Analiza și proiectarea sistemelor software

Curs 6

PROIECTARE ARHITECTURALĂ
în contextul proiectării

Proiectare:

- *Arhitectură*
- Interfeţe
- Componente

creştere grad de detaliere

Proiectare arhitecturală – dezvoltarea unui model abstract de nivel înalt al sistemului.

- Etapă timpurie în procesul de proiectare a sistemului.
- Implică *identificarea elementelor majore* ale sistemului şi a *comunicării* dintre acestea.

Proiectare de detaliu – descompunerea şi rafinarea componentelor arhitecturii.

PLAN CURS

Arhitectura software

Perspective arhitecturale

Stiluri şi mecanisme arhitecturale

Proiectare arhitectură sistem

Evaluare alternative arhitecturale

Descriere arhitectură

ARHITECTURA SOFTWARE

“Arhitectura software a unui sistem de calcul este
setul structurilor unui sistem
necesare pentru a realiza raţionamente referitoare la acesta,
structuri conţinând elementele software,
relaţiile dintre acestea şi
proprietăţile vizibile din exterior ale ambelor”.

Arhitectura software: reprezentare pe nivel superior de *abstractizare* şi *generalizare* a sistemului ce va fi dezvoltat.

http://www.sei.cmu.edu/architecture/definitions.html

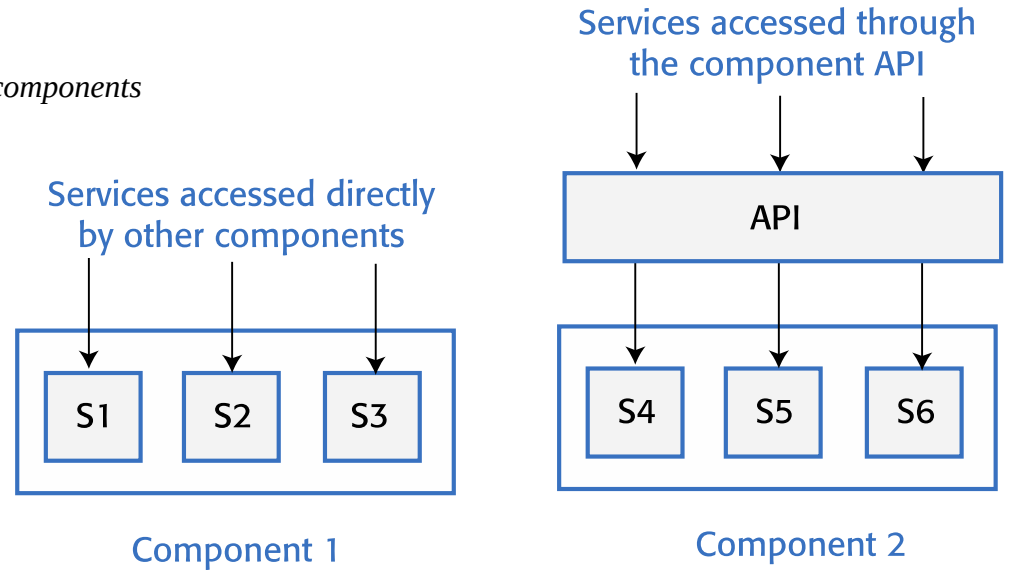
ARHITECTURA SOFTWARE ȘI COMPONENTE

Componentă = element care implementează un set coerent de funcționalitate.

Furnizează o colecție de unul sau mai multe servicii care pot fi folosite de alte componente.

Proiectare arhitectură software ⇒ proiectare interfețe componente (și lăsarea implementării interfețelor unei etape ulterioare din procesul de dezvoltare).

Figure 4.1 Access to services provided by software components
Ian Sommerville – Engineering Software Products



ARHITECTURA SOFTWARE

Importanţă:

- *comunicare* cu stakeholder-ii,
- evidenţierea *deciziilor de proiectare*, care au impact profund asupra procesului software şi asupra rezultatului acestuia,
- model intelectual *relativ mic* şi *uşor de manipulat*,
- *reutilizare*.

