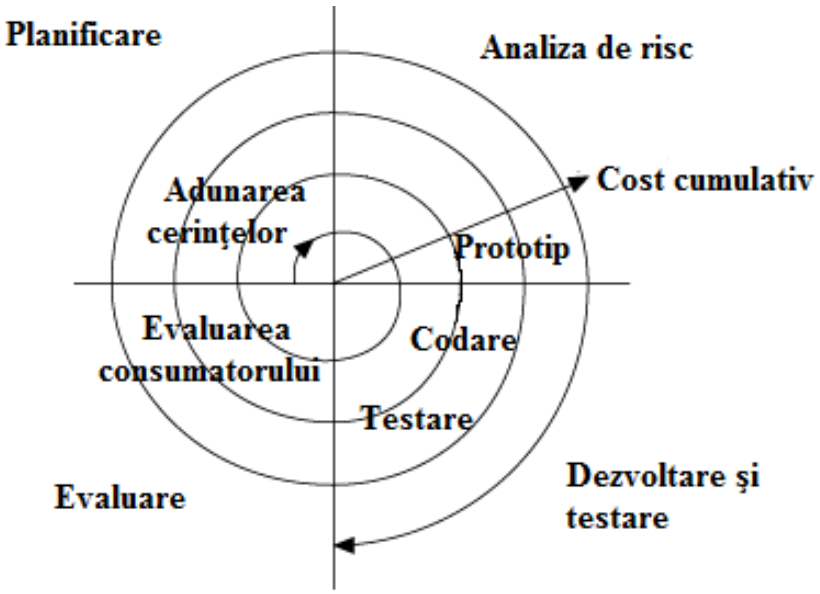
Avantaje:

- software funcţional disponibil devreme în ciclul de viaţă;
- mai flexibil (cerinţele şi scopurile pot fi modificate mai ieftin);
- mai uşor de testat şi de corectat erorile în timpul unei iteraţii mai mici;
- mai uşor de gestionat riscul, deoarece piesele cu posibile defecte sunt identificate şi reparate în timpul propriilor iteraţii;

Dezavantaje:

- fiecare fază a unei iteraţii este rigidă şi nu se suprapune cu altele;
- pot apărea probleme referitoare la arhitectura sistemului deoarece nu toate cerinţele sunt specificate la început pentru întreg ciclul;

1.4 Modelul spiralei:



Modelul spiralei este similar cu cel incremental, cu un accent pus pe analiza riscurilor. Există patru faze: planificarea, analiza de risc, dezvoltarea şi testarea şi evaluarea. Un proiect software trece în mod repetat prin aceste etape în decursul unor iteraţii (numite spirale în cadrul acestui model). Spirala de bază începe cu faza planificării, sunt definite cerinţele şi riscul este estimat. Fiecare fază ulterioară construieşte pe spirala de bază.

Cerinţele sunt descoperite în faza de planificare. În etapa de analiză a riscului, un proces este efectuat pentru a identifica riscul şi pentru a găsi soluţii alternative. Prototipul este produs la sfârşitul acestei faze.

Software-ul este produs în faza de dezvoltare şi este testat.

Etapa de evaluare permite cumpărătorului să evalueze rezultatele proiectului la zi, înainte de continuarea către următoarea spirală.

Componenta unghiulară reprezintă progresul iar raza reprezintă costul.

Modelul spiralei poate fi generalizat după cum urmează:

- cerinţele sistemului sunt definite în detaliu. Acest pas presupune de obicei intervievarea tuturor utilizatorilor produsului;
- un proiect preliminar este creat pentru noul sistem. Această fază este cea mai importantă a modelului. Toate alternativele posibile şi disponibile care pot contribui la eficientizarea proiectului din punctul de vedere al costului suntloate în calcul şse realizează strategii de folosire a acestora. Etapa a fost adăugată special pentru a identifica şi rezolva toate posibilele riscuri din dezvoltarea proiectului. Dacă riscurile indică orice fel de incertitudini în cerinţe, se pot folosi prototipuri pentru a continua cu datele disponibile şi pentru a găsi soluţii posibile pentru ajustarea la schimbările de cerinţe;
- un prim prototip al noului sistem este construit din proiectul preliminar. Este un sistem la scară mică şi reprezintă o aproximare a caracteristicilor produsului final;
- un al doilea prototip este construit printr-o procedură în 4 paşi:

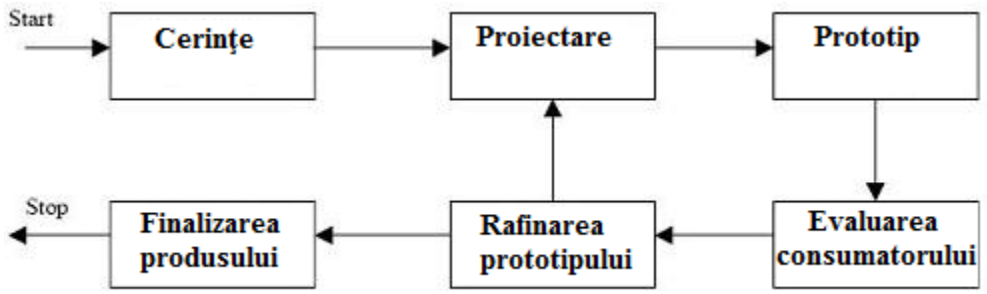
- se evaluează primul prototip;
- se definesc cerinţele următorului prototip;
- se planifică şi se proiectează următorul prototip;
- se construieşte şi se testează următorul prototip;

Modelul spiralei este folosit de obicei în proiecte mari. Pentru cele mai mici, dezvoltarea software agilă este o alternativă viabilă. Armata statelor unite a adoptat acest model pentru programul „Future Combat Systems". Este de asemenea rezonabil să fie folosit modelul în cazurile în care scopurile sunt instabile dar arhitectura trebuie realizată astfel încât să permită cantităţi mari de încărcare şi stres

Avantaje:
- cantitate mare de analiză a riscului;
- util în cazul proiectelor mari;
- software-ul este produs devreme în ciclul de viaţă;

Dezavantaje:
- poate fi un model costisitor;
- analiza de risc necesită expertiză foarte specifică;
- succesul proiectului este foarte dependent de fază analizei de risc;
- nu funcţionează bine pentru proiecte mici;

1.5 Modelul cu prototipuri



Aceasta este o versiune ciclică a modelului liniar. Odată ce analiza cerinţelor este îndeplinită şi proiectarea este terminată, procesul de dezvoltare este pornit. Odată ce prototipul este creat, este dat consumatorului pentru evaluare. Consumatorul oferă feedback dezvoltatorului care rafinează produsul conform cu aşteptările consumatorului. După un număr finit de iteraţii, pachetul final este dat consumatorului. În această metodologie, software-ul evoluează ca rezultat al schimbului periodic de informaţii dintre consumator şi dezvoltator.

Acesta este modelul cel mai popular din industria IT. Majoritatea produselor software de succes au fost create folosind acest model, întrucât este foarte dificil să înţelegi toate cerinţele utilizatorilor dintr-o singură încercare. Există multe variante ale modelului, din cauza diverselor stilurilor manageriale ale companiilor. Versiuni noi ale unui produs software apar ca rezultat al modelului cu prototipuri.

În general prototipul simulează numai câteva aspecte ale atributelor programului final şi poate fi complet diferit de implementarea finală.

Scopul convenţional al unui prototip este de a permite utilizatorilor de software să evalueze propunerile dezvoltatorilor încercându-le practic şi nu pe baza descrierilor. Utilizarea prototipurilor poate fi utilă şi end user-ilor pentru a descrie cerinţe pe care dezvoltatorii nu le-au considerat, aşadar controlarea prototipului poate fi un factor cheie în relaţia comercială dintre dezvoltatori şi clienţi.

Folosirea prototipurilor are o serie de beneficii: proiectantul soft şi dezvoltatorul pot obţine feedback de la utilizatori devreme în ciclul de viaţă. Clientul şi contractorul pot vedea dacă produsul software respectă cerinţele pe care se construieşte programul soft. De asemenea, oferă informaţii inginerului soft legate de estimările iniţiale şi de acurateţea lor, acesta putând determina dacă se pot respecta deadline-urile.

