Universitatea Politehnica București

Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Cicluri de viață ale software-ului

**Tudor Ioana** Coordonator științific: **Popa Marius** Conf. Dr. Ing. **Ștefan Stăncescu**

**Marinescu Alexandru Valentin**

Grupa: 441A

**2016**

Cuprins

1. Noțiuni generale - Tudor Ioana
2. Etapele ciclului de viață – Tudor Ioana
3. Modele de cicluri de viață
	1. Modelul în cascadă – Tudor Ioana
	2. Modelul în V – Tudor Ioana
	3. Modelul incremental – Popa Marius
	4. Modelul în spirală – Popa Marius
	5. Modelul agil – Popa Marius
	6. Modelul Code-and-Fix – Marinescu Alexandru Valentin
	7. Modelul cu prototipuri – Marinescu Alexandru Valentin
	8. Modelul haos – Marinescu Alexandru Valentin
4. Bibliografie
5. Noțiuni generale

Ca orice produs fabricat complex, un program este realizat urmând un anume proces de dezvoltare ce se bazează pe o formalizare a activităților. Scopul formalizării este obținerea unui ansamblu de mecanisme care, în cazul în care sunt aplicate sistematic, permit obținerea într-un mod repetitiv și fiabil a produselor software de calitate constantă. Descrierea procesului rămâne totuși generală întrucât nu este posibil să se definească un standard unic adaptat la toate tipurile de aplicații.

Dezvoltarea unui program poate fi vazută ca stabilirea unui șir de descrieri din ce în ce mai precise și din ce în ce mai apropiate de un program executabil și de documentația sa. Trecerile de la o descriere la alta sunt deseori numite rafinări successive. O dată dezvoltat un program, el este pus în exploatare. Se pun astfel problem de întreținere, de corectare a defectelor, de ameliorare a anumitor caracteristici sau de urmărire a evoluției cerințelor. Întreținerea poate impune modificarea funcțiilor sistemului și deci, un proces de redezvoltare. Din această cauză se vorbește despre un ciclu de viață al unui program.

Ciclul de viață al unui produs software reprezintă, așadar, intervalul de timp de la momentul deciziei de realizare până la retragerea sau înlocuirea totală a acestuia cu un nou produs software, reprezentând orizontul de timp în care operează și evoluează produsul. Cu alte cuvinte, ciclul de viață este o abordare sistematică ce presupune dezvoltarea, utilizarea, mentenanța și retragerea software-ului.

Din punct de vedere al utilizatorilor cele mai importante componente ale ciclului de viață sunt utilizarea curentă (operaționalitate) și mentenanța întrucât produsele software sunt dezvoltate pentru a satisface nevoile consumatorilor. Din punct de vedere al realizatorilor, etapele cele mai importante sunt cele în care se construiește produsul software.

Durata ciclului de viață al unui produs software depinde de caracteristicile de calitate si performanță ale acestuia dar și de o serie de factori externi produsului software precum: stabilitatea funcțională, relațiile sale cu mediul, dinamica metodelor, a tehnicilor, a conceptelor etc.

Ciclul de realizare este o noțiune derivată din cea de ciclu de viață acoperind intervalul dintre momentul deciziei inițiale de realizare și până la punerea în funcțiune a produsului software. Acest interval, împărțit în general în etape, presupune analizarea problemei, proiectarea produsului software, implementarea codului și testarea și experimentarea acestuia. Prin urmare, ciclul de realizare reprezintă o parte a ciclului de viață și anume cea care are ca scop dezvoltarea produsului.

Dezvoltarea de software de înaltă calitate necesită procese de producție software, principii de inginerie software pe care să se bazeze aceste procese și practici ale ingineriei software. Principiile dezvoltării software sunt legate de calitate, management și de inginerie. Ele se reflectă în calitatea produsului, în timpul de realizare și în costurile implicate.

Principiile calității sunt:

* Prevenirea defectelor
* Detectarea și corectarea defectelor existente
* Stabilirea și eliminarea cauzelor care pot produce anumite defecte
* Audit și conformitate cu standarde și proceduri

 Principiile de management sunt:

* Planificarea tuturor activităților necesare realizării optime din punct de vedere al timpului, bugetului, resurselor și standardelor de calitate
* Monitorizarea permanentă, îmbunătățirea progresivă a planurilor, revizuirea planurilor dacă anumite limite de toleranță au fost depășite
* Definirea clară a structurii, rolurilor, responsabilităților și liniilor de comunicație între grupuri de persoane implicate în realizare

 Principiile de inginerie sunt:

* Descrierea clară a problemei care de regulă întâmpină următoarele dificultăți majore:
* Cerințele sunt exprimate în termenii problemei reale și trebuie traduse în termenii uzuali contextului software
* Utilizatorul nu poate preciza complet cerințele sale sau nu poate găsi legătura dintre ele
* Definirea și selectarea soluției pintr-o definire clară a componentelor
* Controlul riguros al relațiilor dintre componente
1. Etapele ciclului de viață

Pe baza celor menționate mai sus, un ciclu de viață al unui produs software ar putea consta din următoarele etape:

1. Analiza cerințelor
2. Proiectarea software-ului
3. Dezvoltarea software-ului
4. Testarea software-ului
5. Producția software-ului
6. Mentenanța și eliminarea

**Analiza cerințelor**

În cadrul acestei etape este foarte important să se documenteze cât mai exact cerințele pentru software-ul ce urmează a fi dezvoltat sau achiziționat. Lipsa unor cerințe clare poate determina dezvoltarea unui produs slab din punct de vedere calitativ, depășirea timpului alocat și a resurselor disponibile pentru proiect și cel mai probabil va determina creșteri semnificative ale costurilor necesare reluării documentării, analizei și concretizării aplicării cerințelor.

