Subiect 1

Modelul cascada vs Modelul spirala vs Modelul V vs Agile etc

a. Avantaje si dezavantaje?

Modelul cascada:

Avantaje:

- Sistem bine documentat
- Permite controlul efficient al proiectelor si estimarea exacta a timpilor de executie
- bun management al proiectului

Dezavantaje:

- un produs executabil care sa demonstreze functionalitatea este disponibil destul de tarziu, dupa integrare
- multe erori sunt descoperite tarziu => cost crescut
- toate riscurile sunt incluse intr-un singur ciclu de dezvoltare

Modelul spirala:

Avantaje:

- abordare evolutionista
- ajuta la intelegerea riscurilor si la identificarea modalitatilor de tinere sub control a acestora
- prototipizarea este folosita ca mecanism de reducere a riscurilor
- functionalitati aditionale pot fi adaugate mai tarziu
- software-ul este produs devreme in ciclul de dezvoltare

Dezavantaje:

- Analiza riscurilor este o activitate critica
- Poate sa fie costisitor
- Succesul proiectului este dependent de pasul de analiza a riscurilor
- Nu functioneaza foarte bine pentru proiectele mai mici

Modelul V:

Avantaje:

- Este simplu si usor de folosit
- Activitatile de testare se petrec inainte de inceperea scrierii codului
- Functioneaza foarte bine pentru proiecte mici, ale caror cerinte sunt usor de inteles

Dezavantaje:

- Este un model foarte rigid si cel mai putin flexibil
- Software-ul este dezvoltat in timpul pasului de implementare, deci nu avem prototipuri ale softului pe care urmeaza sa-l producem
- Daca se petrec schimbari la jumatatea proiectului, atunci documentele de testare si de cerinte trebuie updatate

Prototipuri:

Avantaje:

- Necesar redus de documentatie
- Costuri reduse de intretinere
- Clientii pot fi mult mai implicati in deciziile legate de designul final al sistemului
- Comunicare imbunatatita

Dezavantaje:

- O aplicatie incompleta poate sa duca la utilizarea acesteia diferit fata de cum a fost proiectata
- Complexitatea sistemului poate sa creasca fata de planurile originale

Agile:

Avantaje:

- Satisfacerea clientului prin livrarea continua de soft
- Soft functional este livrat frecvent
- Metode de comunicare face-to-face
- Adaptabilitate crescuta la schimbari
- Chiar si schimbarile tarzii sunt acceptate

Dezavantaje:

- Nu se pune accent pe design si documentatie
- Doar programatorii seniori sunt capabili sa ia decizii necesare in timpul procesului de dezvoltare
- In cazul unor livrabile, in special cele mari, este greu sa evaluezi efortul necesar la inceputul procesului de dezvoltare

b. La ce gen de proiecte le-ati recomanda ?

Modelul Cascada : Adecvat pentru proiecte in care cerintele sunt bine intelese de la inceput si nu se modifica pe parcursul procesului de dezvoltare. De obicei proiectele bazate pe modelul cascada sunt scurte.
Modelul Spirala: Adecvat pentru proiectele de risc mediu spre ridicat, unde costurile si evaluarea riscurilor sunt importante, unde userii sunt nesiguri de ceea ce doresc, cerintele sunt complexe si sunt asteptate schimbari insemnate.
Modelul in V : Adecvat pentru proiecte mici-medii, unde cerintele sunt clar definite si fixate. Ar trebui ales cand sunt disponibile ample resurse tehnice, cu efectuarea unei expertize tehnice.
Prototipuri : Se folosesc atunci cand trebuie ca sistemul dorit sa aiba multa interactiune cu end-userii. Recomandat pentru sisteme online, interfete web etc.
Agile : Adecvat pentru proiecte in care apar noi schimbari care trebuie implementate. Libertatea de schimbare oferita de „agile” este foarte importanta pentru ca noile schimbari pot fi implementate cu un cost scazut, datorita frecventei noilor incremente produse.

c. La ce gen de echipe le-ati recomanda ?
Cascada: Adecvat pentru echipe in care pot aparea membri noi intrucat documentatia si proiectarea structurii reprezinta un avantaj pentru acestia. Deasemenea este adecvant pentru echipe in care membrii apar si se implica la diferite faze ale dezvoltarii in comparatie