Există două mari categorii de utilizare a prototipurilor:

1.5.1 Prototipuri cu aruncare:

Numite şi prototipuri la capătul apropiat. Prototipurile cu aruncare sau rapide se referă la crearea unui model care va fi până la urmă aruncat în loc să devină o parte integrată a software-ului final. După ce se termină adunarea cerinţelor preliminare, un model funcţional simplu al sistemului este construit pentru a arăta vizual utilizatorilor cum arată cerinţele lor aplicate într-un sistem finalizat

Prototipurile rapide implic crearea unui model al diferitelor părţi ale sistemului într-o fază incipientă, după o investigaţie relativ scurtă. Metoda folosită la construirea modelului este de obicei informală, cel mai important factor fiind viteza cu care modelul este pus la dispoziţie. Modelul devine apoi punctul de început de la care utilizatorii reexaminează aşteptările şi clarifică cerinţele. Apoi, modelul prototip este aruncat şi sistemul este dezvoltat formal, bazat pe cerinţele identificate.

Motivul cel mai evident pentru folosirea acestei abordări este că poate fi îndeplinită repede. Dacă utilizatorii primesc un feedback rapid al cerinţelor lor, vor putea să le schimbe devreme în procesul dezvoltării produsului. Astfel de schimbări sunt foarte eficiente din punctul de vedere al costului din moment ce în momentul respectiv nu există nimic de reluat. Dacă un proiect este schimbat după ce s-a efectuat o muncă semnificativă, orice schimbare minoră poate cauza eforturi mari pentru a fi implementată, din moment ce sistemele software pot avea dependenţe. Viteza este crucială în implementarea unui prototip de aruncat, deoarece cu un buget limitat de bani şi timp se poate cheltui puţin pe un prototip care nu va fi luat în considerare.

Un alt avantaj este abilitatea de a se construi interfeţe pe care utilizatorii le pot testa. Interfaţa cu utilizatorul este ceea ce acesta vede ca fiind sistemul şi văzându-l în faţa sa, îi este mult mai uşor să înţeleagă cum va funcţiona.

Prototipurile pot fi clasificaţi în funcţie de gradul în care seamănă cu adevăratul produs ca aparenţă şi interacţiune. O metodă de a crea un prototip de aruncat cu fidelitate redusă este prototiparea pe hârtie. Prototipul este implementat cu creionul pe hârtie şi mimează funcţia adevăratului produs, dar nu arată deloc ca acesta. O altă metodă prin care se pot modela prototipuri de fidelitate crescută este de a folosi un GUI (Graphic User Interface) în care se crează un prototip care arată ca produsul dorit dar nu are nicio funcţionalitate.

Rezumatul acestui tip de prototipare este următorul:

a) scrierea cerinţelor preliminare;
b) proiectarea prototipului;
c) utilizatorul foloseşte prototipul şi specifică noile cerinţe;
d) se repetă paşii b) şi c) până când cerinţele nu se mai modifică;
e) scrierea cerinţelor finale;
f) dezvoltarea produselor reale;

1.5.2 Prototipuri evolutive

Prototiparea evolutivă este diferită de cea cu aruncare, întrucât scopul principal este de a construi un prototip foarte robust într-o manieră structurată şi de a-l rafina constant. Prototipul evolutiv, atunci când este construit va forma inima noul sistem. Când se dezvoltă un sistem folosind această metodă, acesta este constant rafinat şi reconstruit. Tehnica permite echipei să adauge trăsături, sau să facă schimbări care nu ar fi putut fi concepute în faza de cerinţe şi proiectare.

Pentru ca un sistem să fie folositor, trebuie să evolueze prin folosire în mediul pentru care a fost proiectat. Un produs nu este niciodată terminat, el se maturizează constant pe măsură ce mediul de operare se modifică.

Prototiparea evolutivă are avantajul că toate prototipurile sunt sisteme funcţionale. Deşi e posibil să nu aibă toate funcţionalităţile pe care utilizatorii le-au dorit, ele pot fi folosite ca o bază intermediară până la livrarea sistemului final.

Dezvoltatorii se pot concentra pe dezvoltarea părţilor sistemului pe care le înţeleg, în loc să lucreze la dezvoltarea întregului produs.

1.6. Modelul dezvoltării rapide a aplicaţiilor (RAD)

RAD modelează un proces liniar secvenţial cu un ciclu foarte scurt de dezvoltare. Este o adaptare de mare viteză a modelului în cascadă. Viteza este mare datorită abordării axate pe construirea componentelor. Este folosit în special pentru aplicaţiile sistemelor informatice şi prevede următoarele faze:

- Business modeling:

fluxul de informaţie între funcţiile business este modelat astfel încât să răspundă la următoarele întrebări:
Ce informaţie contribuie la funcţionarea procesului?
Ce informaţie este generată şi cine o generează?
Unde se duce informaţia?
Cine o procesează?

- Modelarea datelor:

Fluxul de informaţii definit ca parte a fazei de business modeling este rafinat într-un set de obiecte de date care sunt necesare pentru susţinerea proiectului. Atributele fiecărui obiect sunt identificate, iar relaţiile dintre aceste obiecte sunt definite.

- Modelarea proceselor:

Obiectele de date sunt transformate pentru a obţine fluxul de informaţii necesar pentru a implementa funcţia de business.

- Generarea de aplicaţii:

Modelul RAD presupune folosirea uneltelor RAD precum VB, VC++, Delphi. Sunt refolosite componente existente de program, sau sunt create componente reutilizabile. În orice caz, unelte automate sunt folosite pentru a facilita construirea software-ului.

- Testarea şi predarea:

Deoarece modelul RAD pune accent pe reutilizare, multe componente de program au fost deja testate. Astfel este minimizat timpul de testare şi de dezvoltare.

Bibliografie:

- http://codebetter.com/blogs/raymond.lewallen/archive/2005/07/13/129114.aspx
- http://www.stylusinc.com/Common/Concerns/SoftwareDevtPhilosophy.php
- http://en.wikipedia.org/wiki/Spiral_model
- http://ieeecomputersociety.org