Prin urmare, project managerii și persoanele care se ocupă de marketing trebuie să găsească răspunsuri pentru specificații legate de publicul tință care va folosi produsul, modul de utilizare al acestuia, tipul de date introduse în sistem etc. După ce au fost strânse toate aceste informații se creaază un document ce deservește drept model în următoarea etapă a ciclului de viață a produsului.

**Proiectarea**

În etapa de proiectare sunt dezvoltate detaliile tehnice ale sistemului. Astfel, sistemul este descompus în componente mai ușor de controlat, numite module. Implementarea sistemelor complexe devine posibilă tocmai prin această descompunere modulară, în caz contrar fiind imposibil de stăpânit mulțimea detaliilor tehnici care ar trebui luate în considerare. Datorită proiectării modulare trebuie avut în vedere pentru fiecare modul doar detaliile corespunzătoare lui. Totodată, proiectarea modulară ajută și la întreținerea sistemului. O structură modulară bine proiectată este importantă atât pentru implementarea sistemului cât și pentru modificarea sa ulterioară. Prin urmare, acesta este unul din principalele motive pentru care paradigma orientată spre obiecte câștigă teren: un sistem care a fost proiectat ca fiind orientat pe obiecte este prin definiție un sistem modular.

**Dezvoltarea**

Această etapă se bazează pe specificațiile de proiectare furnizate de etapa anterioară și implicit pe detaliile despre arhitectura produsului stabilite anterior. Munca de implementare și dezvoltare este împărțită prin urmare pentru fiecare modul în parte, începând astfel programarea propriu-zisă. Cu alte cuvinte pentru fiecare modul, se scriu efectiv programele, se creează fișierele de date și se dezvoltă baze de date.

**Testarea**

În această etapă fiecare modul este testat în prima fază independent, într-un sistem bine proiectat pentru a simula interacțiunile dintre modulul țintă și restul sistemului. Acest tip de testare individuală este foarte utilă deoarece permite identificarea mai ușoară a posibilelor probleme. Ulterior se testează modulele combinate, adică se realizează o testare generală a întregului sistem.

**Producția**

În urma testării și corectării erorilor apărute, produsul este trimis la client pentru a fi folosit. Clientul verifică dacă produsul este corespunzător cerințelor sale și, dacă este mulțumit, își dă acordul pentru intrarea în etapa de mentenanță.

**Mentenanța**

Întrucât pe parcursul utilizării produsului de către client, pot apărea diverse probleme ce nu au fost identificate în etapa de testare, dezvoltatorul oferă o perioadă de mentenanță în care asigură o garanție a produsului. Această etapă semnifică mai exact corectarea anomaliilor în funcționare de către programatorul inițial, fără a modifică însă cerințele produsului.

1. Modele de cicluri de viață

Modelele ciclurilor de viață reprezintă diversele procese și metodologii care sunt selectate pentru dezvoltarea proiectelor în funcție de scopul și destinația acestora. Există multe modele de cicluri de viață care au fost dezvoltate cu rolul de a îndeplini anumite obiective necesare, definind diferitele etape prin care trebuie sa treacă un produs software și ordinea în care trebuie parcurse.

Alegerea unui astfel de model are un impact foarte mare asupra etapei de testare, influențând ce anume va fi testat și când sau unde va fi testat. De asemenea, influențează foarte mult tehnica ce trebuie aleasă pentru testare. Există mai multe modele sau metodologii de testare software printre care:

* Modelul în cascadă
* Modelul în V
* Modelul incremental
* Modelul în spirală
* Modelul agil

Este de la sine înțeles că alegerea unui model potrivit pentru produsul software ce se dorește a fi dezvoltat este foarte importantă întrucât pe baza acestui model se desfășoară procesele de testare și dezvoltare. Companiile aleg astfel modele diferite în funcție de ce este mai potrivit pentru aplicația software.

Cu toate acestea, în zilele noastre cel mai folosit este modelul “agil”, un tip de model incremental. Modelul în cascadă este foarte vechi, întrucât etapa de testare începe doar după ce etapa de dezvoltare s-a terminat, fapt ce conduce la apariția mai multor erori. Prin urmare, costurile de reparare a acestor erori sunt foarte mari. În cazul modelului agil, după realizarea fiecărui modul se realizează un demo pentru client. Așadar, clientul poate urmări fiecare caracteristică în parte și poate decide daca corespunde cerințelor sale sau nu.

Modelul în V este, de asemenea, adoptat de multe companii pentru produsele lor, el reprezentând practic modelul de verificare și validare. În cadrul acestui model, ciclul de viață al dezvoltatorului și cel al tester-ului sunt legate între ele. Cu alte cuvinte, testarea este realizată în paralel cu dezvoltarea. Similar, modelul incremental, cel în spirală și cel iterativ sunt bazate pe cerințele clientului și pe nevoile produsului.

* 1. Modelul în cascadă



Fig. 1

Modelul în cascadă reprezintă primul model de ciclu de viață definit, termenul fiind introdus de către Winston W. Royce în anul 1970. Dezvoltarea modelului cascadă îşi are originea în industriile de manufactură şi construcţii: medii fizice foarte structurate în care schimbările după implementare pot fi foarte costisitoare, dacă nu chiar imposibile. Din moment ce nu existau metodologii mai formale de dezvoltare software în acel moment, modelul orientat pe hardware a fost pur şi simplu adaptat pentru dezvoltarea software.