cu Agile cand toti membrii echipei sunt necesari inca de la inceput pentru fiecare iteratie. De exemplu testerul poate fi implicat abia la sfarsitul procesului de dezvoltare.
If you have several project teams located in different geographic locations, co-ordination of work needs to be more detailed and stringent. Work assignments need to be well-defined to avoid confusion and redundancy of work. In such cases, Waterfall is likely more beneficial as it provides clear-cut deliverables and milestones.
Spirala: Adecvat mai mult pentru seniors si architects in prima faza pana la evaluarea obiectivelor, riscurilor si a concretizarii taskurilor. Apoi este adecvat pentru orice tip de echipa dupa stabilirea riscurilor si a obiectivelor. Intrucat lucrurile sunt clare/risk-free si cateodata chiar si atractive intru cat in acest model se asteapta schimbari semnificative dar care sunt risk-free.
V: Adecvat echipelor cu programatori disciplinati si avansati intrucat foloseste mecanisme de TDD care pot deveni foarte costly in timp fara experienta.
Prototipuri: Adecvat echipelor care lucreaza foarte mult cu front-end(web-designers) sau alte tipuri de interfatare directa cu clientul.
Agile: Applying the agile approach on geographically separate teams may introduce new challenges. As noted by Martin Fowler, a well-known agile evangelist, Because agile development works best with close communication and an open culture, agilists working offshore feel the pain much more than those using plan-driven approaches.

Subiect 2

Utilizand o lista simplu inlantuita scrieti o specificatie tehnica pentru detectia existentei unui ciclu in lista (este lista finita sau infinita/ciclica de la un punct incolo?).

Nu este permisa alocarea de structuri de date suplimentare sau a marcarii/alterarii unora din elementele listei.

The idea is to have two references to the list and move them at **different speeds**. Move one forward by 1 node and the other by 2 nodes.

- If the linked list has a loop they will *definitely* meet.
- Else either of the two references(or their next) will become null.

Java function implementing the algorithm:

```
boolean hasLoop(Node first) {
   Node slow = first;
   Node fast = first;

   while(fast != null && fast.next != null) {
      slow = slow.next;          // 1 hop
      fast = fast.next.next;     // 2 hops

      if(slow == fast)  // fast caught up to slow, so there is a loop
         return true;
   }
   return false;  // fast reached null, so the list terminates
}
```

Subiect 3

Se doreste alocarea unui numar de persoane pe o serie de posturi disponibile, dupa cum urmeaza: team leader(i), developer(i), tester(i), researcher(i). Ordonati in ordinea descrescatoare a importantei urmatoarele calitati, pentru fiecare post in parte. Motivati alegerea si puneti la punct un algoritm de mapare a resuselor.

- Spirit inovativ
- Cunostinte solide matematice
- Calitatea codului scris
- Calitatea documentatiei scrise
- Inclinatie spre “despicarea firului in patru”
- Abilitatea de exprimare
- Constanta in activitate
- Abilitatea de a performa la inalta calitate activitati monotone
- Ce alte calitati credeti ca ar trebui luate in considerare si in ce mod?

!Leadershipul înseamnă direcţionare şi motivare. - deci pentru Team Lider foarte mult trebuie sa aiba calitati de motivare, directionare comunicare

Team lideri: abilitatea de exprimare, spirit inovativ, inclinatie spre despicarea firului in 4, constanta in activitate, calitatea codului scris, abilitatea de a performa la inalta calitate activitati monotone, calitatea documentatiei scrise.

Developerii: Inclinatie spre “despicarea firului in patru, calitatea codului, constanta in activitate, abilitatea de exprimare, abilitatea de a performa la inalta calitate activitati monotone, calitatea documentatiei, cunostinte matematice

Testerii: calitatea documentatiei, abilitatea de a performa la inalta calitate activitati monotone, constanta in activitate, abilitatea de exprimare, …

Researcherii: Spirit inovativ, constanta in activitate, abilitatea de exprimare, inclinatie spre descpicarea firului in 4, calitatea documentatiei scrise, performarea la inalta calitatea a abil. monotone, cunostinte solide matematice, calitatea codului

Subiect 4

Descrieti utilizand diagrame UML structura si functionarea cat mai completa a structurii de conducere a statului roman: Presedentie, Guvern, Ministri, Parlament, Constitutie, Legi, etc.

Incercati sa atingeti cat mai multe aspecte pentru ca, in mod teoretic, cel ce primeste specificatia dumneavoastra sa aiba o vedere de ansamblu cat mai cuprinzatoare si detaliata asupra functionarii statului.

Se poate merge pe class diagram(ar merge niste interfete + mosteniri) + use-case diagram.