--Mălăescu Alexandru --

2.1. Ciclul de viaţă în cascadă

2.1.1 Schema generală

2.1.2 Descrierea schemei

2.1.3 Paşi de analiză

2.1.4 Aplicarea ciclului de viaţă în cascadă pe un produs software

2.2 Ciclu de viaţă în V

2.2.1 Schema generală

2.2.2 Descrierea schemei

2.2.3 Aplicaţii ale ciclului de viaţă în V pe un produs software

2.1. Ciclul de viață în cascadă

2.1.1 Schema generală:

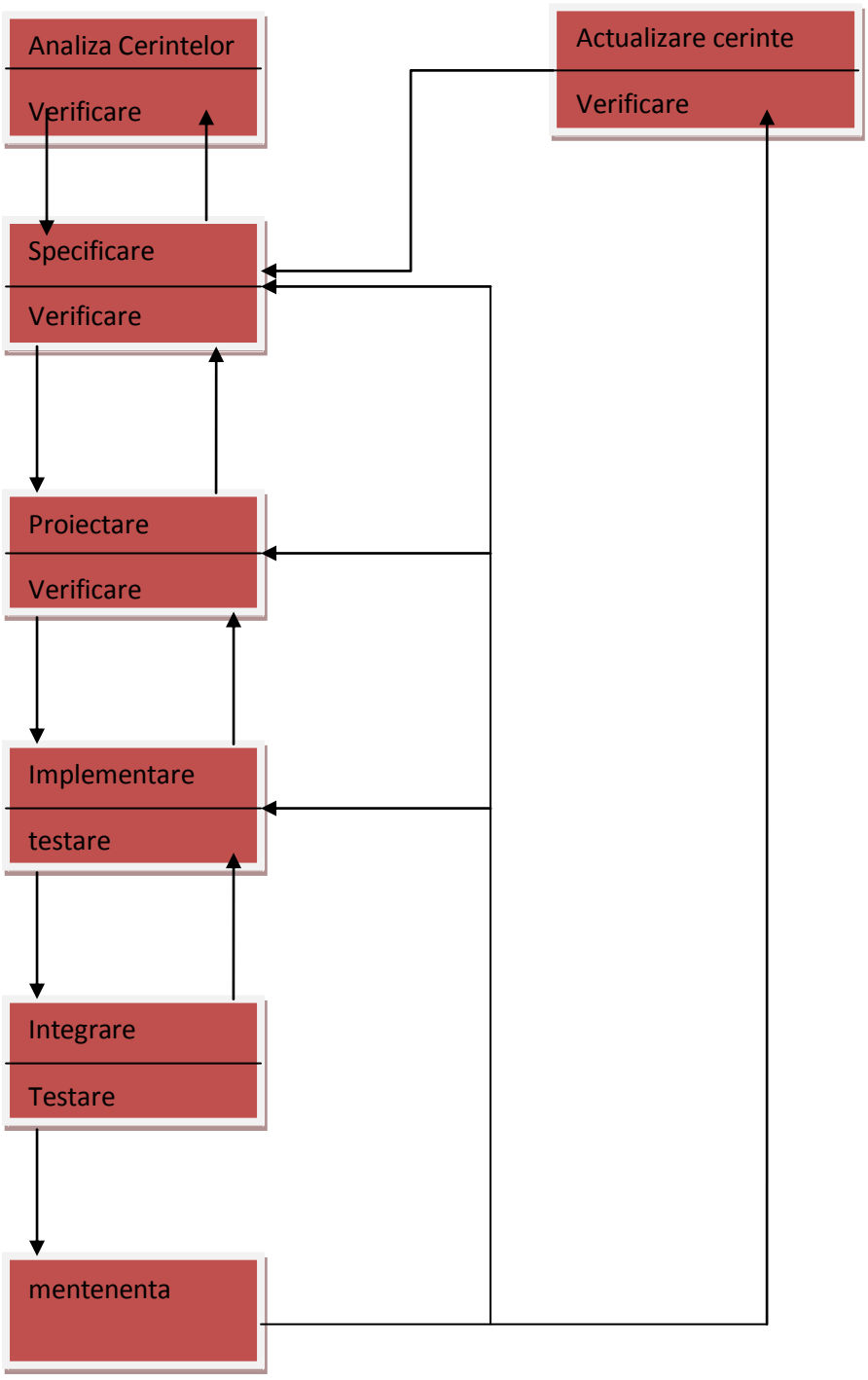


Figura 1

2.1.2 ***Descrierea schemei***

In faza de *analiza a cerintelor* se stabilesc obiectivele proiectului, precum si identificarea restrictiilor. Restrictiile sunt de mai multe tipuri: restrictii specifice problemei pe care produsul software isi propune sa o rezolve, restrictii financiare, restrictii legale, restrictii sociale. In cazul in care, proiectul isi propune sa imbunatateasca un produs software deja existent, in cadrul fazei de analiza, se vor studia caracteristicile dar si neajunsurile produsului software existent in piata.

In faza de *formulare a specificatiilor* este necesara elaborarea unui document in care sunt precizate riguros functiile produsului.

In faza de *proiectare a produsului*, sunt necesare urmatoarele etape: proiectarea structurilor de date, arhitectura software a produsului, detalierea altgoritmilor, proiectarea interfetelor, determinarea cerintelor hardware, alegerea tehnologiilor de implementare.

Implementarea si testarea produsului presupune: implementarea structurilor, scrierea codului, testarea fiecarui modul pentru identificarea erorilor si a nivelului in care acesta corespunde specificatiilor.

In *faza de integrare si testare* se testeaza interactiunea dintre componente, se verifica concordanta intre functiile realizate de produs si cerintele impuse, se valideaza produsul din punct de vedere al executantului, se trimite produsul la beneficiar pentru testele de acceptanta.

Mentenanta presupune intretinerea, adapatarea dar si dezvoltarea continua a produsului.

Se observa ca schema ciclului de viata in cascada prezinta reactii. In cazul in care in faza curenta apar erori, putem modifica etape ale fazei anterioare astfel incat erorile ce apartin fazei curente sa dispara. Observam ca operatia de testare apare doar in fazele de *implementare* si *integrare*. *Paşii din ciclu nu sunt discreţi în totalitate, prelucraţi o singură dată. Mai degrabă, ciclul de dezvoltare a sistemelor este iterativ şi evolutiv. El defineşte un flux secvenţial principal din punctul de origine pentru "a identifica noi probleme şi oportunităţi". Fiecare pas reprezintă un punct principal şi are transmiteri identificabile ce simbolizează întregul. La orice pas din ciclu pot fi descoperite problemele şi/sau oportunităţile neidentificate anterior. În astfel de cazuri este importantă asigurarea integrării corecte în sistem a soluţiei la noua problemă, sau noua oportunitate. Efectuarea chiar a unei schimbări minore la un sistem fără a lua în considerare ceea ce s-a stabilit anterior poate cauza efecte neanticipate şi de nedorit. Aceste efecte pot dăuna grav exploatării sistemului. Prin urmare, aşa cum se arată în figura 1, toţi paşii ciclului de dezvoltare a sistemelor permit întoarcerea la punctul de origine în orice moment. Această caracteristică face ca ciclul de viaţă să prezinte, mai degrabă, o caracteristică în "spirală" sau în "vârtej" decât una pur secvenţială sau o caracteristică în "cascadă".*

2.1.3 Paşi de analiză

Pas 1: Analiza de oportunitati si deficiente ale produsului actual

Identificarea de noi probleme si oportunitati, dar este necesara si o analiza a produsului software existent, pentru intelegerea clara a problemei.

Pas2: Revizuire documentatie si specifiatiilor ale produsului existent

Sistemul existent trebuie să fie bine analizat şi documentat înainte de proiectarea, dezvoltarea şi implementarea schimbărilor. Analizarea sistemului existent implică:
activităţi precum sunt următoarele:
–revizuirea sistemului de producţie;
– luarea deciziilor ţinând seama de fluxul de producţie;
– revizuirea informaţiilor curente disponibile pentru asistarea luării deciziilor (de exemplu, tranzacţiile şi rapoartele);
– identificarea deficienţelor din sistemul informaţional existent.

De îndată ce a fost analizat, sistemul informaţional existent este documentat pentru activităţile ulterioare şi pentru cerinţe de comunicare.

Pas 3: Ce anume se doreste sa realizeze sistemul nou proiectat

După ce au fost depistate deficienţele din sistemul informaţional şi sistemul existent a fost bine analizat şi documentat, pot fi determinate cerinţele informaţionale. Deoarece sistemul informaţional trebuie să conţină informaţiile necesare luării deciziilor, soluţiile la problemele informaţionale pot fi obţinute prin restructurarea informaţiilor existente sau luareaîn considerare a altora noi.

Pas 4: Cum realizam ceea ce am propus la pasul 3

Colectarea informaţiilor şi analiza cerinţelor stabilesc criteriile pentru identificarea mijloacelor alternative de soluţionare a problemelor. De aceea, în timp ce pasul anterior defineşte "ce" este de dorit, acest pas defineşte "cum" se va face. Personalul necesar şi tehnologiile viabile identificate sunt incluse în sistem, putând fi structurate pentru a susţine soluţia definită în pasul anterior. În general, sunt disponibile mai multe variante, care diferă prin gradul de realizare a sarcinii prestabilite.

Pas 5: Proiectarea sistemului

Până la acest punct au fost identificate soluţiile dorite şi mijloacele de realizare a lor. Acum este posibilă dezvoltarea şi testarea sistemului. Acest pas conţine instalarea hardwareului sau software-ului necesar, precum şi generarea şi testarea programelor de calculator. Software-ul poate fi achiziţionat ca un sistem complet necesitând unele modelări sau poate fi dezvoltat de către organizaţie. Produsul final al acestui pas este un sistem operabil şi de încredere care satisface obiectivele iniţiale de proiectare pentru rezolvarea unei probleme sau pentru a profita de o oportunitate.

Pas 6: Implementarea sistemului

După ce noul sistem a fost dezvoltat şi testat, se poate realiza conversia de la vechiul la noul sistem. Un element cheie al procesului de implementare îl reprezintă rezolvarea problemelor organizaţionale şi de comportament care apar adesea. Implementarea de noi sisteme informaţionale schimbă de obicei – uneori dramatic – locurile de muncă, responsabilităţile, cerinţele oamenilor şi relaţiile de comunicare. Pentru a face faţă factorului uman şi pentru a minimiza rezistenţa la schimbare, acestea trebuie previzionate. Corect implementate, sistemele pot fi foarte motivatoare şi pot ajunge la participanţii din organizaţie.

Pas 7: Mentenanta

După implementarea noului sistem, este important de apreciat cât de eficiente sunt soluţiile oferite la diverse probleme. Evaluarea, prin urmare, înseamnă aprecierea gradului de variaţie dintre performanţele planificate şi cele actuale ale sistemului. Dacă noul sistem a eşuat în atingerea obiectivelor din proiectare sau prezintă noi probleme sau oportunităţi, s-ar putea să trebuiască iniţiat un nou ciclu de viaţă al dezvoltării sistemului. Dacă noul sistem operează satisfăcător, atunci sistemul poate fi menţinut la nivelul curent de exploatare până când apar noi probleme sau oportunităţi.[1]

2.1.4 Aplicarea ciclului de viaţă în cascadă pe un produs software

In continuare, voi exemplifica cele 7 etape pe un produs software. Voi considera un sistem de gestiune al tranzactiilor pe baza cardurilor magnetice. Dupa cum se cunoaste, identificarea utilizatorului se realizeaza pe baza codului pin inscris pe card. Sistemul citeste codul pin de pe card si solicita utilizatorului introducerea codului de la tastatura. Daca utilizatorul a introdus corect codul pin (codul pin introdus de utilizator se potriveste cu ceea ce a citit sistemul), acestuia i se permite sa efectueze diferite tranzactii (vizualizare cont, retragere numerar din cont, plati la diversi furnizori, etc) .