Numit si modelul clasic al ciclului de viață sau modelul liniar, modelul în cascadă prezintă dezvoltarea unui program ca o succesiune de etape ce se înlănțuie într-o derulare liniară, pornind de la analiza cerințelor până la livrarea produsului către client. Cu alte cuvinte, fiecare etapă trebuie finalizată complet înainte de a se putea trece la următoarea etapă. La sfârșitul fiecărei etape se realizează o revizuire pentru a determina dacă proiectul este pe drumul cel bun și dacă să se continue sau să se renunțe la proiect. Modelul în cascadă se pretează bine pentru proiectele mici ale căror cerințe sunt foarte clare.

Întrucât modelul în cascadă presupune finalizarea completă a unei etape înainte de trecerea la următoarea etapă, necesităţile de program ar trebui bine conturate înainte de a începe procesul de creare, altfel munca depusă pentru un concept ce se bazează pe cerințe incorecte este pierdută. Designul programului ar trebui să fie perfect înaintea începerii implementării, altfel implementarea produsului greşit duce la muncă pierdută.

De asemenea, modelul în cascadă pune accentul pe documentaţie (cum ar fi documentele din care reies necesităţile şi documentele de design), la fel ca şi codul sursă. În metodologiile mai puţin documentate, dacă membrii echipei părăsesc echipa, multe dintre cunoştinţe se pierd şi poate fi dificilă continuarea proiectului. Dacă există o documentaţie completă, membrii noi ai echipei pot să se familiarizeze cu proiectul prin citirea documentaţiei.

**Rezultatul unui proiect realizat cu ajutorul modelului în cascadă**

Documentele produse în urma aplicării acestui model sunt:

* Document de cerinţe
* Planul proiectului
* Document de design al sistemului
* Document de design detaliat
* Plan de test şi raport de test
* Cod final
* Manuale software (manualul utilizatorului, manual de instalare etc)
* Raporturi de evaluare

Cu excepţia raporturilor de evaluare, aceste docmente reprezintă chiar rezultatele etapelor. Pentru a valida un rezultat al unei etape înaintea începerii următoarei etape, se realizează deseori evaluări. Evaluările sunt necesare în special pentru etapele de realizare a cerinţelor şi a design-ului, din moment ce alte mijloace de certificare deseori nu sunt disponibile. Evaluările sunt întâlniri formale pentru a descoperi deficienţele într-un produs și au ca rezultat rapoartele de evaluare.

**Limitările modelului în cascadă**

Modelul în cascadă se bazează pe o secvență de etape bine delimitate. Documentele produse de fiecare etapă sunt evaluate în cadrul reviziilor care validează trecerea de la o etapă la alta. Din nefericire, proba efectivă a bunei sau a proastei funcționări a sistemului este realizată numai în cadrul fazei de integrare, când este posibilă evaluarea concretă a programului.

De asemenea, abordarea în cascadă conduce la rezultate satisfăcătoare numai în cazul în care este posibilă înlănțuirea etapelor fără prea multe problem. Asta presupune ca ansamblul cerințelor să fie perfect cunoscut și problema complet înțeleasă de analiști. Totodată, trebuie ca soluția să fie ușor de determinat de către de proiectanți și codificarea ei să fie simplă, ideal această să poată fi redusă la generarea automată a codului plecând de la documentele de proiectare.

În realitate, se constată că aceste condiții pot să nu fie îndeplinite datorită:

* Neînțelegerii cerințelor de către client sau analist
* Lipsei clarității cerințelor
* Alegerilor tehnologice
* Schimbărilor de personal

 Datorită acestor aspecte sunt necesare reveniri la etape anterioare ale procesului de dezvoltare. Dacă aceste reveniri sunt ocazionale și limitate la etapele adiacente, modelul în cascadă este pertinent.

**Avantaje:**

* Control total asupra etapelor în sensul că ele sunt ordonate și, deci sunt previzibile, prin evidențierea clară a ariei de cuprindere a fiecărei etape sau subetape;
* Este ușor de însușit de către membrii echipelor de analiză și proiectarea, inclusiv de cei noi, cu o experiență mai mică;
* Fiecare etapă este însoțită de o documentație perfect structurată (controlată);

**Dezavantaje:**

* Sistemul se predă doar după parcurgerea etapelor anterioare, ceea ce înseamnă o lungă perioadă de timp, sufficient ca beneficiarul să-și schimbe pretențiile;
* Nu prezintă o abordare dinamică a sistemelor
* Nu este deschis schimbărilor ce pot interveni pe parcurs
	1. Modelul în V

Modelul în V reprezintă o variantă a modelului în cascadă, prin care se introduc conceptele de sistem și componente (subsisteme), aplicându-se teste explicite la un sistem ierarhic pentru sporirea controlului asupra modului în care se desfășoară etapele.



 Fig. 2

 Partea din stânga este parcursă descendent și conține etapele propriu-zise, în timp ce partea din dreapta se parcurge ascendent și este reprezentată de verificările și validările elementelor create anterior. Modelul scoate în evidență cu mai multă claritate separările dintre ceea ce implică participarea utilizatorului, modelul arhitectural și cel al implementării. Astfel, utilizatorul este implicat doar în etapele din partea superioară a “V-ului”. Arhitectura sistemului este surprinsă în partea din mijloc. Etapa de implementare se referă la partea inferioară a diagramei și poate reprezenta fie asamblarea componentelor soft, fie codificarea unor componente.

