

Universitatea POLITEHNICA din București  
Facultatea de Electronică, Telecomunicații și Tehnologia Informației

**Arhitecturi software orientate pe servicii**

## **Lucrare de dizertație**

**Prezentată ca cerință parțială pentru obținerea  
titlului de *Master***

**în domeniul *Electronică, Telecomunicații și Tehnologia Informației*  
programul de studii *Tehnologii Software Avansate pentru Comunicatii***

**Conducător științific**  
**Eduard Popivici**

**Absolvent**  
**Andrei Mihaescu**

**Anul 2016**

## Declarație de onestitate academică

Prin prezenta declar că lucrarea cu titlul *Arhitecturi software orientate pe servicii*, prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității “Politehnica” din București ca cerință parțială pentru obținerea titlului de *Master* în domeniul Inginerie Electronică și Telecomunicații/ Calculatoare și Tehnologia Informației, programul de studii *Tehnologii Software Avansate pentru Comunicatii* este scrisă de mine și nu a mai fost prezentată niciodată la o facultate sau instituție de învățământ superior din țară sau străinătate. Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referințe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și fac referință la sursă. Reformularea în cuvinte proprii a textelor scrise de către alți autori face referință la sursă. Înțeleg că plagiatul constituie infracțiune și se sancționează conform legilor în vigoare. Declar că toate rezultatele simulărilor, experimentelor și măsurărilor pe care le prezint ca fiind făcute de mine, precum și metodele prin care au fost obținute, sunt reale și provin din respectivele simulări, experimente și măsurători. Înțeleg că falsificarea datelor și rezultatelor constituie fraudă și se sancționează conform regulamentelor în vigoare.

București, Iunie 2016.

Absolvent: Andrei Mihaescu

.....

# Cuprins

<b>Lista figurilor</b> . . . . .	iii
<b>Lista tabelelor</b> . . . . .	iv
<b>1. Introducere</b> . . . . .	1
1.1. Ce este o arhitectura software? . . . . .	1
1.2. De ce este arhitectura importantă ? . . . . .	2
1.3. Obiectivele arhitecturii software . . . . .	2
1.4. Principii arhitecturale cheie . . . . .	3
<b>2. Principii fundamentale ale arhitecturii software</b> . . . . .	4
2.1. Principii de proiectare . . . . .	5
2.1.1. Standarde de proiectare . . . . .	6
2.1.2. Nivelele aplicației . . . . .	6

## Lista figurilor

2.1. Organizarea pe "arii de responsabilitate" . . . . .	4
--	---

## Lista tabelelor

# Capitolul 1

## Introducere

### 1.1 Ce este o arhitectura software?

Arhitectura software reprezintă procesul de definire a unei soluții structurate care îndeplinește toate cerințele tehnice și operaționale, totodată optimizând metrici comune de calitate precum performanța, securitatea și facilitățile de gestionare. Aceasta presupune o serie de decizii bazate pe o gamă largă de factori fiecare din aceștia având un impact considerabil asupra calității, performanței, gestionabilității și bunei funcționării a aplicației.

Philippe Kruchten, Grady Booch, Kurt Bittner și Rich Reitman au derivat și rafinat definiția arhitecturii software bazându-se pe munca lui Mark Shaw și David Garlan (Shawn and Garlan 1996). Definiția lor este următoarea:

”Arhitectura software înglobează setul deciziilor semnificative legate de organizarea unui sistem software ce includ selectarea elementelor structurale și a interfețelor din care sistemul este compus; comportamentul așa cum reiese din interacțiunea acestor elemente; compunerea acestor elemente structurale și comportamentale în subsisteme mai mari; și un stil arhitectural care guvernează această organizare. Arhitectura software implică constrângeri și compromisuri legate de funcționalitate, utilitate, robustețe, performanță, reutilizare, inteligibilitate, economie, tehnice și estetice.”

În cartea *”Patterns of Enterprise Application Architecture”*, Martin Fowler evidențiază câteva teme recurente explicând conceptul de arhitectură. El identifică aceste teme după cum urmează: ”Descompunerea de nivel înalt a unui sistem în părți componente; deciziile care sunt dificil de schimbat; există multe arhitecturi într-un sistem; ceea ce este arhitectural important se poate schimba de-a lungul ciclului de viață al sistemului; și, la final, arhitectura se rezumă la lucrurile importante.”

În cartea *”Software Architecture in Practice (2nd edition)”* Bass, Clements, and Kazman definesc arhitectura astfel: ”Arhitectura software a unui program sau a unui sistem de calcul reprezintă structura sau structurile, ce înglobează elementele software, proprietățile lor vizibile către exterior și relația între acestea. Arhitectura se preocupă cu partea publică a interfețelor; detaliile private ale elementelor - cele ce sunt strict legate de implementarea internă - nu sunt legate de arhitectură.”