Utilitate:

- *analizarea* eficienţei soluţiei în îndeplinirea cerinţelor,
- considerarea de *alternative arhitecturale* în stadiile incipiente ale procesului software,
- *reducerea riscului* asociat construirii software-lui.

PLAN CURS

Arhitectura software

Arhitectura și atributele de calitate

Perspective arhitecturale

Stiluri şi mecanisme arhitecturale

Proiectare arhitectură sistem

Evaluare alternative arhitecturale

Descriere arhitectură

ARCHITECTURA ȘI ATRIBUTELE DE CALITATE

Orice sistem software are o arhitectură software (chiar dacă nu a fost proiectată expicit).

Cel mai simplu sistem este format dintr-un singur element aflat în relație cu el însuși.

Arhitectura software îndeplinește cerințele funcționale și promovează atribute de calitate.

Arhitectura software este critică în obținerea atributelor de calitate.

Proiectarea arhitecturii este condusă, în special, de cerințele pentru atribute de calitate.

ATRIBUTE DE CALITATE : exemple

Responsiveness – Sistemul întoarce rezultatele către utilizatori în timp rezonabil. (sensibilitate)

Reliability – Facilitățile sistemului se comportă așa cum se așteaptă dezvoltatorii și utilizatorii. (fiabilitate)

Availability – Sistemul poate oferi serviciile de fiecare dată când sunt cerute de utilizatori. (disponibilitate)

Security – Sistemul se protejează pe sine și datele utilizatorilor de atacuri neautorizate și de intruziuni. (securitate)

Usability – Utilizatorii pot accesa facilitățile de care au nevoie și le pot utiliza rapid și fără erori. (utilizabilitate)

Maintainability – Sistemul se poate actualiza rapid și se pot adăuga facilități noi fără costuri exagerate. (mentenabilitate)

Resilience – Sistemul poate să continue furizarea de servicii în cazul unei căderi parțiale sau al unui atac extern. (reziliență)

SOLUȚII ARHITECTURALE PENTRU ATRIBUTE DE CALITATE

Securitate

Folosirea unei arhitecturi în straturi (layered) cu activele critice plasate în straturile interioare.

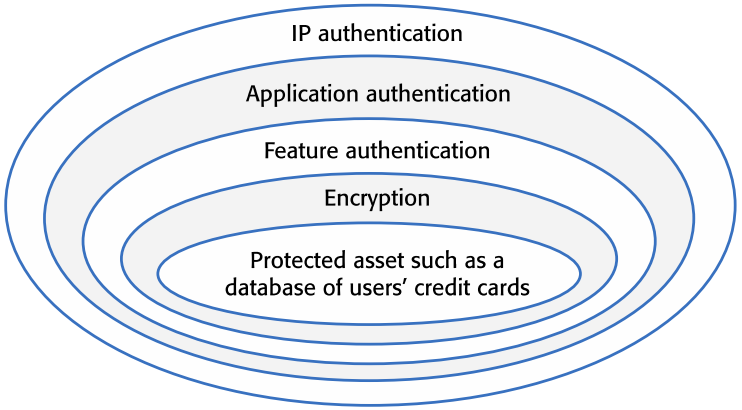


Figure 4.5 Authentication layers
Ian Sommerville – Engineering Software Products

Un atacator trebuie să penetreze toate straturile pentru a compromite sistemul.

Straturile pot include : straturi de autentificare, strat separat de autentificare la nivel de facilitate critică, strat de encriptare, etc.

Fiecare strat este o componentă separată astfel încât, dacă una din aceste componente este compromisă, celelalte straturi rămân intacte.

SOLUȚII ARHITECTURALE PENTRU ATRIBUTE DE CALITATE

Discuție referitoare la securitate:

Arhitectură de securitate cu informații centralizate :

- Mai ușor de proiectat și construit protecția
- Informația protejată poate fi accesată mai eficient.

Breșă de securitate $\Rightarrow$ se pierde totul.

