CAPITOLUL I

Goals, Roles and Myths of Software Engineering

• Deosebirea fundamentală dintre o lucrare de laborator sau un proiectul și un sistem software diferă prin: - dimensiune
- persistența în timp (lucrările de laborator sau proiectele au o durată mică de viață, 2-3 săpt. până sunt terminate, în schimb un proiect software de dvs. trebuie să persiste zeci de ani
- la sistemele software nu ai o cerință clară, este mai mult un proces de descoperire

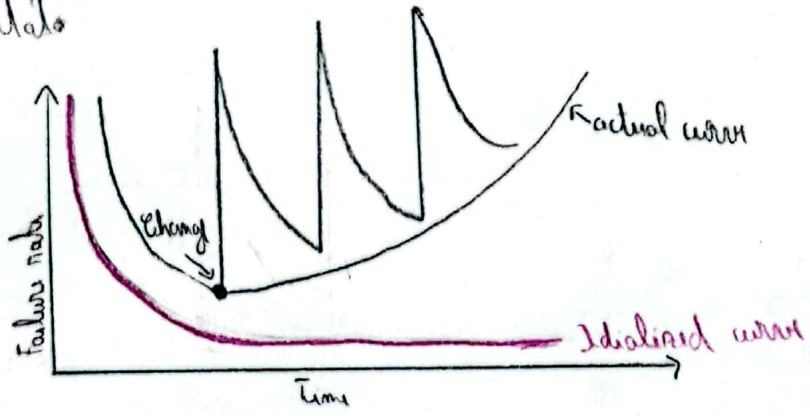
• Diferențe între ingineria software și restul tipurilor de inginerie:
- în ingineria software modelezi exclusiv informații ci nu materie ca în celelalte forme
- în celelalte tipuri de inginerii, descrierea lucrării cu materie primă există partea conceptuală iar aceasta trebuie implementată efectiv. În ingineria software partea a doua lipsește cu totul

Întrebări:

o Schimbările într-un sistem software apar aproape exclusiv datorită bug-urilor și greșelilor de proiectare. Dacă acestea ar fi evitate curba evoluției în timp a erorilor (failure rate) ar tinde asimptotic la zero

Schimbările într-un sistem software apar din nevoia de a întreține și a evolua programele, intervin noi cerințe de implementat, pentru a face sistemul să interacționeze cu un alt sistem modernizat și pt. a satisface nevoile noilor medii de calcul ⇒ Afirmația este fundamental FALSĂ

Software-ul nu se uzează în timp dar se deteriorează și devine tot mai greu de întreținut, de înțeles din această cauză a schimbării sistemului este acest fenomen de schimbare repetată



○ Testarea ocupă circa 25% din costul total al dezvoltării unui sistem software.

Afirmația este fundamental FALSĂ deoarece conform afirmațiilor lui Brooks privind costurile pentru realizarea unui produs software, testarea ocupă 50% din costul total fiind alcătuită din testare unitară (25%) și testare de modul (25%)

○ În mod normal între un sistem software timpul și costurile alocate testării sunt egale cu timpul și costurile alocate scrierii codului propriu zis.

Afirmația este fundamental FALSĂ deoarece timpul și costurile alocate testării potrivit lui Brooks sunt de 50% iar timpul și costurile pentru scrierea de cod propriu zis reprezintă 16%.

○ Curba reală a evoluției în timp a erorilor (Failure Rate) este în principal influențată de numărul de programatori adăugați în timpul dezvoltării sistemului

Afirmația este fundamental FALSĂ deoarece Failure Rate este în principal influențată de deteriorarea și tot mai greu înțelegerea software-ului acesta devenind la schimbarea repetată a sistemului, iar după fiecare schimbare Failure Rate crescând

○ Costul defectelor poate fi făcută să tindă în timp către o dacă cerințele sunt bine înțelese de la început și se aplică sistematic metode de testare eficiente.

Afirmația este fundamental FALSĂ deoarece Failure Rate de-a lungul schimbărilor repetate ale sistemului tinde să crească.

○ Legea lui Brooks, se referă doar la sistemele cu modularitate mai slabă întrucât doar în cazul acestui sistemul nu este o activitate perfect partiționabilă.

Afirmația este fundamental FALSĂ. Legea lui Brooks afirmă că adăugarea de noi programatori (ingineri software) la un proiect în întârziere va face ca proiectul să întârzie și mai mult. Motivul este acela că proiectele software nu sunt o activitate perfect partiționabilă deoarece implică foarte multe interacțiuni, iar adăugarea unor noi programatori implică și o repartizare (uneori imposibil de realizat) a task-urilor din proiect. În plus fiecare om adus în plus aduce după sine relații de comunicare mai complexe. În sistemele cu modularitate foarte bună, crește gradul de partiționare, și deci capacitatea de

a taskurilor între membrii echipei.

Totuși, Legea lui Brooks rămâne valabilă atât pt. sistemele cu modularitate foarte bună cât și pentru cele cu modularitate mai slabă, din 2 motive:

i) surplusul de comunicare este semnificativ indiferent de modularitate

ii) oricât de modular ar fi proiectat un sistem totuși nu se poate ca sistemul să devină unul perfect partiționabil

□ Întrebări Marinescu

- Ce face diferența dintre curba reală și cea ideală?

Diferența este făcută de schimbarea repetată a sistemului.

- Ce spun Legile evoluției softwarelui?

Legile evoluției softwarelui ale lui Lehman ne spun că sistemul ar trebui să evoluieze pentru a rămâne relevant, dar dacă crești prea adesea mai multă schimbări și astfel tinde să îl degradeze, de aia trebuie să facem operații pentru al întreține

○ Legile evoluției software-ului ale lui Lehman nu se aplică la sistemele ale căror cerințe sunt perfect înțelese de la început, pentru că astfel de sisteme nu au nevoie să evolueze.

Afirmația este fundamental FALSĂ. Conform legilor lui Lehman sistemul trebuie să evoluieze pentru a rămâne relevant, deoarece în timp acesta nu va mai corespunde cerințelor și din această cauză are nevoie de întreținere.

○ Legile evoluției software-ului ale lui Lehman avertizează că un sistem care se va modifica în permanență pentru a se adapta noilor cerințe tinde să devină imposibil de întreținut.

Afirmația este fundamental FALSĂ, deoarece sistemele trebuie adaptate continuu, altfel devin din ce în ce mai puțin satisfăcătoare, conținutul funcțional al sistemelor trebuie să crească continuu, pentru a menține satisfacția utilizatorilor de-a lungul timpului. Pe măsură ce un sistem evoluează complexitatea sa crește dacă munca nu e efectuată pentru a-l menține sau a-l reduce. Calitatea sistemelor va părea că declină, numai dacă nu sunt menținute riguros și adaptate la schimbările operaționale de mediu. Rata medie efectivă a activității globale într-un sistem care evoluează este de-a lungul duratei de viață a produsului.