Ciclul de viață a produselor software (SDLC).

În timpul dezvoltării produselor software s-a constatat că există anumite tipuri de activități care trebuie făcute la un moment dat:

- *Analiza cerințelor*
- *Proiectarea arhitecturală*
- *Proiectarea detaliată*
- *Implementarea (Scrierea codului)*
- *Integrarea componentelor*
- *Verificarea și Validarea (Testarea)*
- *Mentenanța (Întreținerea)*

Aceste activități sunt în strânsă legătură cu cele patru faze ale ingineriei programării: analiza, proiectarea, implementarea și testarea.

Analiza cerințelor software.

Așa cum arată statisticile principala cauză a eșecului proiectelor software este insuficienta implicare a clienților. Buna înțelegerea a problemelor clientului, neapărat legată de împărtășirea unei viziuni comune între echipa de dezvoltare și client sunt două fațete de bază ale implicării clientului. Fără ele, echipa de dezvoltare nu are referințele de bază pentru a putea ști dacă ceea ce dezvoltă este corespunzător sau nu și, prin urmare, implicarea puternică a clientul în proiect și înțelegerea comună a problemelor lui este vitală. Altfel, este ca și cum te-ai deplasa într-o direcție oarecare, fără să știi unde vrei să ajungi. Mesajul acesta este adesea (și cel mai bine!) ilustrat printr-o caricatură despre diferența care apare adesea în practică, între problema clientului, înțelegerea problemei și ceea ce se realizează de fapt:

Ce este Analiza cerințelor software?

Analiza cerințelor software (pe care o vom numi în continuare Analiza Software) este aceea dintre disciplinele existente în domeniul Software care determină ce trebuie făcut, preluând problema clientului și exprimând-o într-un inteligibil de către dezvoltator.

Mai detaliat, Analiza Software este aceea dintre disciplinele implicate în procesul de dezvoltare a produsului software ale cărei obiective sunt:

- înțelegerea problemelor curente ale organizației client și a rațiunii pentru care este dispus să investească într-un produs software;
- asigurarea unei viziuni comune asupra problemelor și a viitoarei soluții pentru toți participanții în proiect;
- culegerea și actualizarea cerințelor pentru viitorul produs software și traducerea cerințelor clientului în cerințe software pe care echipa de dezvoltare să le poată înțelege și transforma în funcționalități efective;
- definirea limitelor viitorului sistem (acesta este un element extrem de important în economia proiectului, având în vedere că în proiectele reale există întotdeauna o presiune crescândă în direcția extinderii acestor limite).

Activitatea de Analiză este, în același timp, aceea care furnizează elementele pentru negocierea și renegocierea bugetului și pentru planificarea resurselor. Ea conturează produsul și, prin urmare, proiectul. Impactul Analizei asupra costurilor proiectului și, în general, asupra succesului acestuia este, așa cum am prezentat deja, foarte mare.

Pentru a fi foarte concret, în activitatea unui Analist Software intră următoarele mari tipuri de activități:

- interviuri la client;
- analiza propriu-zisă a cerințelor;
- dezvoltare cerințe (specificație);

- managementul cerințelor.

De ce este nevoie de Analiză Software?

The Standish Group arăta că primele trei cauze ale problemelor privind calitatea și livrarea la timp, în proiectele software, sunt:

- insuficient input (contribuție) de la utilizatori;
- cerințe și specificații incomplete;
- schimbarea frecventă a cerințelor.

Ori, toate acestea sunt lucruri de care se ocupă, în primul rând, Analiza Software. De asemenea, trebuie să fim conștienți că schimbarea frecventă a cerințelor, de exemplu, nu este lucru accidental, care într-un proiecte se întâmplă, în altul nu – dimpotrivă, este un lucru care se întâmplă întotdeauna. Prin urmare, nici nu se pune problema ca într-un proiect software să nu îți propui să controlezi un factor atât de important.

De asemenea, Analiza, prin output-urile sale (adică Specificațiile software) răspunde la una dintre nevoile fundamentale ale proiectelor: nevoia de comunicare a cerințelor între membrii echipei de proiect și de formalizare a acestora.

