Proiectarea Sistemelor Software Complexe

Curs 2 – Indicatorii de Calitate ai Unui Sistem Software Complex

2.1 Context.

Orice arhitectură trebuie proiectată astfel încât să se asigure faptul că sunt respectate anumite constrângeri impuse indicatorilor de calitate. În general indicatorii de calitate sunt: fiabilitatea, scalabilitatea, performanța și securitatea.

Indicatorii de calitate fac parte din cerințele non-funcționale ale unui sistem; prin intermediul acestor indicatori se cuantifică cum sunt îndeplinite cerințele funcționale. Orice sistem software complex are astfel de cerințe non-funcționale care sunt exprimate sub forma indicatorilor de calitate. Pentru a fi utile, cerințele referitoare la indicatorii de calitate trebuie sa fie formulate clar și concret. O greșeală frecvent întâlnită în documentele care descriu arhitectura unui sistem este reprezentată de formulări generice de genul: “Sistemul trebuie sa fie scalabil”. Acesta este o formulare imprecisă care nu spune prea multe. Nu este clar dacă scalabilitatea se referă la numărul de conexiuni simultane, sau la numărul de cereri simultane, sau la volumul mare de date, sau la toate aceste aspecte.

Definirea cu exactitate care dintre măsurile de mai sus trebuie respectate de sistem este crucială pentru proiectarea unei arhitecturi solide. Astfel o formulare corect ar fi: “Sistemul trebuie să poată fi scalat în ceea ce privește distribuirea de la 100 de utilizatori desktop aflați în locații geografice diferite la 10.000 de utilizator fără a crește costul de instalare și configurare ”. Această formulare este mult mai precisă, astfel pentru un arhitect este clar că trebuie să găsească o soluție care să permită instalarea și distribuirea sistemului cu efort “zero”.

2.2 Performanța (Performance)

Performanța ca și indicator de calitate reprezintă o măsură care definește fie volumul de procesări pe care o aplicație trebuie să îl poată face pe unitatea de timp sau termenul (deadline-ul) care trebuie respectat pentru finalizarea corectă a unei aplicații. Prima măsură a performanței este importantă pentru mai toate sistemele software din domeniul financiar, al telecomunicațiilor și guvernamental, toate aceste aplicații trebuind să proceseze sute, mii de tranzacții sau poate chiar zeci de mii de tranzacții pe secundă. A două măsură a performanței este importantă pentru aplicațiile de timp-real care sunt întâlnite mai ales în domeniul militar; pentru acest tip de aplicații întârzieri de o milisecundă pot avea consecințe grave. Există o serie de modalități în care performanța unui sistem poate fi cuantificată acestea putând varia de la o aplicație la alta. În acest curs vor fi analizate trei modalități de a cuantifica performanța unui sistem software: puterea de procesare, timpul de răspuns și termenul.

2.2.1 Puterea de Procesare (Throughput)

Puterea de procesare (throughput) reprezintă o măsură a volumului de procesări care trebuie realizate în unitatea de timp. Volumul de procesări se măsoară de cele mai multe ori în tranzacții pe secundă (tps) sau mesaje procesate pe secundă (mps). De exemplu, o aplicație de online banking poate să garanteze procesarea a 1000 de tranzacții pe secundă, iar o aplicație pentru gestionarea inventarului poate să proceseze 50 de mesaje pe secundă.

Este important să se înțeleagă ce se specifică prin puterea de procesare. Astfel într-un anumit context poate fi vorba de puterea de procesare medie calculată pentru un anumit interval de timp sau poate fi vorba de un vârf de procesare. Aceste două lucruri sunt diferite și influențează în mod diferit arhitectura sistemului.

Un exemplu elocvent este reprezentat de o aplicație online care prea pariuri. În majoritatea timpului puterea de procesare necesară este foarte mică întrucât nu se întâmplă mai nimic. Situația se schimbă însă atunci când are loc o cursă de cai, astfel înainte cu 10-5 minute de începutul cursei aplicația poate sa primească până la câteva sute ce cereri. În acest caz este crucial ca aplicația sa poată să proceseze în timp util toate cererile primite altfel afacerea va avea de suferit. De aceea în acest scenariu aplicația trebuie să fie proiectată astfel încât să asigure o putere de procesare care să satisfacă un vârf de cereri și nu un volum mediu.