Arhitectură de securitate cu informații distribuite :

- Costuri mai mari pentru realizarea protecției
- Durează mai mult până se accesează toate informațiile

Breșă de securitate $\Rightarrow$ doar informația dintr-o singură locație este pierdută.

SOLUȚII ARHITECTURALE PENTRU ATRIBUTE DE CALITATE

Siguranță

Izolarea facilităților critice d.p.d.v. al siguranței într-un număr mic de subsisteme.

Disponibilitate

Includerea de componente redundante și de mecanisme pentru toleranță la defecte.

Performanță

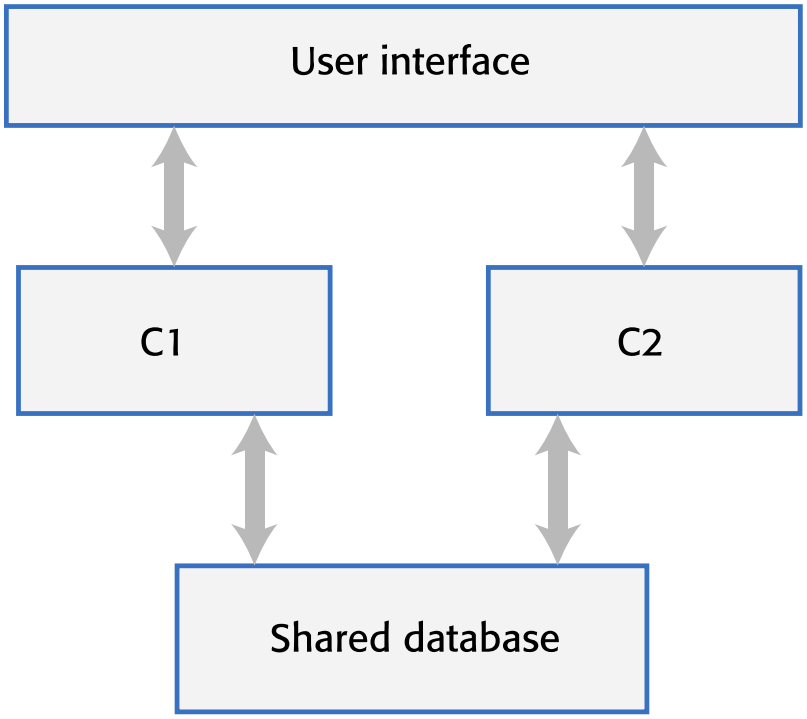
Localizarea operațiilor critice d.p.dv. al performanței și minimizarea comunicațiilor. Implică folosirea de componente mari (granularitate grosieră).

Mentenabilitate

Utilizare de componente mici, înlocuibile.

MENTENABILITATE ȘI PERFORMANȚĂ

Figure 4.2 Shared database architecture
Ian Sommerville – Engineering Software Products



Exemplu v.1:

- C1 rulează încet deoarece trebuie să reorganizeze informația din baza de date înainte de a o folosi.

- Singurul mod de a face C1 mai rapidă ar putea fi modificarea bazei de date.