Proiectarea

Pe baza cerințelor din faza de analiză, acum se stabilește *arhitectura* sistemului: componentele sistemului, interfețele și modul lor de comportare:

- *Componentele* sunt elementele constructive ale produsului. Acestea pot fi create de la zero sau reutilizate dintr-o bibliotecă de componente. Componentele rafinează și capturează semnificația detaliilor din documentul cerințelor;
- *Interfețele* ajută la îmbinarea componentelor. O interfață reprezintă granița dintre două componente, utilizată pentru comunicarea dintre acestea. Prin intermediul interfeței, componentele pot interacționa;
- *Comportamentul*, determinat de interfață, reprezintă răspunsul unei componente la stimulii acțiunilor altor componente.

Documentul de proiectare descrie planul de implementare a cerințelor. Se identifică detaliile privind limbajele de programare, mediile de dezvoltare, dimensiunea memoriei, platforma, algoritmii, structurile de date, interfețele etc.

În această fază trebuie indicate și *prioritățile critice* pentru implementare. Acestea sugerează sarcinile care, dacă nu sunt executate corect, conduc la eșecul sistemului. Totuși, chiar dacă prioritățile critice sunt îndeplinite, acest fapt nu duce automat la succesul sistemului, însă crește nivelul de încredere că produsul va fi o reușită.

Folosind scenariile tipice și atipice, trebuie realizate compromisurile inerente între performanță și complexitatea implementării. *Analiza performanțelor* presupune studierea modului în care diferitele arhitecturi conduc la diferite caracteristici de performanță pentru fiecare scenariu tipic. În funcție de frecvența de utilizare a scenariilor, fiecare arhitectură va avea avantaje și dezavantaje. Un răspuns rapid la o acțiunea a utilizatorului se realizează deseori pe baza unor costuri de resurse suplimentare: indecși, managementul cache-ului, calcule predictive etc. Dacă o acțiune este foarte frecventă, ea trebuie realizată corect și eficient. O acțiune mai rară trebuie de asemenea implementată corect, dar nu este evident care e nivelul de performanță necesar în acest caz. O situație în care o astfel de acțiune trebuie implementată cu performanțe maxime este închiderea de urgență a unui reactor nuclear.

Planul de implementare și planul de test, descrise mai jos, pot fi incluse de asemenea în fazele de implementare și respectiv testare. Însă unul din scopurile fazei de proiectare este stabilirea unui plan pentru terminarea sistemului, de aceea cele două planuri au fost incluse în acest paragraf.

Planul de implementare stabilește programul după care se va realiza implementarea și resursele necesare (mediul de dezvoltare, editoarele, compilatoarele etc.).

Planul de test definește testele necesare pentru stabilirea calității sistemului. Dacă produsul trece toate testele din planul de test, este declarat terminat. Cu cât testele sunt mai amănunțite, cu atât este mai

mare încrederea în sistem și deci crește calitatea sa. Un anume test va verifica doar o porțiune a sistemului. *Acoperirea testului* este procentajul din produs verificat prin testare. În mod ideal, o acoperire de 100% ar fi excelentă, însă este rareori îndeplinită. De obicei, un test cu o acoperire de 90% este simplă, însă ultimele 10% necesită o perioadă de timp semnificativă.

De exemplu, să considerăm BIOS-ul (Basic Input/Output System) construit de IBM la începutul anilor ’80 în strânsă legătură cu sistemul de operare DOS (Disk Operating System) al firmei Microsoft. Din rațiuni de performanță, BIOS-ul a fost plasat într-un chip ROM, și deci patch-urile pentru eventualele erori erau imposibile. Astfel, au fost necesare teste cu acoperire de 100%. Codul propriu-zis al BIOS-ului era destul de mic, câteva mii de linii. Însă deoarece BIOS-ul are o natură asincronă, testul a presupus mai întâi crearea unui mediu asincron care să aducă sistemul în starea dorită și apoi trebuia generat un eveniment care să declanșeze un test. Foarte repede, setul de test a devenit mult mai mare decât BIOS-ul. A apărut astfel problema testării însuși a mediului de test! În final, o acoperire de 100% a fost atinsă, dar cu un cost foarte ridicat. O soluție mai ieftină a fost înscrierea BIOS-ului într-o combinație dintre EPROM (Electronic Programmable Read Only Memory) și ROM. Cea mai mare parte a produsului era plasat în ROM, iar patch-urile erau plasate în EPROM. Aceasta a fost abordarea adoptată de Apple pentru Macintosh.

