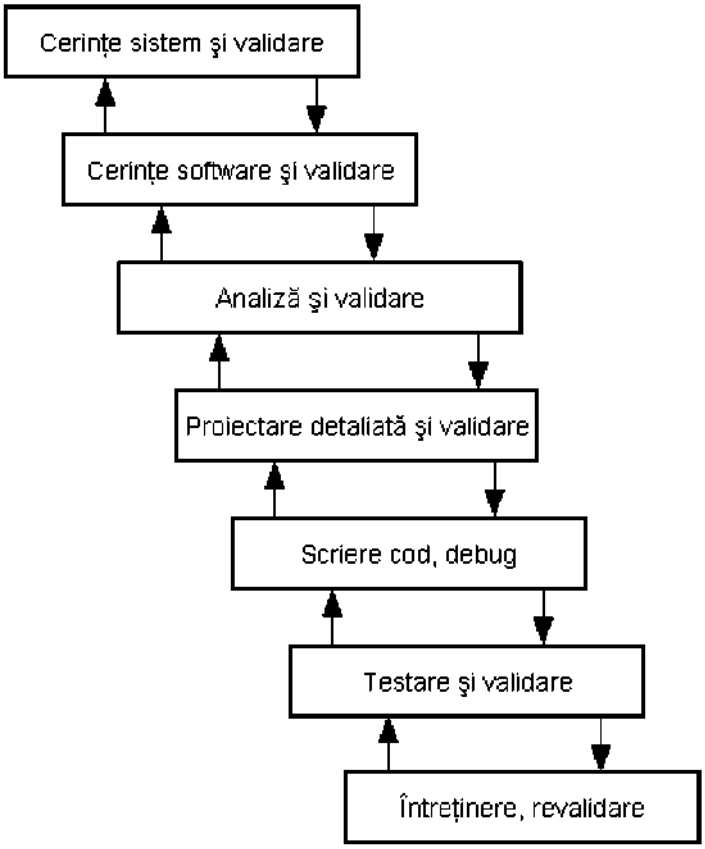
Modele de dezvoltare software.

1. Modelul cascadă

Modelul cascadă, propus de Barry Boehm, este unul din cele mai cunoscute exemple de modele din ingineria programării. Există numeroase variante ale acestui model. Într-o variantă detaliată, modelul cascadă cuprinde următoarele etape:



După fiecare etapă există un pas de validare. Procesul „curge” de la etapă la etapă, ca apa într-o cascadă. În descrierea originară a lui Boehm, există o întoarcere, un pas înapoi interactiv între fiecare două etape. Astfel, metoda cascadă este de fapt o combinație de metodologie secvențială cu elemente ciclice. Totuși, în practica inginerească, termenul „cascadă” este utilizat ca un nume generic pentru orice metodologie secvențială.

Acesta este modelul după care de obicei sistemele sunt dezvoltate în practică. De asemenea, reordonarea fazelor s-a dovedit a fi sub-optimală. Există o mare atracție pentru acest model datorită experienței, tradiției în aplicarea sa și succesului pe care l-a implicat. O sarcină complexă este împărțită în mai mulți pași mici, ce sunt mai ușor de administrat. Fiecare pas are ca rezultat un produs bine definit (documente de specificație, model, etc.)

Modelul cascadă cu feedback propune remedierea problemelor descoperite în pasul precedent. Problemele la pasul *i* care sunt descoperite la pasul *i* + 3 rămân neremediabile. Un model realist ar trebui să ofere posibilitatea ca de la un anumit nivel să se poată reveni la oricare dintre nivelele anterioare.

Dezavantajul principal al modelului în cascadă apare deoarece clientul obține o viziune practică asupra produsului doar în momentul terminării procesului de dezvoltare. De asemenea, modelul nu are suficientă putere descriptivă, în sensul că nu integrează activități ca managementul resurselor sau managementul configurației. Aceasta face dificilă coordonarea proiectului.

După cum am menționat la prezentarea metodologiei generice secvențiale, și modelul cascadă impune înghețarea specificațiilor foarte devreme în procesul de dezvoltare pentru a evita iterațiile frecvente (reîntoarcerile în fazele anterioare atunci când în faza curentă s-au detectat erori: în timpul analizei se descoperă erori de specificații, în timpul implementării se descoperă erori de specificații/proiectare etc., astfel încât procesul poate implica multiple secvențe de iterații ale activităților de dezvoltare). Înghețarea prematură a cerințelor conduce la obținerea unui produs prost structurat și care nu execută ceea ce dorește utilizatorul. Conduce de asemenea la obținerea unei documentații neadecvate deoarece schimbările intervenite în iterațiile frecvente nu sunt actualizate în toate documentele produse.

2. Modelul spirală

Modelul spirală, propusă tot de Boehm, este un alt exemplu bine cunoscut de model a ingineriei programării. Acest model încearcă să rezolve problemele modelului în cascadă, păstrând avantajele acestuia: planificare, faze bine definite, produse intermediare. El definește următorii pași în dezvoltarea unui produs:

- studiul de fezabilitate;
- analiza cerințelor;
- proiectarea arhitecturii software;
- implementarea.