⇓

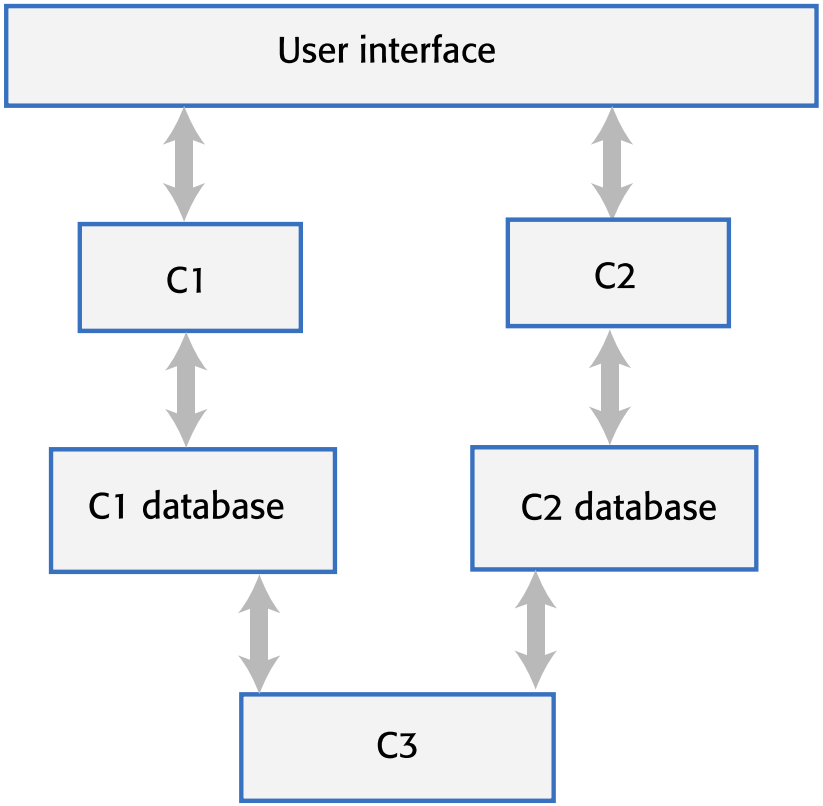
- C2 trebuie modificat corespunzător

Totuși

- Timpul de răspuns al lui C2 ar putea fi afectat.

MENTENABILITATE ȘI PERFORMANȚĂ

Figure 4.3 Multiple database architecture
Ian Sommerville – Engineering Software Products



Exemplu v.2:

- Fiecare componentă are o copie proprie a părților din baza de date de care are nevoie.

⇓

- Dacă o componentă trebuie să modifice organizarea bazei de date, aceasta nu afectează cealaltă componentă.

DAR

- Este necesară componenta C3 pentru a asigura păstrarea consistenței bazei de date când se modifică date partajate de C1 și C2.

Totuși

- O arhitectură cu baze de date multiple ar putea fi mai lentă iar implementarea și modificările ar putea fi mai costisitoare.

COMPROMIS: Mentenabilitate vs performanță

Mentenabilitate = *dificultatea* și *costul* modificărilor după lansarea sistemului.

Performanță = timpul de răspuns la o cerere de serviciu.

Soluție arhitecturală :
-Descompunere în componente mici, înlocuibile.

Consecință :
-Timpul de comunicare între componente.

⇒

Consecință :
-Răspuns lent ⇒ Performanță redusă

Compromis

Exemplu: Numărul de componente

CONFLICTE ARCHITECTURALE

Conflictul arhitectural apare atunci când două atribute de calitate sunt în tensiune, adică creșterea unei calități rezultă în scăderea celeilalte.

Exemple.

- Utilizarea de componente cu granularitate grosieră îmbunătăţeşte performanţa dar reduce mentenabilitatea.
- Introducerea de date redundante îmbunătăţeşte disponibilitatea dar complică asigurarea securităţii.
- Localizarea caracteristicilor legate de siguranţă înseamnă de obicei mai multă comunicare şi deci degradarea performanţei.

⇓

Proiectarea arhitecturală presupune o serie de ***decizii*** în scopul realizării ***compromisului*** optim pentru îndeplinirea ***cerinţelor de calitate***.

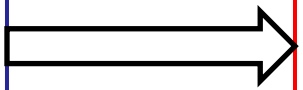
COMPROMIS: Securitate vs utilizabilitate

Securitate = abilitatea sistemului de a rezista la tentativele neautorizate de folosire a datelor și serviciilor, concomitent cu oferirea serviciilor către utilizatorii legitimi.

Utilizabilitate = ușurința de a învăța și de a accesa și folosi eficient sistemul software.

Soluție arhitecturală
-Arhitectură stratificată, cu activele critice în straturile interioare.

Consecințe :
-Informații (ex. parole) necesare pentru a penetra fiecare strat de securitate.
-Încetinirea interacțiunii cu sistemul.



Consecințe :
Efort de memorare
Insatisfacție utilizator

Compromis

Exemplu : Număr de straturi de securitate.