2.2.2 Timpul de Răspuns (Response Time)

Acest indicator măsoară întârzierea introdusă de procesarea unei tranzacții. Timpul de răspuns este de cele mai multe ori măsurat ca timpul necesar unui sistem software pentru a răspunde la o anumită modificare apărută la intrările sistemului. Un timp de răspuns mic face ca utilizatorul unei aplicații să fie mai eficient, ceea ce evident este benefic pentru firma în care el lucrează. Un exemplu sugestiv este o aplicație de tip punct de vânzare folosită pentru un magazin de tip supermarket. Astfel atunci când este scanat un articol un răspuns rapid, de o secundă sau mai puțin, pentru afișarea prețului înseamnă că, clientul va fi servit rapid.

Și în acest caz este important să se distingă între valoarea medie a acestui indicator și cea garantată. Unele aplicații necesită ca toate cererile să fie tratate într-un anumit interval de timp, ceea ce înseamnă că este vorba de un timp de răspuns garantat. Altele însă pot să specifice valori medii pentru timpul de răspuns ceea ce înseamnă că întârzieri mai mari sunt permise atunci când sistemul este foarte încărcat. În acest ultim caz se mai poate impune o restricție de tip limită superioară pentru timpul de răspuns. De exemplu se poate cere ca 95% din cereri să fie tratate în mai puțin de patru secunde, iar o cerere nu trebuie să dureze mai mult de 15 secunde.

2.2.3 Termenul(Deadline)

Acest indicator măsoară intervalul de timp în care sistemul software trebuie să finalizeze un anumit task, finalizarea taskului după expirarea termenului fiind echivalentă cu apariția unei erori în sistem. Acest indicator este specificat în special pentru sistemele software de timp real. Astfel de sistem fiind întâlnite chiar și în sistemul bancar, de exemplu, o tranzacție efectuată la un bancomat este considerată invalidă dacă durează mai mult decât o perioadă de timp specificată.

2.3 Scalabilitatea

Scalabilitatea reprezintă un indicator ce măsoară cât de bine se comportă sistemul dacă dimensiunea problemei pentru care el a fost proiectat să o rezolve crește. Pentru ca acest indicator să devină unul concret este necesar să se stabilească ce poate să crească.

Proiectarea sistemelor software scalabile nu este un lucru ușor. De foarte multe ori necesitatea pentru scalabilitate nu este evidentă încă de la început. Este foarte important ca arhitectul să nu introducă în nucleul arhitecturii structuri care nu sunt scalabile. Chiar dacă scalabilitatea este prevăzută ca și o cerință pentru sistem de cele mai multe ori testarea scalabilității sistemului nu se poate realiza fie pentru că este prea costisitor din punct de vedere financiar fie fiindcă agenda proiectului nu permite acest lucru.

În continuare vor fi prezentate câteva exemple de indicatori concreți care exprimă scalabilitatea unui sistem.

2.3.1 Numărul de Cereri Simultane (Request Load)

Considerăm o aplicație care pe o anumită platformă hardware garantează o putere de procesare de 100 tps și un timp mediu de răspuns de o secundă. Dacă numărul de cereri simultane pentru acest sistem software crește de zece ori se pune întrebarea dacă arhitectura sistemului poate suporta o astfel de creștere.

În cazul ideal dacă nu se modifică platforma hardware pe care rulează sistemul, pe măsură ce numărul de cereri crește, puterea de procesare a aplicației ar trebuie să rămână constantă (100 tps), iar timpul de răspuns ar trebui să crească liniar (10 s). O arhitectură scalabilă ar permite în aceste condiții suplimentarea puterii de calcul a platformei hardware pentru a crește puterea de procesare a sistemului software și a micșora timpul de răspuns. Suplimentarea puterii de calcul se poate realiza în două feluri: (1) adăugarea mai multor procesoare și probabil mai multă memorie (scalare în sus – scale up) sau (2) distribuirea sistemului software pe mai multe mașini (scalare în exterior – scale out).

Scalarea în sus funcționează dacă sistemul a fost proiectat ca și sistem multi-fir, sau dacă mai multe instanțe ale aplicației pot fi executate în paralel pe aceeași mașină. Ultima variantă însă va consuma mai multă memorie întrucât procesele sunt mult mai costisitoare în ceea ce privește consumul de resurse decât firele de execuție.