Modelul în spirală recunoaște că problema principală a dezvoltării programelor este riscul. Riscul nu mai este eliminat prin aserțiuni de genul: „în urma proiectării am obținut un model corect al sistemului", ca în modelul cascadă. Aici riscul este acceptat, evaluat și se iau măsuri pentru contracararea efectelor sale negative. Exemple de riscuri:

- în timpul unui proces îndelungat de dezvoltare, cerințele noi ale clientului sunt ignorate;
- clientul schimbă cerințele;
- o firmă concurentă lansează un program rival pe piață;
- un dezvoltator/arhitect părăsește echipa de dezvoltare;
- o echipă nu respectă un termen de livrare pentru o anumită componentă.

În modelul spirală se consideră că fiecare pas din dezvoltare conține o serie de activități comune:

- pregătirea: se identifică obiectivele, alternativele, constrângerile;
- gestionarea riscului: analiza și rezolvarea situațiilor de risc;
- activități de dezvoltare specifice pasului curent (de exemplu analiza specificațiilor sau scrierea de cod);
- planificarea următorului stadiu: termenele limită, resurse umane, revizuirea stării proiectului.

Metodologia spirală cuprinde următoarele etape, grupate pe patru cicluri:

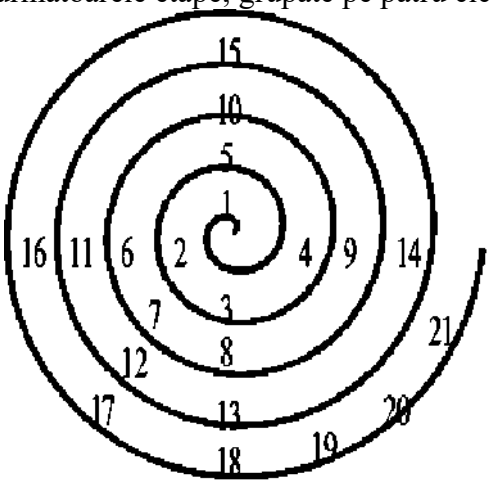


Fig. 2.2.1. Metodologia spirală

Ciclul 1 – Analiza preliminară:

1) Obiective, alternative, constrângeri
2) Analiza riscului și prototipul
3) Conceperea operațiilor
4) Cerințele și planul ciclului de viață
5) Obiective, alternative, constrângeri

Analiza riscului și prototipul

Ciclul 2 – Analiza finală:

6) Simulare, modele, benchmark-uri
7) Cerințe software și validare
8) Plan de dezvoltare
9) Obiective, alternative, constrângeri
10) Analiza riscului și prototipul

Ciclul 3 – Proiectarea:

11) Simulare, modele, benchmark-uri
12) Proiectarea produsului software, validare și verificare
13) Integrare și plan de test
14) Obiective, alternative, constrângeri
15) Analiza riscului și prototipul operațional

Ciclul 4 – Implementarea și testarea:

16) Simulare, modele, benchmark-uri
17) Proiectare detaliată
18) Cod
19) Integrarea unităților și testarea acceptării
20) Lansarea produsului

Procesul începe în centrul spiralei. Fiecare ciclu terminat reprezintă o etapă. Pe măsură ce spirala este parcursă, produsul se maturizează. Cu fiecare ciclu, sistemul se apropie de soluția finală. Deși este considerată ca un exemplu generic pentru metodologia ciclică, metoda are și elemente secvențiale, puse în evidență de evoluția constantă de la o etapă la alta.

3. Modelul V

Numele Modelului V înseamnă Verificare și Validare și corespunde metodologiei secvențiale de dezvoltare a softului.