## 1.2 De ce este arhitectura importantă ?

Ca orice structură complexă, software-ul trebuie construit pe o bază solidă. Neluarea în considerare a anumitor scenarii cheie, cât și ignorarea anumitor probleme comune de design pot pune aplicația în pericol. Uneltele și platformele moderne ajută la construirea aplicațiilor, însă nu pot înlocui nevoia unei proiectări atente a aplicației, bazată pe scenarii și cerințe. Riscurile pe care le presupune o arhitectură slab gândită sunt instabilitatea, inabilitatea de a susține cerințele actuale și viitoare de business sau dificultatea de instalare și gestiune într-un mediu de producție.

Un sistem ar trebui proiectat luând în considerare utilizatorul, infrastructura și obiectivele de business. Pentru fiecare din aceste arii, trebuie gândite scenarii cheie, identificate proprietățile relevante și arii cheie de satisfacție și desatisfacție. Unde este posibil, este indicat să se stabilească metrici precise care vor putea fi evaluate pentru a măsura succesul fiecărei arii.

Cel mai probabil vor exista compromisuri și un echilibru trebuie găsit între cerințele concurente din aceste trei arii. De exemplu experiența utilizatorului, este adesea o funcție de business și infrastructură și schimbările într-una din zone o poate drastic afecta. În mod similar, schimbările la nivelul experienței de utilizare pot avea impact la nivelul infrastructurii și business. Performanța ar putea fi importantă din punct de vedere business și al utilizatorului, dar administratorul de sistem poate nu avea mijloacele financiare pentru a atinge obiectivele în 100% din timp. Un compromis ar fi să atingă obiectivele 80% din timp.

Rolul arhitecturii este acela de a defini modul în care elemente majore și componente din cadrul aplicației sunt folosite sau interacționează între ele. Selectarea structurilor de date și a algoritmilor, cât și detaliile de implementare specifice fiecărei componente nu sunt de interes pentru proiectare. Problemele de arhitectură și proiectare de multe ori se intercalează. În unele cazuri, deciziile țin mai mult de arhitectură. În altele, în schimb țin mai mult de proiectare și de cum acestea contribuie la realizarea arhitecturii.

## 1.3 Obiectivele arhitecturii software

Arhitectura software urmărește să găsească un compromis între specificațiile de business și cele tehnice înțelegând cazurile de utilizare și apoi găsind căi de implementare. Obiectivele arhitecturii sunt acelea de a găsi cerințele care influențează structura aplicației. O arhitectură bine realizată reduce riscurile de business asociate cu construirea unei soluții tehnice. Un design bun este suficient de flexibil ca să poată gestiona devierile de la tehnologiile software și hardware care pot apărea în timp, cât și cerințele utilizatorilor. Un arhitect trebuie să ia în considerare efectul global al deciziilor de proiectare, cât și compromisurile inerente între factorii de performanță, dar și cele cu privire la utilizator, infrastructură și cerințele de business.

## 1.4 Principii arhitecturale cheie

Pentru proiectarea unei arhitecturii următoarele principii cheie ar trebui luate în calcul:

- **Arhitectura trebuie să fie concepută pentru a suporta schimbări, nu pentru a rămâne neschimbată.** Întotdeauna trebuie luat în calcul cum aplicația s-ar putea modifica de-a lungul timpului pentru a putea răspunde noilor cerințe și provocări.
- **Modelarea trebuie făcută pentru a reduce riscul.** Este indicată folosirea uneltelor de proiectare, a sistemelor de modelare precum **UML** - *Unified Modeling Language* și a vizualizare unde este cazul pentru a putea evidenția cerințele și a deciziile arhitecturale și de proiectare, analizându-le impactul.
- **Folosirea modelelor și a uneltelor de vizualizare pentru comunicare și colaborare.** Comunicarea eficientă a designului, a deciziilor luate și schimbărilor necesare este necesară unei arhitecturi bune. Este recomandată folosirea modelelor, a vederilor și a altor mijloace de vizualizare a arhitecturii pentru a comunica și împărtăși eficient ideile cu clienții, rezultând astfel într-o comunicare rapidă a schimbărilor de proiectare.
- **Identificarea deciziilor tehnice cheie.** Este esențială înțelegerea deciziilor tehnice cheie pentru evitarea greșelilor comune. Investirea în luarea acestor decizii bine de prima dată este importantă pentru a obține un design flexibil și foarte puțin expus riscului de a fi afectat de viitoare schimbări.