În general, este suficient ca testele să cuprindă scenariile tipice și atipice, fără să verifice întregul sistem, cu absolut toate firele de execuție. Acesta poate conține ramificații interne, erori sau întreruperi care conduc la fire de execuție netestate. Majoritatea sistemelor sunt pline de bug-uri nedescoperite. De obicei, clientul participă în mod logic la testarea sistemului și semnalează erori care vor fi îndepărtate în versiunile ulterioare.

Implementarea

În această fază, sistemul este construit, ori plecând de la zero, ori prin asamblarea unor componente pre-existente. Pe baza documentelor din fazele anterioare, echipa de dezvoltare ar trebui să știe exact ce trebuie să construiască, chiar dacă rămâne loc pentru inovații și flexibilitate. De exemplu, o componentă poate fi proiectată mai restrâns, special pentru un anumit sistem, sau mai general, pentru a satisface o direcție de reutilizare.

Echipa trebuie să gestioneze problemele legate de calitate, performanță, biblioteci și debug. Scopul este producerea sistemului propriu-zis. O problemă importantă este *îndepărtarea erorilor critice*. Într-un sistem există trei tipuri de erori:

- *Erori critice:* Împiedică sistemul să satisfacă în mod complet scenariile de utilizare. Aceste erori trebuie corectate înainte ca sistemul să fie predat clientului și chiar înainte ca procesul de dezvoltare ulterioară a produsului să poată continua;
- *Erori necritice:* Sunt cunoscute, dar prezența lor nu afectează în mod semnificativ calitatea observată a sistemului. De obicei aceste erori sunt listate în notele de lansare și au modalități de ocolire bine cunoscute;
- *Erori necunoscute:* Există întotdeauna o probabilitate mare ca sistemul să conțină un număr de erori nedescoperite încă. Efectele acestor erori sunt necunoscute. Unele se pot dovedi critice, altele pot fi rezolvate cu patch-uri sau eliminate în versiuni ulterioare.

Testarea (Verificarea și Validarea)

Calitatea produsului software este foarte importantă. Multe companii nu au învățat însă acest lucru și produc sisteme cu funcționalitate extinsă, dar cu o calitate scăzută. Totuși, e mai simplu să-i explici clientului de ce lipsește o anumită funcție decât să-i explici de ce produsul nu este performant. Un client satisfăcut de calitatea produsului va rămâne loial firmei și va aștepta noile funcții în versiunile următoare.

În multe metodologii ale ingineriei programării, faza de testare este o fază separată, realizată de o echipă *diferită* după ce implementarea s-a terminat. Motivul este faptul că un programator nu-și poate descoperi foarte ușor propriile greșeli. O persoană nouă care privește codul poate descoperi greșeli evidente care scapă celui care citește și recitește materialul de multe ori. Din păcate, această practică poate determina o atitudine indiferentă față de calitate în echipa de implementare.

Tehnicile de testare sunt abordate preponderent din perspectiva producătorului sistemului. În mod ideal, și clientul trebuie să joace un rol important în această fază.

Testarea este un proces complex care poate fi împărțit în două faze:

Verificarea: În procesul de verificare ne asigurăm că programul este stabil și că funcționează corect din punctul de vedere al dezvoltatorilor. Întrebarea la care răspundem este: produsul este construt corect?

Validarea: În procesul de validare ne asigurăm că programul îndeplinește cerințele utilizatorului. Un exemplu de validare este testul de acceptare, în care produsul este prezentat clientului. Clientul spune dacă este mulțumit cu produsul sau dacă mai trebuie efectuate modificări;

Mentenanța (Întreținerea)

Întreținerea: După ce programul este livrat clientului, mai devreme sau mai târziu sunt descoperite defecte sau erori ce trebuie reparate. De asemenea, pot apărea schimbări în specificațiile utilizatorilor, care vor diverse îmbunătățiri. Întreținerea constă în gestionarea acestor probleme.