Subiect 5

Se considera urmatoarea structura de conducere (*N specifica o multiplicitate a nucleelor PM-DEV-TEST), structura functionala in cadrul companiei Microsoft (Redmond):

Sarcinile fiecarui post sunt urmatoarele:

- GM
  - alege proiectul
  - il descrie intr-o intalnire celor 3 lead.
  - se ocupa de impartirea banilor, gasirea de noi resurse, rezolvarea de probleme deosebit de importante (care trec de lead)
- PM-lead
  - imparte proiectul in sub-arii
  - scrie specificatiile high-level
  - supervizeaza pm-ii
  - se sincronizeaza cu dev-lead si test-lead
- Dev-lead
  - supervizeaza dev-ii
  - se sincronizeza cu pm lead si test lead
  - scrie cod la nevoie
  - face code review
- Test-lead
  - supervizeaza testerii
  - se sincronizeaza cu dev lead si pm lead
  - face code review la testeri
- PM
  - scrie specificatiile mid-level
  - imparte aria in “work item”-uri
  - face research
  - aduna resursele de care au nevoie DEV si TEST
  - se sincronizeaza cu ceilalti PM
- DEV
  - implementeaza specificatiile
  - se sincronizeaza cu alti DEV-i
- TEST
  - scrie test planul
  - il implementeaza

▫se sincronizeaza cu ceilalti TEST

▫face code review DEV-ului

Intrebari:

•Care sunt avantajele si dezavantajele structurii de conducere particulare prezentate?

•Cum ati modifica structura pentru a compensa dezavantajele?

•Motivatie completa pentru fiecare aspect

----------subiect2015----------

Subiecte „Managementul Proiectelor Software" - conf. dr. ing. Costin-Anton BOIANGIU

S1. Se consideră preluarea unui proiect ce are ca scop dezvoltarea, predarea cu punerea în funcțiune și mentenenanța unui sistem de comunicație tip Skype sau YahooMessenger.
Descrieți viziunea voastră asupra proiectului din următoarele puncte de vedere:
- Structură departamente înglobate în activitate;
- Structură ierarhică intra și inter-departamentală;
- Fluxul documentației;
- Strategii și variante de planificare;
- Riscuri și precauții specifice proiectului;

Acolo unde este posibil, utilizați diagrame UML.

S2. Un tester are de verificat funcționalitatea a 12 module procesatoare de date de mari dimensiuni. Pentru a ușura testarea, modulele sunt create astfel încât în cazul unei funcționări corecte să ruleze fix 12 minute fiecare iar în cazul în care apare o eroare în oricare din fazele execuției, un mecanism de captare excepții pornește un timer intern și dacă nu se reușește refacerea funcționalității termină modulul după exact 13 minute (marker ghinion :D) de la pornirea sa. Problema este însă că testerul este la finalul programului său de muncă de 8 ore și ați dori în calitate de PM ca mâine la începutul programului să aveți rezultatele testării. Testerul poate măsura în scriptul său de validare doar durata totală a execuției scriptului, durată ce apare imediat după finalizarea execuției acestuia. Presupunând că vă așteptați ca maxim un modul să fie defect, ce îi veți cere testerului să ruleze în scriptul său astfel încât la întoarcerea sa mâine la serviciu să vă trimită diagnosticul: dacă vreun modul este defect și dacă da, care?

S3. Descrieți utilizând diagrame UML structura și funcționarea cât mai completă a structurii facultății de Automatică și Calculatoare: departamente, decan, prodecani, directori de departamente, săli, cursuri, laboratoare, centre de cercetare, studenți, bibliotecă, secretariat, personal auxiliar, etc... Încercați să atingeți cât mai multe aspecte pentru ca, în mod teoretic, cel ce primește specificația dumneavoastră să aibă o vedere de ansamblu cât mai cuprinzătoare și detaliată asupra funcționării facultății noastre.

S4. ***MergeSort*** este un algoritm de sortare cu complexitatea O(n*log(n) inventat de John von Neumann în 1945. Este un exemplu de algoritm de tip divide et impera. Algoritmul merge sort execută următorii pași:
- Dacă lista este de lungime 0 sau 1, atunci este deja sortată. Altfel:
- Împarte lista nesortată în două subliste aproximativ egale.
- Sortează fiecare sublistă recursiv prin reaplicarea algoritmului merge sort.
- Se interclasează cele două liste și se obține lista inițială sortată.

QuickSort este un celebru algoritm de sortare, dezvoltat de C. A. R. Hoare și care, în medie, efectuează O(n*log(n)) comparații pentru a sorta n elemente. În cazul cel mai defavorabil, efectuează O(n^2) comparații. De obicei, în practică, quicksort este mai rapid decât ceilalți algoritmi de sortare de complexitate O(n*log(n)) deoarece bucla sa interioară are implementări eficiente pe majoritatea arhitecturilor și, în plus, în majoritatea implementărilor practice se pot lua, la proiectare, decizii ce ajută la evitarea cazului când complexitatea algoritmului este de O(n^2)
Quicksort efectuează sortarea bazându-se pe o strategie divide et impera. Astfel, el împarte lista de sortat în două subliste mai ușor de sortat. Pașii algoritmului sunt:
- Se alege un element al listei, denumit pivot
- Se reordonează lista astfel încât toate elementele mai mici decât pivotul să fie plasate înaintea pivotului și toate elementele mai mari să fie după pivot. După această partiționare, pivotul se află în poziția sa finală.
- Se sortează recursiv sublista de elemente mai mici decât pivotul și sublista de elemente mai mari decât pivotul.
- O listă de dimensiune 0 sau 1 este considerată sortată.