 Modelul se potrivește și în mediul obiect-orientat deoarece simulează prototiparea și reutilizarea unor structuri superioare. Prin structurile ierarhice și modulare pe care le promovează, el oferă un control puternic asupra sistemului în curs de realizare. Acest lucru îl face utilizabil și în sistemele complexe. Modelul presupune abordarea și dezvoltarea pe componente a sistemului, adică abordarea pe părți și integrarea lejeră a lor. El devine și mai puternic dacă promovează iterația, adică reluarea unor etape, activități sau subactivități.

 De asemenea, modelul face distincție evidentă între verificare și validare. Prima se referă la testarea sistemului în diverse stagii ale sale (dacă s-a construit corect din punct de vedere logic) în timp ce validarea va scoate în evidență faptul că sistemul, în forma lui finală, corespunde sau nu cerințelor inițiale. Acest aspect este considerat un dezavantaj al modelului pentru că validarea se realizează prea târziu.

**Avantaje:**

* Simplu și ușor de folosit
* Activitățile de testare au loc înainte de codare. Acest aspect conduce la economie de timp și la o rată de succes mai mare decât în cazul modelului în cascadă
* Urmărire proactivă a defectelor, adică defectele sunt identificate încă din primele etape
* Este evitată acumularea de defecte
* Funcționează foarte bine pentru proiectele mici cu cerințe ușor de înțeles

**Dezavantaje:**

* Foarte rigid și puțin flexibil
* Software-ul este dezvoltat în etapa de implementare, deci nu sunt create prototipuri timpurii
* Dacă pe parcurs survin schimbări, documentele de test împreună cu cele ce conțin cerințele trebuiesc actualizate

Modelul în V se utilizează pentru proiectele mici și medii ale căror cerințe sunt clar definite și fixate. De asemenea, acest model ar trebui ales atunci când sunt disponibile resurse tehnice ample, însoțite de expertiza tehnică necesară. Pentru a alege modelul în V, este necesar un nivel mare de încredere din partea clientului, deoarece acest model prezintă riscuri mari datorită lipsei de prototipuri.

* 1. Modelul Incremental

Modelul ciclului de via "Incremental" este opus modelului « în cascadă ». El se bazeaz pe o idee foarte simpl: dac un sistem este prea complex pentru a fi neles, conceput sau realizat într-o singură faz este mai bine să fie realizat n mai multe faze, prin evoluie.



Fig. 3 – Schema modelului incremental

**Cum se aplică**

Într-o fază inițială:

* Se studiază scopul proiectului și fezabilitatea sa. În cazul deciziei de continuare a proiectului:
* Se detaliază cerințele utilizatorilor și se efectuează o analiză de nivel înalt, pentru a se determina cerintele software la un nivel general.
* Se determină o arhitectură generală a sistemului.
* Se împart cerințele în subseturi care pot fi implemenate în incremente separate. Se stabilește planificarea în timp a incrementelor.
* Fiecare increment implementeaza un subset de cazuri de utilizare de nivel inalt, care exprimă cerințele utilizatorilor. El este construit urmând abordarea “cascadă”: analiza detaliată a cerințelor din subset, proiectarea, implementarea și testarea. Rezultatul este un produs care satisface un subset al cerintelor și este livrat utilizatorilor. Un produs tipic constă din 10-50 incremente.

Se incepe cu o implementare simplă a unui subset al cerințelor software. Rezultă o primă livrare.

Scopul primei implementări este de a crea un produs la care utilizatorul poate reacționa. El trebuie să pună în evidență aspectele cheie ale problemei și să furnizeze o soluție suficient de simplă pentru a fi ințeleasă și ușor de implementat.

Fiecare nouă iterație include analiza ultimei versiuni și adăugarea de noi funcționalitați, ceea ce presupune reproiectarea, codificarea și testarea. Se urmărește ca proiectarea să fie directă, modulară, să suporte reproiectarea.

Analiza unei iteratii se bazeaza pe feedback-ul utilizatorului și pe instrumentele de analiză disponibile. Ea se referă la: modularitea produsului, cuplarea modulelor, utilizabilitatea, fiabilitatea, eficiența și realizarea scopurilor.

Fiecare iterație este o variantă îmbunatațită a celei anterioare, de aceea metoda se mai numește “îmbunatațirea iterativă” (iterative enhancement).

Se utilizeaza măsuri pentru evaluarea evoluției sistemului. Măsurile prin care se evaluează un software sunt dificil de înțeles ca valori absolute, dar schimbările valorilor lor în evoluția unui sistem sunt o bază de comparație. Astfel de măsuri sunt: numărul de defecte, efortul de actualizare, dimensiune, complexitatea, cuplarea modulelor. Schimbările diferitelor aspecte se pot monitoriza și se pot stabili limite pentru anumite măsuri, pentru a semnala probleme sau anomalii.

Utilizarea analizei și a măsurătorilor ca ghid în procesul iterativ este o diferență majoră între această metodă și metodele de “dezvoltare agila” (agile software development). Ele sunt suportul pentru determinarea eficienței procesului și a calitații produsului.

Fiecare iteraie a ciclului de via iterativ și incremental reproduce ciclul de via n cascad la o scar mai mic. Obiectivele unei iteraii sunt stabilite pe baza evalurii iteraiilor precedente. Documentaia este construit treptat, n timpul fiecrei iteraii. La sfritul dezvoltrii, documentele obinute au aceeai form cu cele obinute n maniera convenional.