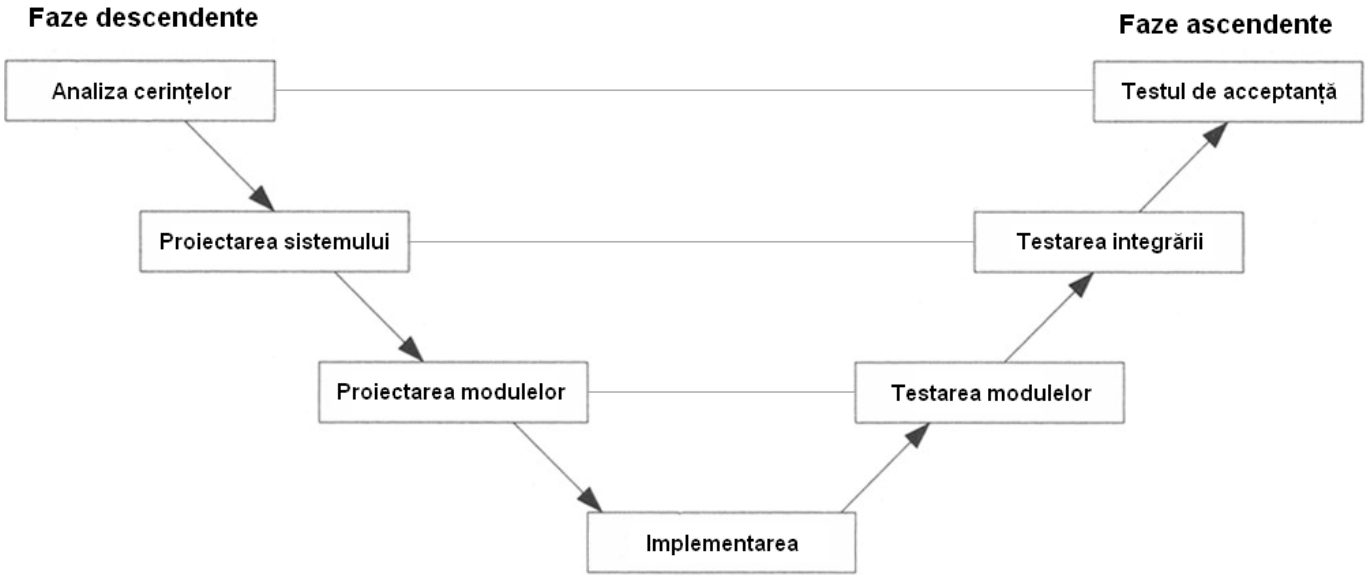
Dezvoltat pentru reglementarea dezvoltării de software în administraţia federală germană.

Evidenţiază testarea pe tot parcursul ciclului de dezvoltare.

Trecerea la faza următoare se face numai după ce toate produsele din faza curentă au trecut testele de verificare şi validare.

Există mai multe variante ale acestui model.

Procesul de verificare şi validare are scopul detectării cât mai multor erori în ciclul de dezvoltare



Procesele de verificare permit legătura inversă cu procesele de bază, care corespund modelului cascadă, astfel la sfârşitul fiecărei etape a ciclului de dezvoltare,uneori şi pe parcursul desfăşurării etapei, se face verificarea corectitudinii cerinţelor pentru diferite nivele de dezvoltare.

Are aceleași dezavantaje ca și modelul cascadă. Ar trebui folosit pentru proiecte mici și medii ca durată de dezvoltare unde cerințele sunt clare și definitive.

4. Prototipizarea

O problemă generală care apare la dezvoltarea unui program este să ne asigurăm că utilizatorul obţine exact ceea ce vrea. Prototipizarea vine în sprijinul rezolvării acestei probleme. Încă din primele faze ale dezvoltării, clientului i se prezintă o versiune funcţională a sistemului. Această versiune nu reprezintă întregul sistem, însă este o parte a sistemului care cel puţin funcţionează.

Prototipul ajută clientul în a-şi defini mai bine cerinţele şi priorităţile. Prin intermediul unui prototip, el poate înţelege ce este posibil şi ce nu din punct de vedere tehnologic. Prototipul este de obicei produs cât mai repede; pe cale de consecinţă, stilul de programare este de obicei (cel puţin) neglijent. Însă scopul principal al prototipului este de a ajuta în fazele de analiză şi proiectare şi nu folosirea unui stil elegant.

Se disting două feluri de prototipuri:

- de aruncat (throw-away);
- evoluţionar.

În cazul realizării unui prototip de aruncat, scopul este exclusiv obţinerea unei specificaţii. De aceea nu se acordă nici o importanţă stilului de programare şi de lucru, punându-se accent pe viteza de dezvoltare. Odată stabilite cerinţele, codul prototipului este „aruncat", sistemul final fiind rescris de la început, chiar în alt limbaj de programare.

În cazul realizării unui prototip evoluţionar, scopul este de a crea un schelet al aplicaţiei care să poată implementa în primă fază o parte a cerinţelor sistemului. Pe măsură ce aplicaţia este dezvoltată, noi caracteristici sunt adăugate scheletului existent. În contrast cu prototipul de aruncat, aici se investeşte un efort considerabil într-un design modular şi extensibil, precum şi în adoptarea unui stil elegant de programare.

Această metodă are următoarele avantaje:

- permite dezvoltătorilor să elimine lipsa de claritate a specificaţiilor;
- oferă utilizatorilor şansa de a schimba specificaţiile într-un mod ce nu afectează drastic durata de dezvoltare;
- întreţinerea este redusă, deoarece validarea se face pe parcursul dezvoltării; se poate facilita instruirea utilizatorilor finali înainte de terminarea produsului.

Dintre dezavantajele principale ale prototipizării amintim:

- deoarece prototipul rulează într-un mediu artificial, anumite dezavantaje ale produsului final pot fi scăpate din vedere de clienţi;
- clientul nu înţelege de ce produsul necesită timp suplimentar pentru dezvoltare, având în vedere că prototipul a fost realizat atât de repede;
- deoarece au în fiecare moment şansa de a face acest lucru, clienţii schimbă foarte des specificaţiile;
- poate fi nepopulară printre dezvoltători, deoarece implică renunţarea la propria muncă.