RadixSort este un algoritm ce nu folosește comparații de chei, ci distribuie cheile în grupe pe baza unei informații particulare referitoare la natura cheilor. Fie tipul cheilor Key alcătuit din k componente: f1, f2, ..., fk cu tipurile t1, t2, ... tk. Se consideră că o cheie (a1, a2, ..., an) este inferioară unei chei (b1, b2, ..., bn) unde ai și bi sunt valori ale câmpului fi cu i=1, 2, ..., k dacă există o valoare j între 0 și k-1 astfel încât: a1=b1, a2=b2, ..., aj=bj și aj+1<bj+1. RadixSort sortează după compartimente (grupe) mai întâi după câmpul fk (considerat câmpul cel mai puțin semnificativ), apoi după câmpul fk-1, și așa mai departe până la câmpul f1 (considerat câmpul cel mai semnificativ). Complexitatea algoritmului este O(k*n).
Cerință: Cei trei algoritmi de sortare prezentați mai sus au fiecare avantaje și dezavantaje. Efectuați o comparație a acestora, evidențiind unde anume diferă, în ce gen de aplicație, pe ce tip de arhitecturi le-ați utiliza pe fiecare?

S5. "Agile" vs "Cascadă" vs „V". Avantaje vs Dezavantaje. La ce gen de proiecte le-ați recomanda? La ce fel de echipe le-ați recomanda? Nu descrieți modelele in sine ci efectuați doar comparația.

Precizări generale:
- Scrieți fiecare subiect pe câte o foaie separată, cu numele si grupa dvs;
- Alegeti 4 dintre cele 5 subiecte;
- Timp de lucru: 80 min

S4 2015:

Quick sort: When you don't need a stable sort and average case performance matters more than worst case performance. A quick sort is O(N log N) on average, O(N^2) in the worst case. A good implementation uses O(log N) auxiliary storage in the form of stack space for recursion.

Advantage: Pretty fast in most cases

Disadvantage: The worst-case complexity of quick sort is O(n^2), which is worse than the O(n log n) worst-case complexity of algorithms like merge sort, heapsort, binary tree sort, etc.

Merge sort: When you need a stable, O(N log N) sort, this is about your only option. The only downsides to it are that it uses O(N) auxiliary space and has a slightly larger constant than a quick sort. There are some in-place merge sorts, but AFAIK they are all either not stable or worse than O(N log N). Even the O(N log N) in place sorts have so much larger a constant than the plain old merge sort that they're more theoretical curiosities than useful algorithms.

Avantaje: stabil, intotdeauna O(n log n), usurinta implementarii

Dezavantaje: foloseste multa memorie

Heap sort: When you don't need a stable sort and you care more about worst case performance than average case performance. It's guaranteed to be O(N log N), and uses O(1) auxiliary space, meaning that you won't unexpectedly run out of heap or stack space on very large inputs.

Avantaje: foarte eficient, consuma putina memorie, simplitate, performanta consistenta

Dezavantaje: instabil, mai incet decat quick si merge

Introsort: This is a quick sort that switches to a heap sort after a certain recursion depth to get around quick sort's O(N^2) worst case. It's almost always better than a plain old quick sort, since you get the average case of a quick sort, with guaranteed O(N log N) performance. Probably the only reason to use a heap sort instead of this is in severely memory constrained systems where O(log N) stack space is practically significant.

Insertion sort: When N is guaranteed to be small, including as the base case of a quick sort or merge sort. While this is O(N^2), it has a very small constant and is a stable sort.

Bubble sort, selection sort: When you're doing something quick and dirty and for some reason you can't just use the standard library's sorting algorithm. The only advantage these have over insertion sort is being slightly easier to implement.

Non-comparison sorts: Under some fairly limited conditions it's possible to break the O(N log N) barrier and sort in O(N). Here are some cases where that's worth a try:

Counting sort: When you are sorting integers with a limited range.

Radix sort: When log(N) is significantly larger than K, where K is the number of radix digits.

Avantaje: foarte rapid daca K e mic

Dezavantaje: consum de memorie, mai incet in cazurile reale, inutil pt K prea mare

Bucket sort: When you can guarantee that your input is approximately uniformly distributed.