**Planificarea**

Principalul risc în utilizarea unui model incremental este de a lucra într-o manieră “ad-hoc”. Determinarea unui plan de acțiuni este de primă importanță pentru succesul abordării incrementale. În faza de analiză preliminarăse determină scopul proiectului și se încearcă determinarea riscurilor majore ale proiectului, se determină o listă o cerințelor și constrangerilor mai importante, pentru a construi un plan de dezvoltare. Se stabilește natura fiecarui increment și ordinea de construire a incrementelor.

**Modelul incremental – se recomandă când:**

* cerinţele întregului sistem sunt clar definite şi înţelese;
* cerinţele majore sunt definite; cu toate acestea, unele detalii pot evolua în timp;
* este necesară o lansare mai devreme a produsului pe piaţă;
* se utilizează o noua tehnologie;
* există unele caracteristici şi obiective cu risc ridicat.

**Avantaje:**

1. În fiecare etapă este livrat un produs executabil, care satisface o parte din cerințele utilizatorului. Opus modelului cascadă în care se elaborează documente.
2. Prototipurile sunt livrate clientului/utilizatorilor.
3. Feedback-ul utilizatorilor este distribuit pe întreg parcursul dezvoltării.
4. În cazul apariției unor schimbari în cerințe, acestea pot fi încoporate în urmatorul prototip.
5. este mai flexibil – implică costuri mai mici la schimbarea scopului şi cerinţelor;
6. este uşor de testat şi depanat în timpul unei iteraţii mai mici;
7. reduce costurile iniţiale de livrare;
8. riscul este mai uşor de gestionat, deoarece piesele riscante sunt identificate şi tratate în timpul iteraţiei;

**Dezavantaje**

1. Ciclul de via incremental se bazeaz pe evoluia prototipurilor executabile. Din nefericire, programele nu se preteaz n mod natural evoluiei, din contr, ele sunt deseori foarte "fragile" la modificri. Efectul unei modificri locale se poate propaga n ansamblul aplicaiei. Fiecare nou increment poate necesita reorganizarea structurii interne, degradand arhitectura initiala a programului, facndu-l greu de verificat i de ntreinut.
2. Erorile de proiectare sunt mai greu de eliminat.
3. Abordarea obiect bazat pe modularitate i ncapsulare conduce la programe mai robuste i mai rezistente fa de schimbri. Din acest punct de vedere, abordarea obiect furnizeaza un cadru confortabil pentru dezvoltarea prin evoluie, ntr-o manier iterativ.
4. Clientul poate vedea ce se poate face si poate cere mai mult!
5. Abordarea incrementală se poate transforma ușor într-una « codifică și repară » (« build and fix »).

În cursul dezvoltrii, clientul poate utiliza prototipurile. Demonstrarea prototipurilor prezint numeroase avantaje:

- utilizatorul este pus n faa situaiilor de utilizare concrete care-i permit s-i defineasc mai bine dorinele i s le comunice echipei de dezvoltare;

 - utilizatorul devine partener al proiectului; el si ia partea de responsabilitate n noul sistem i de fapt l accept mai uor;

* 1. Modelul spirală

Barry Boehm a definit acest model plecând de la slabiciunile modelului cascadă, în special lipsa sa de flexibilitate la schimbări ale cerințelor. Este focalizat pe analiza riscurilor în mod incremental, repetând modelul cascadă într-o serie de cicluri.

Fiecare ciclu constă din 4 faze.

1. **Determinarea obiectivelor:** definirea produsului, determinarea scopurilor și a constrangerilor, generarea alternativelor.
2. **Evaluarea alternativelor**: analiza riscurilor, prototiparea
3. **Dezvoltarea produsului**: proiectarea de detaliu, testarea unitară, integrarea
4. **Planificarea urmatorului ciclu**: evaluarea de catre client, planificarea dezvoltarii și a livrarii către client.

**Caracteristici principale ale Modelului Spirală**

* este asemănător cu modelul incremental, dar cu mai mult accent pus pe analiza riscului
* are patru faze: planificare, analiza riscului, inginerie şi evaluare
* spirala de bază, începe în faza de planificare, cerinţele sunt colectate şi riscul este evaluat
* la sfârşitul fazei de analiza de risc este realizat un prototip
* software-ul este produs în faza de inginerie
* etapa de evaluare permite clientului să evalueze rezultatul proiectului înaintea spiralei următoare.

**Schema modelului spirală** este reprezentată în figura următoare:



Fig. 4 – Schema modelului spirala

**Avantajele modelului spirală**

1. propune o atitudine pro-activă asupra riscurilor, cu o presupunere explicită a riscurilor şi a rezolvării lor;
2. este foarte flexibil;
3. se realizează multe analize de risc, prin urmare este îmbunătăţită evitarea riscului;
4. este bun pentru proiecte mari şi cu misiuni critice;
5. se realizează un control asupra documentaţiei;
6. pot fi adăugate ulterior funcţionalităţi suplimentare;
7. software-ul este produs devreme în ciclul de viaţă software.

**Dezavantajele modelului spirală**

1. sunt aproape imposibil de estimat de la început timpul şi costurile necesare;
2. poate fi un model costisitor;
3. analiza de risc necesită expertiză ştiinţifică superioară;
4. succesul proiectului depinde foarte mult de faza de analiză de risc;
5. nu funcţionează bine pentru proiecte mai mici.