Pentru rafinarea arhitecturii este recomandată o abordare incrementală și iterativă. Se pornește de la o arhitectură de bază pentru a avea o imagine de ansamblu, după care crează arhitecturii derivate pe măsură ce aceasta este testată și îmbunătățită. Modelul incipient nu trebuie să acopere toate nevoile, ci ar trebui să reprezinte un prim design pe care să poată fi testate cerințele de business. În mod iterativ vor fi adăugate detaliile care vor putea o primă imagine de ansamblu corectă, pentru ca mai târziu să fie adăugate și detaliile de finețe. O greșeală foarte des întâlnită este aceea de a se concentra pe detaliile neesențiale încă din faze incipiente și de a urma direcții greșite făcând presupuneri incorecte sau eșuând în a evalua eficient arhitectura.



## Capitolul 2

### Principii fundamentale ale arhitecturii software

Acest capitol va aborda principiile fundamentale de proiectare ale unei arhitecturii software. Aceasta este adesea descrisă ca organizarea sau structura unui sistem, care reprezintă o colecție de componente ce îndeplinesc o funcție sau un set de funcții specifice. Cu alte cuvinte, arhitectura se concentrează pe organizarea componentelor ce vor oferi o anumită funcționalitate. Organizarea funcțională a componentelor implică grupare acestora în "arii de interes", după cum se poate vedea în schemă de mai jos.

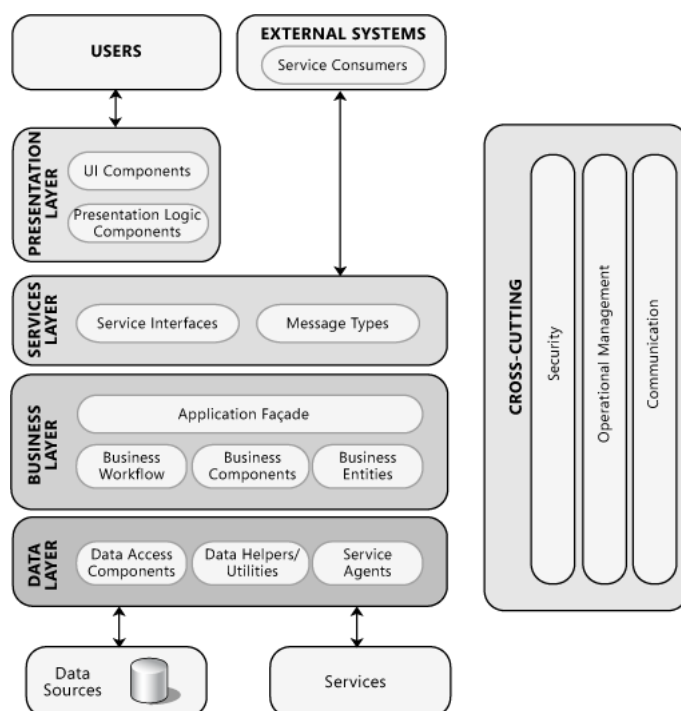


Figura 2.1: Organizarea pe "arii de responsabilitate"

Pe lângă gruparea pe componente, alte arii de interes se concentrează pe interacțiunea între acestea și pe cum ele funcționează împreună.

## 2.1 Principii de proiectare

La începerea procesului de proiectare trebuie avute în vedere principiile care vor contribui la crearea unei arhitecturii care aderă la practici consacrate, reduce costurile și care încurajează ușurința de folosire și posibilitățile de extindere. Acestea sunt:

- **Separarea atribuțiilor.** Aplicația trebuie divizată în funcționalități distincte cu o suprapunere cât mai mică între acestea. Factori importanți pentru a obține aceasta o reprezintă un nivel mare de coeziune și o cuplare slabă între componente, însă separarea greșită funcționalităților poate duce la o cuplare strânsă între acestea chiar dacă fiecare atribuțiile fiecărei componente nu se suprapun.
- **Principiul responsabilității unice.** Fiecare componentă sau modul ar trebui să îndeplinească o singură funcționalitate sau o agregare coerentă de funcționalități.
- **Principiul cunoașterii minime (cunoscut și sub numele de Legea lui Demeter).** O componentă sau un obiect nu ar trebui să cunoască detaliile interne de implementare ale altor componente sau obiecte.
- **Nu te repeta (eng. Don't repeat yourself - DRY)** Funcționalitatea ar trebui să fie implementată într-un singur loc. De exemplu, în cadrul proiectării unei aplicații, funcționalitate ar trebui implementată într-o singură componentă și nu ar mai trebui duplicată și în alta.
- **Minimizarea proiectării anticipate.** Proiectarea ar trebui să se rezume doar la ceea ce este necesar. În unele cazuri însă, atunci când costurile de dezvoltare sau al uni eșec în design sunt foarte mari, proiectarea anticipată este necesară. În altele, în special în cadrul dezvoltării agile, aceasta (eng. **BDUF - big design upfront**) poate fi evitată. Dacă cerințele aplicației sunt neclare sau există posibilitate unor evoluții a designului în viitorul apropiat, evitarea eforturilor de proiectare prematură este indicată. Acest principiu se mai numește și **YAGNI (eng. "You ain't gonna need it")**.