In pasul 1, vom analiza sistemul existent si vom identifica deficientele precum si noile oportunitati de rezolvare a lor. O deficienta fundamentala este aceea ca sistemul valideaza utilizatorul doar pe baza codului pin. In cazul in care cardul este clonat, sau acesta este furat iar raufacatorul reuseste sa faca rost de codul pin, sistemul va permite raufacatorului accesul la contul utilizatorului valid. Asadar este necesara o imbunatatire a sistemului, astfel incat validarea utilizatorului sa se faca pe baza unei trasaturi specifice acestuia care sa nu poata fi furata. Sistemele care folosesc pentru identificare trasaturile umane (amprenta, semnatura, trasaturi ale irisului, etc) poarta denumirea de sisteme biometrice. Presupunem ca sistemul face identificarea utilizatorului pe baza trasaturilor irisului. In acest caz, utilizatorul introduce cardul magnetic, iar sistemul va citi informatiile de pe card. Pe baza lor, sistemul acceseaza o baza de date si gaseste inregistrarea corespunzatoare utilizatorului curent. In continuare, utilizatorul va introduce codul pin. Daca acesta introduce corect codul, sistemul va capta o imagine prin intermediul unei camere, din care va extrage trasaturile irisului utilizatorului curent. In cazul in care trasaturile userului curent coincid cu cele din baza de date, sistemul va autentifica utilizatorul. In caz contrar sistemul va respinge utilizatorul.

[1] **PRIVIRE DE ANSAMBLU ASUPRA DEZVOLTĂRII UNUI SISTEM INFORMAŢIONAL ÎN INDUSTRIA MINIERĂ**
Dr.ing.Ec. **GOLDAN Tudor,** *şef lucrări, Universitatea din Petroşani*

In pasul 2 este necesara o revizuire a specificatiilor sistemului existent. O documentatie a sistemelor ATM din punct de vedere al modului de comunicare, poate fi gasita in standardele : *RFC 1932, RFC 1483, RFC 1577, RFC 1755, RFC 1626.*

La pasul 3 se stabileste *ce* anume doreste sa aduca in plus noul sistem. Sunt necesare deci anumite specificatii :

- Sistemul va solicita intr-o prima etapa utilizatorului sa introduca cardul
- In etapa 2, sistemul citeste informatiile de pe card si identifica utilizatorul intr-o baza de date corespunzatoare
- In etapa 3, sistemul va realiza o prima autentificare a utilizatorului, solicitand acestuia introducerea codului pin
- In cazul in care autentificarea de la pasul 3 este cu succes, sistemul va trece la pasul 4 si va realiza o noua autentificare pe baza unei trasaturi biometrice a acestuia (trasaturi ale irisului). In cazul in care captura se potriveste cu atributul corespunzator din baza de date, sistemul va autentifica userul. In cazul in care etapa 3 nu este cu succes, sistemul va respinge utilizatorul.
- In cazul in care autentificarea de la pasul 4 nu se realizeaza cu succes, sistemul va respinge utilizatorul. In caz contrar, utilizatorul va avea acces la cont si va putea initia noi tranzactii.

La pasul 4 trebuie sa precizam cum realizam ce ne-am propus la pasul anterior. Asadar pentru a realiza ce ne-am propus vom adauga la sistemul existent urmatoarele facilitati :

- O camera care va captura trasaturi ale irisului, si va transmite sistemului captura respectiva
- Implementarea unor algoritmi care extrag din captura anumiti parametrii de interes
- Adaugarea unui nou atribut la baza de date existenta care va stoca parametrii biometrici
- Implementarea unor algoritmi care sa compare captura curenta cu captura sablon din baza de date

La pasul 5, in faza de proiectare , vom adauga la produsul existent urmatoarele componente :

Harware:

- Interfata seriala de comunicare cu camera
- O camera ce va captura trasaturi ale irisului

Software:

- Algoritmi de extragere a parametrilor
- Algoritmi de comparare intre captura curenta si sablonul din baza de date
- Actualizarea bazei de date curente cu un camp suplimentar

La pasul 6, se trece la implentarea si testarea noilor facilitati prezentate la pasul 5. Algoritmii de procesare ale capturelor pot fii implementati in C++ pentru a spori viteza de procesare. Noile facilatati ale bazei de date pot fi implementate in limbajul SGBD-ului specific bazei de date respective.

Ultimul pas, *mentenanta*, presupune urmarirea comportarii noului sistem. Se analizeaza timpul de procesare necesar unei autentificari. Se incearca diverse ipostaze de autentificare: fie prin clonarea unui card, fie prin utilizarea cardului original de un potential

raufacator. In ambele cazuri se analizeaza timpii de raspuns in caz de frauda. In cazul in care timpii de procesare sunt prea mari este necesara optimizarea algoritmilor, deci intoarcerea la pasul de implementare.

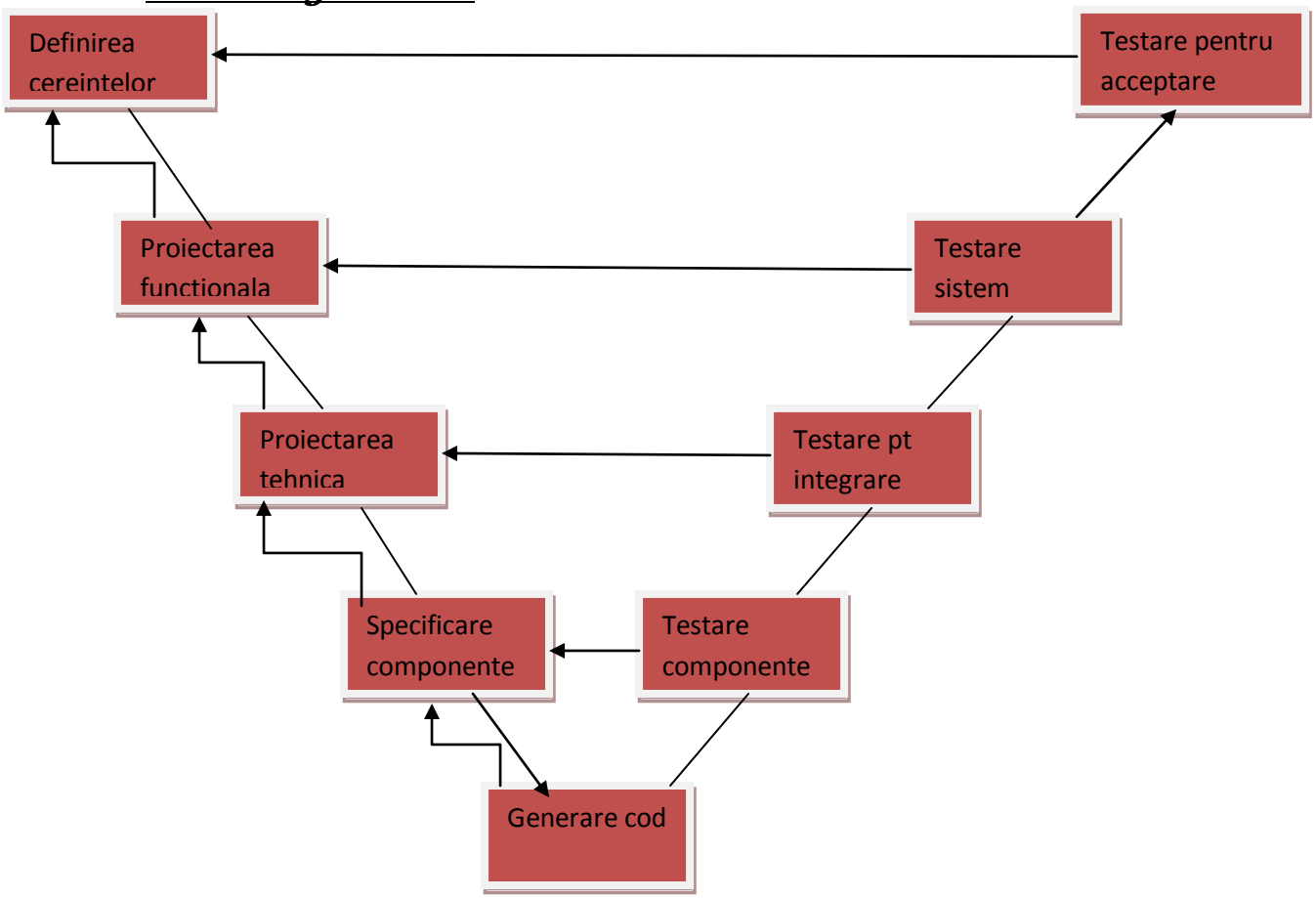
Managerii pot folosi una din cele două abordări principale de control ce sunt relevante pentru definirea problemelor şi a oportunităţilor. Aceste două abordări sunt numite *controlul reacţiilor* şi respectiv *cel al oportunităţilor*.