Scalarea în exterior funcționează dacă efortul, necesar gestionării distribuirii cererilor pe mai multe mașini, este redus sau în cazul ideal zero. Obiectivul principal este acela de a menține toate mașinile pe care rulează aplicația la fel de încărcate în caz contrar investiția în hardware-ul suplimentar este irosită. Distribuirea încărcării în mod egal pe mai multe mașini poartă numele de load-balancing.

Un lucru foarte important de reținut este faptul că în ambele situații scalabilitatea trebuie să se realizeze fără a se modifica arhitectura sistemului. În realitate însă pe măsură ce încărcarea crește se va constata o scădere a puterii de procesare a sistemului și o creștere exponențială a timpului de răspuns. Acest lucru se întâmplă din două motive. Primul motiv este acela că, creșterea numărului de cereri duce la o creștere a competiției pentru resurse (CPU și memorie) între procesele și fire de execuție de pe mașina pe care rulează aplicația. Al doilea motiv este acela că fiecare cerere consumă resurse suplimentare, iar în cele din urmă se va ajunge la epuizarea resurselor și evident la limitarea scalabilității.

2.3.2 Numărul de Conexiuni Simultane (Simultaneous Connections)

Dacă un sistem software a fost proiectat să suporte un număr de 1000 de utilizatori concurenți se pune problema cum va reacționa sistemul dacă acest număr va crește foarte mult. Având în vedere că orice conexiune va consuma resurse este evident că numărul de conexiuni va fi limitat. De exemplu, dacă o aplicație creează cate un proces pentru fiecare conexiune este evident că resursele se vor termina relativ repede, procesele fiind mari consumatoare de resurse.

2.3.3 Dimensiunea Datelor (Data Size)

Acest indicator evaluează performanțele unui sistem software atunci când dimensiunea datelor procesate crește. De exemplu, o aplicație de "instant messaging" este proiectată să prelucreze mesaje text scurte, dar ce se întâmpla daca dimensiunea acestor mesaje crește foarte mult? Sau în cazul unei aplicații care caută informații într-o arhiva de date de dimensiuni specificate, se pune întrebarea ce se întâmplă dacă dimensiunea arhivei crește în ceea ce privește cantitatea de date stocată în ea.

2.3.4 Distribuirea (Deployment)

În ceea ce privește distribuirea unei aplicații interesează dacă creșterea numărului de utilizatori duce la creșterea efortului depus în vederea distribuirii și actualizării aplicației. Ideal ar fi ca aplicația să fie distribuită și actualizată printr-un mecanism automatizat, care să poată distribui și configura dinamic aplicația la un nou client sau să o actualizeze pentru un client vechi.

2.4 Toleranța la Modificări (Modifiability)

Toleranța la modificări este un indicator care măsoară cât este de ușor sau dificil să se modifice sistemul software pentru a implementa noi cerințe funcționale sau non-funcționale. Pentru a se evalua acest indicator se pot anticipa posibile modificări, de cele mai multe ori astfel de modificări sunt precizate chiar în cerințele sistemului. După ce au fost identificate posibilele modificări trebuie să se evalueze impactul pe care modificările îl vor avea asupra arhitecturii sistemului. În final calculându-se costul implicat pentru realizarea acestor modificări.

Arhitectura unui sistem trebuie proiectată în așa fel încât modificările ulterioare care sunt probabile să implice doar modificări locale la nivel de componente. Dacă se constată că o modificare ulterioară care este probabilă implică modificări în lanț, atunci este nevoie de o regândire a arhitecturii întregului sistem.

2.5 Securitatea (Security)

Cele mai uzuale cerințe referitoare la securitate sunt următoarele:

- **Autentificarea**: aplicația poate verifica identitatea utilizatorilor și a altor aplicații cu care comunică;
- **Autorizarea**: utilizatorii și aplicațiile autentificate au anumite drepturi de acces la resursele sistemului;
- **Criptarea**: mesajele trimise de și către aplicație sunt criptate;
- **Integritatea**: asigură faptul că, conținutul unui mesaje nu este modificat în timpul transmisiei;
- **Nerepudierea**: expeditorul unui mesaj este sigur că mesajul a ajuns la destinatar, iar destinatarul este sigur de identitatea expeditorului.

Există o serie de tehnologii care sunt folosite în prezent pe scară largă și care oferă suport pentru aceste aspecte ale securității unei aplicații. De exemplu, Secure Socket Layer (SSL) și Public Key Infrastructure (PKI) sunt folosite foarte des pentru aplicațiile Internet pentru a garanta autentificarea, criptarea și nerepudierea. Autentificarea și autorizarea sunt suportate in Java prin Java Authentication and Authorization Service (JAAS). Și exemplele pot continua.