**Modelul spirală – se recomandă când:**

* evaluarea riscurilor şi costurilor este importantă;
* proiectele sunt cu risc mediu spre ridicat;
* utilizatorii nu sunt siguri de necesităţile lor;
* cerinţele sunt complexe;
* sunt realizate linii de produse noi;
* se aşteaptă schimbări semnificative (cercetare şi explorare).
	1. Modelul agil

Odată cu explozia dezvoltării de software, s-a ajuns la concluzia că modelul anterior (V-Model sau Waterfall) nu ținea cont de o **realitate obiectivă**:

Clienții nu pot defini de la început foarte clar și irevocabil cerințele.

Este una dintre cauzele cele mai importante pentru care proiectele de dezvoltare software nu reușesc să se termine la timp sau nu sunt considerate un succes de către cei care le-au cerut.

O altă probleme cu Waterfall este că progresul unui proiect este cu greu urmărit de către un client. Deși primește raporte detaliate despre ce se întâmplă, đin cauză că el nu vede nimic palpabil până la finalul proiectului ( până la etapa de lansare ), atunci îi este greu să aibă încredere în progresul proectului sau să sprijine eventuale decizii care trebuie luate pentru a finaliza proiectul.

Agile vine să rezolve cele două probleme descrise mai sus și pornește de la câteva principii de bază ( din care voi puncta doar pe cele ce au legătură cu problemele descrise mai sus ):
1. Prioritatea este satisfacerea clientului prin crearea de software valoros și funcțional
2. Schimbările sunt binevenite, indiferent de etapa de dezvoltare la care se află proiectul
3. Software funcțional este principala măsură a progresului
4. Simplicitatea este esențială

Schema de principiu a modelul agil este reprezentată în figura următoare:



Fig. 5 – Schema modelului agil

Ideea este că se lucrează împreună cu clientul. Spre deosebire de Waterall unde clientul înseamnă “ei” și echipa de dezvoltare suntem “noi”.

Dezvoltarea Agil ar putea să arate schematic așa: proiecte micuțe și stabile ca număr de features și lansate la momente pre-stabilite.

Modelul Agil se poate aplica in aproape orice proiect în care o schimbare survenită pe parcurs sau un bug descoperit mai târziu nu implică pierderi cu consecințe mari sau fatale. De exemplu nu se adoptă modele Agil în dezvoltare de software pentru industriile pe care le-am descris mai sus ca fiind foarte potrivite.

**Avantajele modelului agil**

1. obţinerea satisfacţiei clienţilor prin livrarea rapidă şi continuă a software-ului;
2. oamenii şi interacţiunile sunt mai accentuate în cadrul modelului decât procesele şi instrumentele - clienţii, dezvoltatorii şi testării interacționează constant;
3. versiuni de lucru sunt livrate frecvent (mai degrabă săptămâni decât luni);
4. conversaţia faţă-în-faţă este cea mai bună formă de comunicare;
5. cooperare strânsă, de zi cu zi între oamenii de afaceri şi dezvoltatorii produsului;
6. atenţie continuă către excelenţă tehnică şi proiectare bună;
7. adaptare regulată la circumstanţele de schimbare;
8. chiar şi schimbările târzii în cerinţe sunt binevenite;
9. este o abordare realistă pentru dezvoltarea de software;
10. promovează munca în echipă;
11. poate fi dezvoltat rapid;
12. necesarul de resurse este minim;
13. oferă versiuni de lucru parţiale anticipate;
14. este un model bun pentru mediile care se schimbă în mod constant;
15. există reguli minime;
16. permite o dezvoltare simultană şi livrare într-un context general planificat;
17. este uşor de gestionat;
18. oferă flexibilitate pentru dezvoltatori.

**Dezavantajele modelului agil**

1. în cazul unor livrabile software, în special mari, evaluarea efortului necesar pe întreg parcursul ciclului de dezvoltare software este dificilă de realizat;
2. nu se pune accent prea mare pe proiectare şi pe documentaţia necesară;
3. necesită dezvoltatori cu experienţă;
4. nu este potrivit pentru manipularea dependenţelor complexe;
5. există risc mai mare la durabilitate, mentenabilitate şi extensibilitate;
6. depinde foarte mult de interacţiunea cu clienţii, astfel încât, în cazul în care clientul nu este clar, echipa poate fi condusă într-o direcţia greşită;
7. există dependenţă individuală foarte mare, deoarece există documentaţie minimă generată;
8. transferul de tehnologie către noi membri ai echipei poate fi destul de dificil din cauza lipsei de documentaţie.
	1. **Modelul Code-And-Fix**

Modelul de code-and-fix este o metodologie de dezvoltare folosită frecvent în ingineria software.

Aceasta începe cu o planificare inițială simplă sau deloc. Se începe imediat dezvoltarea codului, iar rezolvarea diferitelor probleme sunt rezolvate pe măsură ce apar, până când proiectul este complet.

Code-and-fix este o alegere tentantă, atunci când programatorul este confruntat cu un timp de dezvoltare limitat deoarece dezvoltarea codului este începută imediat, astfel se pot vedea rezultate imediate.

În cazul în care se vor găsi probleme arhitecturale majore târziu în dezvoltare, de obicei, trebuie rescrisă o mare parte a aplicației. Modele alternative de dezvoltare pot să prindă aceste probleme în etapele conceptuale, atunci când se pot face schimbări mai ușoare și mai puțin costisitoare.

Modelul de code-and-fix este adecvată numai pentru proiecte mici, care nu sunt destinate să servească drept bază pentru dezvoltări viitoare.