Controlul reacţiilor implică tratarea problemelor imediat ce ele apar – o abordare de tip management prin excepţii. Este cea mai uşoară şi cea mai folosită tehnică. De exemplu, dacă clienţii sesizează încetinirea onorării comenzilor de livrare a cărbunelui, managementul ar putea reacţiona prin încercarea de a rezolva problema. Altfel, managementul "trăieşte destul de bine singur". Cu alte cuvinte "dacă nu este stricat, nu-l repara".

Controlul oportunităţilor caută în mod continuu oportunităţi ce ar putea fi benefice pentru organizaţie. Decât să trăiască destul de bine singur, managementul caută îmbunătăţiri continue. Aceasta este o situaţie mult mai agresivă şi poate aduce venituri considerabile. De exemplu, managementul poate sesiza ocazia de a câştiga mai mulţi clienţi prin implementarea unui nou sistem de tratare a comenzilor, care va oferi livrări către clienţi într-un mod superior faţă de concurenţi. Aici se poate spune "dacă nu este stricat, mai este încă timp pentru a-l repara".[2]

2.2 Ciclu de viaţă în V

2.2.1 Schema generală



[2] **PRIVIRE DE ANSAMBLU ASUPRA DEZVOLTĂRII UNUI SISTEM INFORMAŢIONAL ÎN INDUSTRIA MINIERĂ**
Dr.ing.Ec. **GOLDAN Tudor,** *şef lucrări, Universitatea din Petroşani*

2.2.2 *Descrierea schemei*

In faza de *definire a cerintelor* se stabilesc obiectivele noului produs software. *Proiectarea functionala* stabileste *ce* anume trebuie sa realizeze produsul software. *Proiectarea tehnica* evidentiaza *cum* anume vor fi implementate tehnicile stabilite in faza de *proiectare functionala*. In faza de *specificare a componentelor* vor fi precizate functiile pe care trebuie sa le implementeze principalele *componente* ale produsului. *Faza de generare a codului* descrie *codul* propriu-zis al produsului.

Observam ca, fata de ciclul de viata in cascada, aici etapele de testare apar in toate fazele (de la faza de *definire a cerintelor*, pana la faza de *specificare a componentelor*). Testarea nu mai este doar o faza cu care se incheie codarea ci devine o componenta integrata a ciclului de viata al proiectului. Pregatirea testarii la nivelul unei faze poate fi pregatita inca din faza de analizei cerintelor, in paralel cu desfasurarea altor faze, ceea ce scurteaza timpul de dare in folosinta al produsului.

2.2.3 *Aplicaţii ale ciclului de viaţă în V pe un produs software*

Pentru exemplificare voi considera acelasi produs software prezentat si in cadrul ciclului de viata in cascada, si anume un sistem ATM.

Faza 1- definirea cerintelor

Se doreste imbunatirea unui sistem ATM, care realizeaza autentificarea utilizatorului prin recunoasterea de trasaturi umane (caracteristici specifice irisului) . Asadar gradul de securitate impotriva falsilor utilizatori va creste semnificativ.

Faza 2- Proiectarea functionala

Se doreste ca sistemul sa achizitioneze o captura din care sa extraga parametrii fundamentali ce dau informatiile corespunzatoare despre irisul utilizatorului curent. Parametrii astfel extrasi vor fi comparatii cu parametrii sablon stocati intr-o baza de date. In urma rezultatului compararii, sistemul va lua decizia de respingere/ acceptare a utilizatorului curent.

Faza 3- Proiectarea tehnica

Pentru achizitia unei capturi, sistemul ca comanda o camera digitala. Camera realizeaza captura si o transmite sistemului. Sistemul apeleaza anumiti algoritmi care extrag parametrii de interes din captura. Sistemul solicita bazei de date, parametrii sablon corespunzatori utilizatorului curent. SGBD-ul furnizeaza sistemului parametrii ceruti. Sistemul apeleaza anumiti algoritmi care realizeaza compararea intre parametrii sablon si parametrii extrasi din captura curenta. Algoritmii comunica sistemului rezultatul testului. Pe baza acestui rezultat sistemul decide daca va accepta/ rejecta utilizatorul curent.

Faza 4- Specificare componente

Pentru realizarea sistemului propus, sunt necesare urmatoarele componente:

Componente hardware: camera digitala, interfata seriala de comunicare intre sistem si camera.

Componente software: algoritmi de extragere de trasaturi, algoritmi de comparare a trasaturilor, baza de date care sa stocheze sabloanele de trasaturi.

Faza 5- Generarea codului

Observatie: Pentru sporirea vitezei de executie a produsului, putem realiza modulul de tesatare a unei componente concomitent cu realizarea altei componente (exemplu : modulul de testare al interfetei de comunicare seriala intre sistem si camera, respectiv modulul de realizare al algoritmilor de extragere de trasaturi) .

--Enache Marinel --

1.1 Performanţe şi decizii optime ale ciclurilor de viaţă

3.1.1 Model de proiectare cu unelte software
3.1.2 Produsele pentru dezvoltare software
3.1.3 Alegerea celui mai rapid ciclu de viaţă

3.2 Ciclul de viaţă al distribuţiei software, etape

3.2.1 Pre-alfa
3.2.2 Alfa
3.2.3 Beta
3.2.4 Seigo
3.2.5 Release Candidate şi Gold
3.2.6 Versiuni stabile/nestabile

3.1 Performanţe şi decizii optime ale ciclurilor de viaţă

3.1.1 Model de proiectare cu unelte software

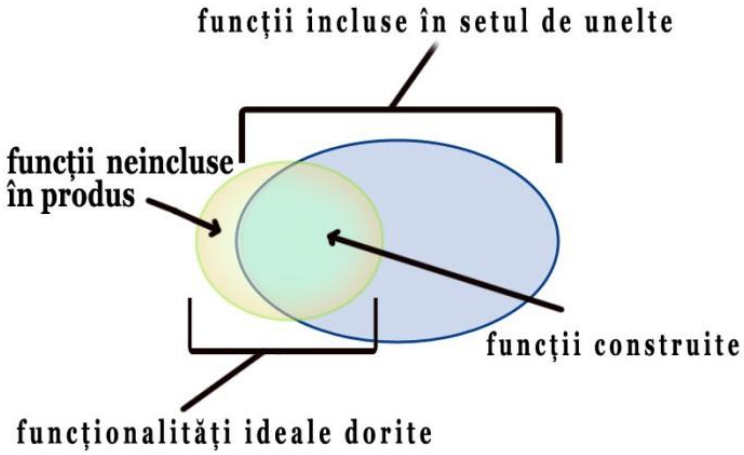
Modelul de proiectare cu instrumente al ciclului de viaţă este o abordare radicală, care este utilizată numai în medii extrem de sensibile la timp.

Pe măsură ce instrumentele au devenit din ce în ce mai flexibile şi mai puternice, aplicaţii complexe, medii vizuale de programare, soft-uri specializate de baze de date, au contribuit la creşterea numărului de proiecte care au în vedere utilizarea instrumentelor de dezvoltare.

Ideea din spatele acestui model, ce utilizează unelte de proiectare, este aceea că se poate introduce o capacitate în produs, numai dacă este susţinută direct de instrumente software existente.

Termenul de instrumente poate semnifica biblioteci de coduri şi clase, generatoare de cod, limbaje de dezvoltare rapidă şi alte pachete software care pot reduce dramatic timpul de implementare.

După cum este figurat în imagine, rezultatul utilizării acestui model este inevitabil, acela că nu se vor putea implementa toate funcţionalităţile pe care ideal vrei să le incluzi, dar prin alegerea anumitor instrumente se vor realiza majoritatea funcţionalităţilor dorite. Când timpul reprezintă o constrângere, există posibilitatea implementării unui număr de funcţionalităţi mai mare decât prin utilizarea altei abordări, dar aceste funcţionalităţi sunt cele ce se introduc rapid şi uşor, nu întotdeauna cele ideale . [1]



Conceptul de produs proiectat cu unelte software, poate oferi un avantaj excepţional al vitezei de dezvoltare dar în acelaşi timp oferă puţin control al funcţionalităţii produsului decât alte modele ale ciclului de viaţă.