2.6 Disponibilitatea (Availability)

Disponibilitatea unei aplicații este strâns legată de fiabilitate. Dacă o aplicație nu este disponibilă atunci când este nevoie de ea, atunci este puțin probabil că aplicația își îndeplinește rolul pentru care ea a fost dezvoltată. Majoritatea aplicațiilor trebuie să fie disponibile cel puțin în timpul orelor de lucru. Aplicațiile Internet trebuie însă să fie disponibile 24 din 24. Disponibilitatea poate fi măsurată ca și raportul de timp în care aplicația este utilizabilă.

Apariția unei defecțiuni face ca aplicația să fie indisponibilă. Defecțiunile influențează fiabilitatea unei aplicații care se măsoară ca fiind timpul mediu dintre apariția defecțiunilor. De obicei sistemele software care necesită o disponibilitate mare trebuie să nu conțină așa numitul “singur punct de defectare” (single point of failure) și să conțină mecanisme care să detecteze defecțiunea automat și să repornească componenta defectată.

Replicarea componentelor este o metoda eficientă de a crește fiabilitatea și evident disponibilitatea unui sistem software. Astfel, atunci când apare o defecțiune la o componentă replicată sistemul poate să funcționeze pentru că folosește celelalte replici ale componentei care încă funcționează. Se poate însă ca performanța sistemului să fie afectată de defecțiune, dar el va fi totuși disponibil.

Recuperarea după apariția unei defecțiuni afectează de asemenea disponibilitatea sistemului. Un sistem software are capacitatea de a se recupera dacă el revine la parametrii de funcționare normali după ce a apărut o defecțiune. Este de dorit ca defecțiunea să fie detectată automat, iar procedura de recuperare, de asemenea să fie inițiată automat. Având în vedere că pe parcursul cât se execută procedura de recuperare sistemul nu este disponibil, este de dorit ca această procedură să fie cât mai scurtă ca durată.

2.7 Integrarea (Integration)

Integrarea este un indicator care măsoară ușurința cu care sistemul poate fi incorporat într-un context de aplicații mai larg. De multe ori valoarea unei aplicații poate fi mărită dacă funcționalitatea sau datele produse de aplicație pot fi folosite în alte moduri decât cele care au fost prevăzute de cel care a proiectat aplicația. Cele mai folosite strategii de integrare sunt cele la nivelul datelor sau cele realizate printr-o interfață API.

Integrarea la nivelul datelor se poate realiza prin stocarea și manipularea datelor în așa fel încât alte aplicații să le poată accesa. De exemplu, poate să fie suficient să se folosească o baza de date relațională pentru stocarea datelor sau poate să fie nevoie de implementarea unei funcții care să permită exportarea datelor într-un format cunoscut (XML sau CSV).

Singurul dezavantaj al integrării la nivelul datelor îl constituie faptul că, aplicațiile care vor accesa datele nu mai sunt restricționate în nici un fel și pot modifica datele fără să respecte anumite reguli. Pentru a se evita acest lucru se poate dezvolta o interfața API prin intermediul căreia să se poată accesa datele, în acest fel putând fi respectate anumite reguli, în plus se poate asigura și o anumită securitate. Evident această a doua soluție este mai costisitoare decât prima, de aceea arhitectul trebuie să aleagă soluția care este potrivită pentru un anumit sistem software.

2.8 Alți Indicatori

Există o serie de alți indicatori de calitate care pot fi importanți pentru anumite tipuri de aplicații, de exemplu:

- **Portabilitatea (Portability)**: ușurința cu care o aplicație poate fi executată pe diverse platforme hardware și software, de obicei este dependentă de tehnologia folosită pentru implementare;
- **Testabilitatea (Testability)**: cât de ușor sau dificil poate fi testă o aplicație; este bine ca arhitectura să fie cât mai simplă;
- **Suportabilitatea (Supportability)**: cât de ușor se poate oferi suport pentru aplicație odată ce a fost scoasă în producție; prin suport se înțelege diagnosticarea și rezolvarea problemelor apărute în timpul funcționării; este bine ca un sistem să fie modular permițând astfel actualizarea doar a modulelor în care a fost găsită o problemă.

Bibliografie

[1] Ian Gorton. Essential Software Architecture. Editura Springer. 2006.