FIG 6 – Schema Code-and-Fix

**Avantaje:**

* Util pentru proiecte mici
* Nu există surplus administrativ

**Dezavantaje:**

* Cost mare pentru depanare
* Nu este flexibil
* Nu există o planificare a resurselor
	1. **Modelul cu prototipuri**

Modelul cu prototipuri este o metodă de dezvoltare a sistemelor ( SDM ) în care un prototip (o aproximare rapidă a unui sistem sau produs final) este construit, testat și apoi refăcut după cum este necesar, până când este realizat un prototip îndeplineste cerințele pentru ca un sistem complet sau produsul pot fi dezvoltate.

Acest model funcționează cel mai bine în cazul în scenariile în care toate cerințele proiectului nu sunt cunoscute în detaliu. Acesta este un proces iterativ, un proces bazat pe încercări și erori, care are loc între dezvoltatorii și utilizatorii.



Fig 7 – Schema Prototip

**Pași necesari în realizarea unui astfel de model:**

1. Noile cerințe ale sistemul sunt definite în cât mai multe detalii posibile. Acest lucru implică, de obicei, constă în intervievarea unui număr de utilizatori care reprezintă toate departamentele sau aspecte ale sistemului existent .
2. Un design preliminar este creat pentru noul sistem.
3. Este creat de la proiectarea preliminară un prim prototip al noului sistem. Aceasta este, de obicei un sistem la scară redusă și reprezintă o aproximare a caracteristicilor produsului final.
4. Utilizatorii evaluază primul prototip, notând punctele slabe și bune ale prototipului, ceea ce trebuie să fie adăugat și ceea ce ar trebui să fie eliminat. Dezvoltatorul colectează și analizează comentariile utilizatorilor.
5. Primul prototip este modificat, pe baza comentariilor furnizate de către utilizatori și un al doilea prototip al noului sistem este construit.
6. Al doilea prototip este evaluată în același mod cum a fost și primul prototip.
7. Pașii anteriori sunt repetați de câte ori este necesar, până când utilizatorii sunt mulțumiți că prototipul reprezintă produsul final dorit.
8. Sistemul final este construit, bazat pe prototipului final.
9. Sistemul final este evaluat detaliat și testat. Întreținerea de rutină se efectuează în mod continuu pentru a preveni problemele pe scară largă și pentru a minimiza timpul în care sistemul nu funcționează.

**Tipuri de modele prototip:**

* Prototip aruncare/rapide
* Prototip incremental
* Prototip extrem
* Protopip evolutiv

**Prototipuri aruncare/rapide:**

 Acest tip de prototipuri foloseste eforturilor foarte mici ce necesită o analiză minimă pentru a construi un prototip. După cerințele actuale sunt înțelese, prototipul este îndepărtat, iar sistemul actual este dezvoltat cu o înțelegere mult mai clară a cerințelor utilizatorilor.

 **Prototip incremental:**

 Prototipuri incremental se referă la construirea de mai multe prototipuri funcționale ale diferitelor sub sisteme și apoi integrarea tuturor prototipuri disponibile pentru a forma un sistem complet.

 **Prototip extrem:**

Prototipuri de tip extrem este utilizat în domeniul de dezvoltare web.

Este compus din trei faze secvențiale. În primul rând, un prototip de bază, ce conține toate paginile existente, prezentate în format HTML. Apoi prelucrarea datelor este simulată cu ajutorul unui prototip a unui strat de servicii. În cele din urmă serviciile sunt puse în aplicare și integrate în prototipul final.

 **Prototip evolutiv:**

 Prototipuri evolutiv se bazează pe construirea de prototipuri funcționale reale cu funcționalitate minimă la început. Prototipul dezvoltat formează baza viitoarelor prototipuri pe care întregul sistem este construit. Utilizarea de prototipuri evolutive constă în utilizarea numai a cerințelor bine înțelese ce sunt incluse în prototip, iar cerințele noi sunt adăugate atunci când acestea sunt înțelese.

**Avantaje și Dezavantaje ale modelului cu prototipuri**

**Avantaje:**

* Participarea utilizatorilor în creeare produsul înainte de implementare
* Reduce timpul și costurile deoarece defectele pot fi detectate mult mai devreme.
* Lipsa diferitelor funcționalități pot fi identificate cu ușurință.
* Deoarece este afișat un model de lucru a sistemului, utilizatorii primesc o mai bună înțelegere a sistemului în curs de dezvoltare .

**Dezavantaje:**

* Practic , această metodologie poate crește complexitatea sistemului deoarece domeniul de aplicare a sistemului se poate extinde dincolo de planurile inițiale .
* Utilizatorii pot devenii confuzi în prototipurile și sistemele actuale .
* Efortul investit în construcția de prototipuri ar putea fi prea mult dacă nu este monitorizat în mod corespunzător
	1. **Modelul Haos**

Modelul haos este o abordare a procesului de dezvoltare software care utilizează idei luate din teoria haosului pentru a aborda problemele comune în timp ce se lucrează într-o echipă.

Se străduiește să unifice cele mai bune metodologii de programare cu cele mai bune tehnici de management de proiect astfel formând în mod ideal, o strategie superioară. Relație modelului haos la teoria haosului este ideea că problemele arhitecturale de mari dimensiuni nu pot fi stabilizate, fără stabilizarea probleme mici în aplicație, inclusiv liniile individuale de cod .

Modelul haos combină o buclă de rezolvare a problemelor liniare cu fractalii ca să sugereze că un proiect constă din mai multe nivele interdependente de rezolvare a problemelor . Comportamentul unui astfel de sistem complex reiese din comportamentul combinat al blocurilor mici de construcție.