Acest model poate fi combinat cu alte modele flexibile ale ciclului de viaţă.

Se poate realiza un model spirală iniţial care conduce la identificarea capacităţilor instrumentelor software, pentru depistarea cerinţelor de bază şi pentru a determina dacă modelul de proiectare cu unelte software este posibil. Se poate combina acest model cu livrarea în etape sau livrarea evoluţionistă sau proiectare cu programare.

Proiectarea utilizând unelte are câteva dezavantaje datorită lipsei de control ale produsului, nu se pot implementa toate caracteristicile produsului de necesitate şi nici nu se vor implementa alte caracteristici în modul dorit, acest lucru duce la dependenţa de producători de software a proiectanţilor. [1]

Dacă programele dezvoltate sunt mici, precum majoritatea software-urilor de unică folosință, atunci nu este o problemă, dar dacă se consideră scrierea unor programe complexe și durabile în timp, atunci fiecare furnizor ale cărui produse sunt utilizate, poate constitui o verigă slabă în evoluția produsului. [1]

3.1.2 Produsele pentru dezvoltare software

COTS (Commercial, off-the-shelf software)

O alternativă care este uneori neglijată de entuziasmul datorat creării unui nou sistem, este opțiunea achiziției de software universal. Produsele software universale rareori satisfac cerințele, dar sunt disponibile imediat și în timpul intervenției între momentul achiziționării unui astfel de soft și lansarea propriului tău soft, utilizatorii vor avea cel puțin câteva facilități valoroase. Ei pot învăța să lucreze în jurul limitării produsului din moment ce beneficiază de software particularizat și pe măsură ce timpul trece, produsele software comerciale sunt revizuite și potrivite îndeaproape nevoilor .

Aplicațiile software particularizate probabil nu se vor alinia viziunii mentale ale produsului software ideal, iar comparațiile între produsele software particularizate și cele universale tind să compare efectiv soft-ul universal cu idealul produs software particularizat.

Cu toate acestea atunci când se proiectează un soft, trebuiesc făcute concesii de proiectare, cost sau programare și există posibilitatea ca produsul particular creat să ajungă la nivelul așteptat. [1]

3.1.3 Alegerea celui mai rapid ciclu de viață

Diferite proiecte au diferite cerințe, chiar dacă toate trebuiesc dezvoltate cât mai rapid posibil. Prezentarea anterioară a modelelor ciclu de viață pentru produsele software, aspectele variabile și diferitele combinații pot constitui o gamă largă de opțiuni în proiectare, însă care variantă se decide a fi mai rapidă se poate studia prin parcurgea unor pași esențiali.

Nu există un așa numit model rapid de dezvoltare a unui ciclu de viață software, deoarece majoritatea modelelor depind de contextul în care sunt utilizate, și de aceea unele modele pot fi considerate mai rapide decât altele, ceea ce înseamnă că fiecare va fi rapid în diferite situații, și lent în altele, iar dacă un model funcționează bine, cu siguranță nu are randament dacă este aplicat într-un caz nepotrivit.

Pentru alegerea adecvată a unui astfel de model se examinează proiectul și se poate răspunde la o serie de întrebări precum cele de mai jos :

- Cât de bine înțelege clientul și proiectantul cerințele la începutul proiectării și dacă această înțelegere se poate schimba pe măsura ce se avansează în etapa de proiectare?
- Cât de bine a înțeles proiectantul arhitectura sistemului și ar avea nevoie să intervină cu modificări la jumătatea proiectului?
- Cât de serios trebuie tratată fiabilitatea sistemului?
- Cât este necesar să se planifice și să se proiecteze înainte sistemul în perioada acestui proiect pentru versiuni viitoare?
- În ce măsură această proiectare este riscantă ?
- Există constrângeri cu privire la timp?
- Este necesar ca la dispoziția clienților să fie vizibil progresul proiectării curente?
- Este necesar să existe un nivel de gestiune și administrare vizibil progresivă în perioada de proiectare?

- Cât de sofisticat trebuie să fie acest proces de proiectare pentru utilizarea cu succes a unui model ciclu de viață?

O serie de astfel de întrebări sunt menite să decidă proiectantul în alegerea modelului optim. În general o abordare liniară sau în cascadă a proiectării, realizată eficient, poate constitui un mijloc rapid de dezvoltare, dar dacă există dubii cu privire la acest tip de abordare se poate alege cea mai flexibilă pentru aplicația dată. [1]

Tabel cu puncte forte/slabe ale modelelor ciclu de viață software

Capacitatea modelului	Cascadă pură	Cod și reglare	Spirală	Cascadă modificată	Prototipuri evoluționiste
Funcționează cu cerințele slab înțelese	Ineficientă	Ineficientă	Excelentă	Acceptabilă către excelentă	Excelentă
Funcționează cu arhitectură slab înțeleasă	Ineficientă	Ineficientă	Excelentă	Acceptabilă către excelentă	Ineficientă către acceptabilă
Produce sisteme extrem de fiabile	Excelentă	Ineficientă	Excelentă	Excelentă	Acceptabilă
Produce sisteme cu creștere de pachete	Excelentă	Ineficientă către acceptabilă	Excelentă	Excelentă	Excelentă
Gestionează riscurile	Ineficientă	Ineficientă	Excelentă	Acceptabilă	Acceptabilă
Poate fi constrâns la un program predefinit	Acceptabilă	Ineficientă	Acceptabilă	Acceptabilă	Ineficientă
Supraîncărcare redusă	Ineficientă	Excelentă	Acceptabilă	Excelentă	Acceptabilă
Oferă corecții intermediare	Ineficientă	Ineficientă către excelentă	Acceptabilă	Acceptabilă	Excelentă
Asigură progres vizibil clienților	Ineficientă	Acceptabilă	Excelentă	Acceptabilă	Excelentă
Asigură management vizibil	Acceptabilă	Ineficientă	Excelentă	Acceptabilă către excelentă	Acceptabilă
Necesită dezvoltări sofisticate	Acceptabilă	Excelentă	Ineficientă	Ineficientă către acceptabilă	Ineficientă

Fiecare notă/evaluare poate fi “Ineficientă”, “Acceptabilă”,” Excelentă”, iar distincții mai fine nu ar fi necesare la acest nivel. Evaluarea consemnată în tabel se bazează pe cel mai bun potențial al modelului, dar eficiența oricărui ciclu de viață depinde de modul implementării.

Pe de altă parte, dacă un model este necorespunzător într-o formă particulară , acest lucru poate fi compensat prin crearea unui model hibrid din unul sau mai multe modele descrise. Desigur, multe din criteriile prezente în tabel vor fi puternic influențate de considerații de dezvoltare, altele decât alegerea inițială a ciclului de viață.

Descrierile detaliate cu privire la prezentările criteriilor modelului ciclu de viață sunt redate mai jos:

Funcționează cu cerințele slab înțelese - spune cât de bine funcționează un model când fie proiectantul sau clientul înțelege greșit cerințele sistemului sau când clientul este predispus la schimbarea cerințelor. Indică cât de bine se potrivește modelul cu dezvoltarea de software.

Capacitatea modelului	Livrarea în etape	Livrarea evoluționistă	Proiectarea programată	Proiectarea cu unelte	Softuri comerciale
Funcționează cu cerințele slab înțelese	Ineficientă	Acceptabilă către excelentă	Ineficientă către acceptabilă	Acceptabil	Excelentă
Funcționează cu arhitectură slab înțeleasă	Ineficientă	Ineficientă	Ineficientă	Ineficientă către excelentă	Ineficientă către excelentă
Produce sisteme extrem de fiabile	Excelentă	Acceptabilă către excelentă	Acceptabilă	Ineficientă către excelentă	Ineficientă către excelentă
Produce sisteme cu creștere de pachete	Excelentă	Excelentă	Acceptabilă către excelentă	Ineficientă	N/A
Gestionează riscurile	Acceptabilă	Acceptabilă	Acceptabilă către excelentă	Ineficientă către excelentă	Excelentă
Poate fi constrâns la un program predefinit	Acceptabilă	Acceptabilă	Excelentă	Excelentă	Excelentă
Supraîncărcare redusă	Acceptabilă	Acceptabilă	Acceptabilă	Acceptabilă către excelentă	Excelentă
Oferă corecții intermediare	Ineficientă	Acceptabilă către excelentă	Ineficientă către excelentă	Excelentă	Ineficientă
Asigură progres vizibil clienților	Acceptabilă	Excelentă	Acceptabilă	Excelentă	N/A

Asigură management vizibil	Excelentă	Excelentă	Excelentă	Excelentă	N/A
Necesită dezvoltări sofisticate	Acceptabilă	Acceptabilă	Ineficientă	Acceptabilă	Acceptabilă

Funcționează cu arhitectură slab înțeleasă- specifică cât de bine lucrează modelul în cazul dezvoltării unei noi aplicații sau când dezvoltarea se face într-o aplicație familiară, dar sunt slabe cunoștințe de manipulare.