În cadrul proiectării unei aplicații sau a unui sistem, obiectivul unui arhitect este să minimizeze complexitatea separând designul în mai multe arii de responsabilitate. De exemplu, interfața grafică, procesele de business și accesul la date reprezintă arii diferite. În cadrul fiecăreia, componentele proiectate ar trebui să fie focusate pe responsabilități diferite, fără a conține cod din cadrul alteia. De exemplu, interfața grafică nu va conține cod care accesează direct sursele de date; în schimb va folosi componente specializate pentru obținerea datelor.

O evaluare cost/beneficiu va fi folosită pentru determinarea investiției necesare. În unele cazuri poate fi necesară o simplificare a structurii, permițând legarea interfeței grafice la un set de date. În general, separarea funcționalităților va fi făcută ținând cont și de partea de business. Cele ce urmează vor prezenta factori care afectează ușurința de design, implementare, testare și mentenanță a aplicației.

### 2.1.1 Standarde de proiectare

La nivelul fiecărui nivel logic, acolo unde este posibil, proiectare componentelor ar trebui să fie consistentă în cadrul unei operații. De exemplu, dacă se folosește tiparul "Table Data Gateway" pentru a crea un obiect care să reprezintă punctul de acces al tabelelor dintr-o bază de date, nu ar mai trebui folosit un altul (ex. "Repository") care folosește o altă paradigmă pentru accesarea datelor și inițializarea entităților de business. Însă, este posibilă folosirea altor tipare pentru operațiuni al unui alt modul care are o varietate mai largă de cerințe, cum ar fi o aplicație care necesită tranzacții business și rapoarte.

Funcționalitatea nu trebuie duplicată în cadrul aceleiași aplicații; o singură componentă va oferi această funcționalitate, care nu va mai exista într-o alta. Aceasta va duce la componente coerente și va ușura optimizarea acestora dacă o anumită funcționalitate va suferi modificări. În caz contrar, duplicarea funcționalității va îngreuna implementarea schimbărilor, scădea claritatea și va introduce potențiale inconsistențe.

Atunci când este posibil, compoziția este preferată în defavoarea moștenirii pentru refolosirea funcționalității deoarece crește dependența între părinte și clase care moștenesc. Aceasta de asemenea reduce ierarhiile de moștenire, ceea ce poate deveni dificil de gestionat.

Stabilirea unui stil de codare și a unei convenții de denumire pentru dezvoltare va oferi un model consistent care va facilita revizuirea codului de către membrii ai echipei care nu l-au scris, ceea ce duce la o administrare îmbunătățită.

Mentținerea calității sistemului se poate face folosind tehnici automatizate de QA pe parcursul dezvoltării, precum scrierea de teste unitare, analiza de dependențe și analiza statică a codului pe parcursul dezvoltării. Define clear behavioral and performance metrics for components and sub-systems, and use automated QA tools during the build process to ensure that local design or implementation decisions do not adversely affect the overall system quality.

Proiectarea componentelor aplicației și a sub-sistemelor cu o înțelegere clară a nevoilor operaționale specifice fiecăreia va ușura semnificativ instalarea și mentenanța acestora. În vederea eficientizării acestor două operațiuni metrice și date operaționale sunt necesare echipei responsabile de infrastructură. Pentru aceasta folosirea unetelor automatizate pentru asigurarea calității pot fi folosite.

### 2.1.2 Nivelele aplicației

Pentru a pune în practică principiul "ariilor de responsabilitate" aplicația este împărțită în blocuri funcționale a căror funcționalitate se suprapune cât mai puțin. Mare avantaj al acestei abordări este că fiecare bloc funcțional va putea fi optimizat independent față de restul aplicației. În plus, dacă unul încetează să funcționeze, acesta nu va cauza și defectarea altora, iar procesele pot rula independent unul față de celălalt. Această abordare contribuie și la facilitarea înțelegerii și proiectării aplicațiilor și simplifică administrarea sistemelor complexe interdependente.

Permițând fiecărui nivel să comunice cu celălalte sau să fie dependent de alte nivele va crește gradul de complexitate al aplicației și o va face mai greu de înțeles și administrat, așadar regulile de interacțiune între acestea trebuie foarte bine explicitate, astfel încât fluxul datelor în aplicație să fie foarte limpede.