Ciclul de viață a haosului definește fazele ale ciclului de viață de dezvoltare de software în fractali care arata ca toate fazele ciclului de viață au loc în alte faze.

În cadrul modelului haos există mai multe niveluri de cerințe. Acestea sunt:

- nivel de program

- nivelul componentelor

- nivel funcțional

- nivel de cod

Ideea de bază din spatele modelului este că de codul software este o integrare complexă de mii de module , funcții , și de linii de cod . Acest haos de integrare garantează o metodă care definește integrarea între întregul program și codul care definește acest program .

 Cele mai multe metodologii de dezvoltare software din ziua de astăzi se concentrează pe comunicare și de detaliul procesului de dezvoltare. Această abordare creează o transparență între dorințele de management la nivel înalt și înțelegerea echipei de dezvoltare a problemelor și a priorităților.

Regula principală din strategia haosului este că întotdeauna se rezolvă problema cea mai importantă în primul rând. Acest lucru reflectă ideologia de producție “Just-in –Time”.

 Modelul haos definește un nivel mai scăzut necesar de interpretare și încearcă să abordeze dezvoltarea de software de la un proces de rezolvare a problemelor liniar, care este fundamental în toate dezvoltarea de software.

Modelul Agil solicita clienților să acorde prioritate funcționalității de afaceri pentru punerea în aplicare . Modelul haos încearcă să rezolve cele mai importante probleme în primul rând de la programul de nivel superior pâna la cel mai mic nivel de generare a codului. Acest punct de vedere subliniază necesitatea critica de a include proiectarea nivelului de cod care trebuie să fie implementat pentru a îndeplini cerințele la nivel de program .

Acest model reprezintă pentru partea umanistă a unui efort de dezvoltare. Echipa de dezvoltare este alcătuită din persoane care trebuie să proiecteze și să configureze modulele din cadrul aplicației software. Fiecare membru al echipei trebuie să ia decizii critice în codul care ar putea avea impact în întregul program. Modelul haos ține cont de interacțiunea dintre membrii echipei atunci când se fac modificări în cod.

Bucla liniară pentru rezolvarea problemelor presupune că echipa de dezvoltare trebuie să fie comunice cu echipa pentru a asigura că tehnica corectă este pusă în aplicare. Această abordare va limita riscul de a găsii o soluție prea complicată. Dezvoltatorii de software care folosesc modelul haos dezvoltă produse o buclă liniară de rezolvare a problemelor și componente pentru a gestiona crearea de software complex.

1. Bibliografie

Notiuni generale + etape ale ciclului de viata

https://www.cert-ro.eu/files/doc/684\_20121030131030076061600\_X.pdf

https://en.wikipedia.org/wiki/Systems\_development\_life\_cycle

http://staff.cs.upt.ro/~dan/curs/fis/Cap2\_FazeCicluDeViata.pdf

Modelul in cascada

http://www.tutorialspoint.com/sdlc/sdlc\_waterfall\_model.htm

https://en.wikipedia.org/wiki/Waterfall\_model

Modelul in V

http://www.tutorialspoint.com/sdlc/sdlc\_v\_model.htm

https://en.wikipedia.org/wiki/V-Model\_(software\_development)

http://www.softwaretestingclass.com/v-model/

Incremental

<https://www.academia.edu/5576984/Cap_4_Ciclul_de_viata_al_produselor_program>

<https://en.wikipedia.org/wiki/Incremental_build_model>

http://istqbexamcertification.com/what-is-incremental-model-advantages-disadvantages-and-when-to-use-it/

Spirala

<https://www.academia.edu/5576984/Cap_4_Ciclul_de_viata_al_produselor_program>

<https://en.wikipedia.org/wiki/Spiral_model>

http://istqbexamcertification.com/what-is-spiral-model-advantages-disadvantages-and-when-to-use-it/

Agil

http://www.tutorialspoint.com/sdlc/sdlc\_agile\_model.htm

<https://en.wikipedia.org/wiki/Agile_software_development>

Code-and-fix

http://zone.ni.com/reference/en-XX/help/371361J-01/lvdevconcepts/lifecycle\_models/

https://courses.cs.washington.edu/courses/cse403/11wi/lectures/20110105%20Software%20Development%20Lifecycle.pdf

Prototipuri

http://searchcio.techtarget.com/definition/Prototyping-Model

<http://www.tutorialspoint.com/sdlc/sdlc_software_prototyping.htm>

Haos:

<https://timross.wordpress.com/2010/01/17/software-development-and-chaos-theory/>

<http://www.wisegeek.com/what-is-the-chaos-model.htm>

http://www.computerhope.com/jargon/c/chaos-model.htm

**Anexa**

Fig. 1 - http://www.scritub.com/files/informatica/1158\_poze/image002.jpg

Fig.2 - http://www.scritub.com/files/informatica/1158\_poze/image004.jpg

Fig. 3 - http://istqbexamcertification.com/what-is-incremental-model-advantages-disadvantages-and-when-to-use-it/

Fig. 4 - https://en.wikipedia.org/wiki/Spiral\_model

Fig. 5 - https://www.sdlc.ws/waterfall-model/

Fig. 6 - https://www.google.ro/search?q=code+and+fix+model&espv=2&biw=1920&bih=955&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjM9IDl2snKAhVo8HIKHZbvDbMQ\_AUIBigB#imgrc=sEyx2oEjHqCfFM%3A

Fig. 7 - http://istqbexamcertification.com/wp-content/uploads/2012/01/Prototype-model.jpg