Produce sisteme extrem de fiabile - referă câte defecte poate avea un sistem dezvoltat cu un model al ciclului de viață care este supus rulării.

Produce sisteme cu creșteri mari de pachete - se referă la gradul de dificultate al adaptării sistemului în creșterea dimensiunii și diversității în timpul vieții.
Acesta implică modificări ale sistemului în diferite moduri care nu sunt anticipate de proiectanții originali.

Gestionează riscurile - se referă la sprijinul modelului pentru identificarea și controlarea riscurilor la program, riscurilor la produs și alte riscuri.

Poate fi constrâns la un program predefinit - se referă la cât de bine un model al ciclului de viață sprijină livrarea produsului la o dată limită.

Supraîncărcare redusă – se referă la cantitatea gestionată și solicitată tehnic necesară pentru utilizarea modelului efectiv. Supraîncărcarea include planificarea, monitorizarea stării, producerea documentației, achiziționarea de pachete și alte activități care nu sunt implicate direct în producerea soft-ului însuși.

Permite corecții intermediare - se referă la abilitatea schimbării aspectelor signifiante ale produsului în perioada de mijloc a programului de dezvoltare. Acest lucru nu include schimbarea misiunii de bază a produsului ci include extinderea semnificativă a acestuia.

Asigură clienților un progres vizibil – reprezintă o extindere în care modelul automat generează semne de progres pe care clientul le poate urmări pentru starea proiectului.

Asigură administrare cu progres vizibil - reprezintă o extindere în care modelul generează automat semne de progres pe care conducerea le folosește în monitorizarea stării proiectului.

Necesită puțină administrare sau dezvoltări sofisticate - se referă la nivelul de educație și formare de care are nevoie proiectantul pentru a utiliza modelul cu succes. Acesta include un nivel sofisticat necesar monitorizării progresului folosind modelul, pentru evitarea riscurilor inerente în model, pentru evitarea pierderii de timp folosind modelul și realizarea beneficiilor care au dus la utilizarea modelului în primul rând. [1]

3.2 Ciclul de viaţă al distribuţiei software, etape

Introducere

O versiune software este o distribuție, publică sau privată, a unei versiuni de produs software inițiale sau noi și îmbunătățite. De fiecare dată când un program sau un sistem suferă modificări, inginerii de software și companiile în lucru decid modul în care vor face distribuția programului sau sistemului, sau schimbările pe care le vor aduce acelui program sau sistem în timp.

O metodă de distribuție a unei modificări de sistem e dată de patch-ul software descărcat sau livrat pe CD. [3]

Etapele distribuției în ciclul de viață

Ciclu de viață al distribuției software este compus din diferite stagii care descriu stabilitatea unei componente de program și volumul de dezvoltare care este necesar înainte de lansarea produsului.

Fiecare versiune majoră a unui produs de regulă trece prin diferite stări pe măsură ce noi caracteristici sunt adăugate, sau prin stagiul alfa , când este activ depanat, sau beta, și în final un stagiu cu toate defectele eliminate numit și stagiu stabil.

Stagiile intermediare pot fi recunoscute, sunt formal anunțate în mod regulat de către dezvoltatorii de software, dar uneori acești termeni sunt utilizați în scop informativ pentru descrierea stării produsului. Convențional, nume de cod sunt adesea utilizate de multe companii pentru versiuni prioritare în lansare produsului, deși actualele produse și caracteristici sunt rareori ținute în secret. [3],[2]

Schema evoluției unui produs software

3.2.1 Pre-alfa

Este considerată o formă incompletă a programului lansată inițial, înainte de emiterea variantelor alfa sau beta. Prin comparație cu ultimile variante amintite, modelul pre-alfa nu conține toate caracteristicile și când este folosit se referă la toate activitățile desfășurate în timpul proiectului software înainte de testarea software. Aceste activități pot include analiza cerințelor, proiectare software, dezvoltare de software și unități de testare.

În lumea Open Source, sunt câteva tipuri de versiuni pre-alfa. Versiunea Milestone include un set specific de funcționalități și sunt lansate imediat ce acestea sunt completate. Nightly builds sunt acele versiuni care sunt de regulă automat verificate de la sistemul de control al reviziei și construite, tipic peste noapte. Aceste versiuni permit celor ce le testează să verifice imediat funcționalitățile recent implementate și să descoperă noile probleme, defecte ale sistemului. [3]

3.2.2 Alfa

Realizarea variantei alfa a produsului software este forma înmânată celor ce se ocupă de testarea soft, care sunt persoane diferite de inginerii de software, dar uzual fac parte din organizația sau comunitatea de dezvoltatori de software.
Ocuparea rapidă a poziției softului pe piață duce uneori la angajarea de către companiile dezvoltatoare a mai multor clienți externi sau parteneri valoroși pentru efectuarea testărilor alfa și acesta este considerată o extindere în faza de testare a produselor de tip alfa.
În prima fază de testare, general dezvoltatorii testează produsul folosind o tehnică a cutiei-albe.
Validările adiționale sunt apoi efectuate folosind tehnica cutiei-negre sau cea a cutiei-gri, de o altă echipă de testare dedicată sistemului, uneori simultan. Trecerea la testarea cutiei-negre în interiorul organizației este cunoscută ca lansarea alfa. [3]

3.2.3 Beta

O versiune Beta este prima variantă lansată în afara organizației sau comunității dezvoltatoare de software, cu scopul evaluării sau testării cutiei-negre, gri de către lumea reală. Procesul de livrare a versiunii beta către alți utilizatori poartă numele de distribuție beta. Nivelul software beta în general conține toate caracteristicile, dar poate conține și elemente nedorite sau simple erori într-un mod variabil mai puțin grav.

Utilizatorii unei versiuni beta sunt numiți testări beta și adesea sunt clienți obișnuiți sau potențiali clienți ai organizației care dezvoltă software. Ei primesc software-ul gratuit sau la un preț redus, dar acționează ca testări liberi.

Versiunile beta testează suportul produsului, mesajul introducerii pe piață (în timpul recrutării clienților beta), fabricația produsului și fluxul de canal global și canal atins.
Versiunile de software beta sunt cel mai probabil necesare demonstrațiilor interne și selectării clienților, dar nestabile și nepregătite pentru lansarea definitivă.
Unii dezvoltatori se referă la această etapă ca o previzualizare, un prototip, sau o previzualizare tehnică sau ca un acces timpuriu. [3]

De multe ori aceste stagii beta iau naştere când dezvoltatorii anunţă o stagnare a unor caracteristici ale produsului, indicând imposibilitatea acceptării unor cerinţe caracteristice pentru versiunea produsului şi numai problemele de soft , bug-urile sau facilităţile neimplementate pot fi adresate.
Dezvoltatorii lansează fie un beta închis, fie un beta deschis, prima fiind o versiune dedicată unui grup select de utilizatori de test, în timp ce versiuna a doua se adresează unei comunităţi mari de utilizatori, de obicei publicul larg.
Testării furnizează orice eroare întâlnită şi uneori mici caracteristici pe care le-ar dori implementate în varianta finală.

Când o variantă beta devine disponibilă publicului general este adesea larg utilizată de tehnologii savvy şi de cei familiarizaţi cu versiunile anterioare ca şi când ar fi produsul finit.

Dezvoltatorii uzuali ai variantelor gratuite sau open-source beta, le lansează publicului larg în timp ce proprietarii de beta se îndreaptă către un grup restrâns de testări.

Beneficiarii de versiunile beta originale trebuie să semneze un contract de nedivulgare. O lansare poate fi numită caracteristică completă când echipa produsului este de acord cu cerinţele funcţionale ale sistemului şi nicio ulterioră facilitate nu va fi inclusă în distribuţie, dar erori signifiante vor continua să existe. Companiile cu un proces software formal vor tinde să pătrundă în perioada beta împreună cu o listă de bug-uri care trebuie să fie fixate pentru părăsirea perioadei beta, şi unele companii pun aceste liste la dispoziţia clienţilor şi testărilor.

Internetul a permis distribuţia de software într-un mod rapid şi necostisitor, iar companiile au adoptat tehnici flexibile de abordare în vederea utilizării sensului cuvântului “beta”.

Cei de la Netscape Communications au lansat o versiune alfa a browserului cu aceeaşi denumire şi au numit-o beta, în ideea păstrării unui statut adecvat. În februarie 2005 , ZDNet a publicat un articol despre un fenomen recent al versiunilor beta adesea prelungite pe perioade de ani de zile şi folosite ca făcând parte din nivelul de producţie, exemplul celor de la Gmail sau Google care au fost sub forma beta o lungă perioadă de timp şi nu au renunţat la acest aspect luând în considerare utilizarea largă, dar în cele din urmă Google a părăsit această variantă prin ianuarie 2006. Această tehnică permite unui dezvoltator să întârzie oferind suport total şi/sau responsabilitate pentru problemele rămase şi uneori versiunea beta este considerată o variantă finală a produsului. [3],[2]

Originile variantelor alfa şi beta

Termenul de beta test aplicat softurilor provine de la un test făcut produselor hardware IBM convenţie datând din perioada cardurilor perforate şi maşinilor de sortat.

Componentele hardware treceau iniţial prin testul alfa pentru funcţionalitatea preliminară şi etapa de fezabilitate a fabricaţiei, apoi urma testul beta care verifica dacă funcţiile au fost realizate corect şi dacă sunt produse la o dimensiune necesară în comerţ şi se încheia cu un test C pentru fiabilitatea produsului. Odată cu apariţia computerelor programabile şi primele programe partajabile, IBM a continuat cu aceeaşi terminologie pentru testarea produselor software. Testele beta erau conduse de oameni sau grupuri altele decât dezvoltatorii, şi cum alte companii au început crearea de software pentru propriile utilizări şi distribuţii către alţii, evoluţia terminologiei a stagnat şi s-a păstrat până în prezent. [3]

3.2.4 Seigo

Stagiul Seigo este un nivel al evoluției software când librăriile necesare produsului sunt finalizate (sau aproape gata) dar restul produsului necesită lustruire și finisare. Stagiul intervine între varianta beta si varianta finală a dezvoltării soft deoarece produsul este încă nepregătit pentru a intra în faza finală de livrare, chiar dacă librăriile sunt de calitate.

Termenul provine după o controversată discuție pe tema linux în "The Linux Action Show !", evenimentul lansării celei de-a doua versiuni finale a KDE 4 neridicându-se la standardul unei variante release candidate, discuție purtată pe Skype între 2 prezentatori și Aaron Seigo.

3.2.5 Release Candidate şi Gold

Termenul de " release candidate" se referă la o versiune cu potențial de produs final, gata să fie lansată chiar dacă se ivesc erori. În acestă etapă, produsul prezintă în mod normal codul complet. Corporația Microsoft adesea folosește termenul de release candidate (distribuția finală)

În timpul anilor 1990, Apple Inc a utilizat termenul de „maestru de aur" pentru distribuțiile finale și maestrul de aur final reprezentând distribuția generală disponibilă. Alți termeni includ gamma (și ocazional delta, sau alte litere grecești) pentru versiunile care sunt substanțial complete, dar aflate încă sub test, și omega pentru testarea finală a versiunilor care sunt considerate a fi bug-free, și pot intra în producție oricând.

O distribuție este numită cod complet când echipa de dezvoltare este de acord că nicio sursă internă de cod nou nu va fi adăugată acestei distribuții, însă pot exista modificări de coduri pentru a înlătura defecte și chiar schimbări de documentație și fișiere de date, și de asemenea coduri pentru cazurile test sau alte utilități.

Noile coduri vor fi adăugate într-o viitoare distribuție.

Distribuția Gold sau distribuția general disponibilă

Versiunile gold lansate sau cele generale sunt formele finale a unui produs particular. Este tipic aproape identic cu distribuția finală, cu bug-uri fixate în ultimul minut. O distribuție Gold este considerată foarte stabilă și relativ un bug-free cu o calitate corespunzătoare pentru distribuții largi și folosită de utilizatori finali. În distribuțiile comerciale de software, această versiune poate fi semnată (folosită să permită utilizatorilor finali posibilitatea verificării codului ce nu a fost modificat de la lansare). Expresia că un produs software "a trecut de aur" înseamnă că partea de cod este completă și va fi produs în număr mare și valabil curând pentru vânzare. Termenul de aur, anectodic se referă la utilizarea discului master de aur care a fost folosit în mod obișnuit în trimiterea variantelor finale producătorilor care îl folosesc să creeze copii de vânzare în masă. În unele cazuri acest disc este realizat chiar din aur, atât pentru estetică cât și rezistență la coroziune.

RTM sau RTW

Microsoft și alții folosesc termenul de „ distribuția finală " RTM (release to manufacturing) pentru a se referi la aceste versiuni(pentru produse comerciale ca Windows XP , ca în" Build 2600 cu distribuția RTM Windows XP"), și "distribuții Web"- RTW pentru produse gratuite descărcabile. Tipic, RTM reprezintă săptămânile și lunile înainte de GA(disponibilitatea generală) deoarece versiunea RTM trebuie stampilată pe disc, respectiv cutie ș.a.m.d. [3],[2],[4]

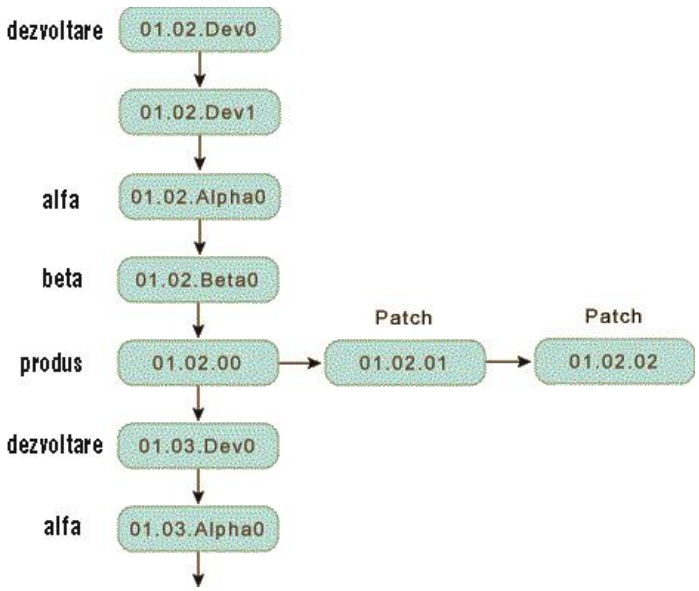
3.2.6 Versiuni stabile/nestabile

În programarea open source, numerele de versiuni sau termenii Stabil şi Nestabil disting stagiile de dezvoltare. Termenul de stabil se referă la o versiune a unui software care este substanţial identic cu o versiune care a trecut suficient printr –o testare de către lumea-reală pentru asigurarea corectitudinii softului, sau acele mici probleme existente sunt cunoscute şi detaliate.

Pe de altă parte termenul nestabil, nu înseamnă neapărat că există probleme, mai degrabă, acele îmbunătăţiri sau schimbări efectuate produselor software care nu au fost riguros testate. Utilizatorii unui astfel de software sunt sfătuiţi să folosească versiunile stabile dacă vor să-şi atingă interesele dorite, şi să utilizeze varianta instabilă când apariţia unor probleme nu ar afecta întreg programul.

În kernelul Linuxului, numerele de versiuni sunt compuse din 3 numere, separate de o perioadă.

Între versiunile 1.0.0 şi 2.6.x, distribuţiile stabile au al doilea număr par şi cele instabile au unul impar. Practica utilizării numerelor impare, pare pentru a indica stabilitatea distribuţiei a fost utilizată de alte proiecte open/closed source . [3],[5],[6]



Exemplificarea versiunilor lansate în perioada unui ciclu de viaţă software

Bibliografie:

[1] ***Steve Mcconnell - Rapid Development - Taming Wild Software Schedules***

[2] ***http://users.csc.calpoly.edu/~jdalbey/205/Mgmt/SDP.html***

[3] ***http://www.onestoptesting.com/release-life-cycle/***

[4] ***http://www.perforce.com/perforce/papers/life.html***

[5] ***http://support.motioneng.com/Software-MPI_04_00/basics/releases/sw_life_cycle.htm***

[6] ***http://web.progress.com/support/Progress_Product_Versioning_Release_Types_Lifecycle-Policy_2010.pdf***