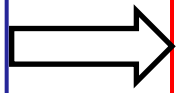
COMPROMIS: Disponibilitate vs competitivitate pe piață

Disponibilitate = măsura (procent din timpul total) timpului de funcționare a sistemului.

Competitivitate pe piață = timp de dezvoltare și costuri reduse.

Solutie arhitecturală
-Componente redundante

Consecințe :
-Componente senzor : detecție defect
–Componente comutator : comutarea operării către o componentă de rezervă.



Consecințe :
-Timp implementare componente suplimentare
-Creștere complexitate
-Risc de introducere erori și vulnerabilități

Compromis

Exemplu: Gradul de redundanță

.

Evaluare formativă

1. Ce relație există între arhitectura sistemului software și atributele de calitate ale acestuia ?

2. Ce este un conflict arhitectural și cum se poate rezolva ? Dați un exemplu.

https://forms.gle/ifZAWMNnTchdiUL66

PLAN CURS

Arhitectura software

Arhitectura și atributele de calitate

Perspective arhitecturale

Stiluri şi mecanisme arhitecturale

Proiectare arhitectură sistem

Evaluare alternative arhitecturale

Descriere arhitectură

MODULE ŞI COMPONENTE

- Un ***modul*** este o *unitate de implementare*.

Modularizarea se realizează pe baza *funcţionalităţii*, urmărindu-se în general *independenţa funcţională* (coeziune puternică şi cuplare externă slabă).

Un modul ***defineşte*** comportament şi interacţiuni cu alte module.

- O ***componentă*** este o *unitate de execuţie.*

MODULE ŞI COMPONENTE

- Un ***modul*** este o *unitate de implementare*.

- O ***componentă*** este o *unitate de execuţie.*

Componentele sunt instanţiate la execuţie pe baza definiţiilor din module.

Componentele au *comportamentul definit în modulele* din care sunt instanţiate şi *interacţionează prin conectori*.

Definirea ***structurii dinamice*** a sistemului (componente şi conectori) urmăreşte, în general, îndeplinirea *cerinţelor de calitate*.

PERSPECTIVE ARHITECTURALE

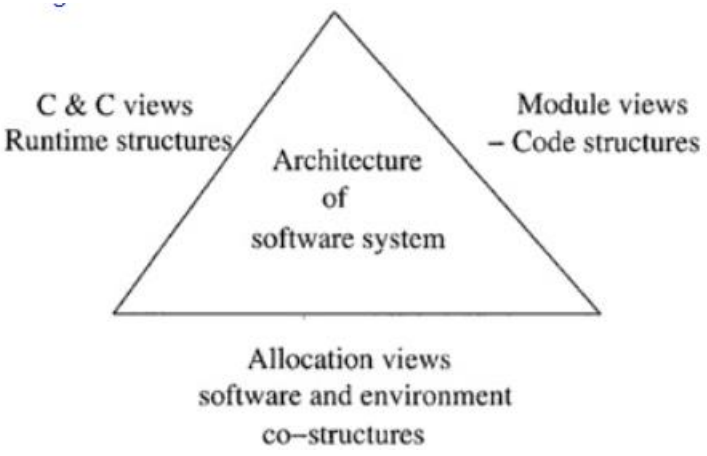
Există pespective arhitecturale multiple asupra aceluiaşi sistem.

Fiecare perspectivă expune un subset de *proprietăţi* şi *atribute* ale sistemului.

Avantaj: reducerea complexităţii reprezentării.

Perspective tipice:

- Statică (module)
- Dinamică (componente şi conectori)
- Alocare



PERSPECTIVE ARHITECTURALE

PERSPECTIVA STATICĂ

Sistemul = colecţie de *unităţi de cod* (module), fiecare implementând o anumită parte a *funcţionalităţii* sale.

Reprezintă *structura de dezvoltare* a sistemului.

Elementele primare:

Modulul: pachet, clasă, procedură, metodă, colecţie de funcţii, colecţie de clase.

Relaţia: agregare, dependenţă, generalizare/specializare.

PERSPECTIVE ARHITECTURALE

PERSPECTIVA DINAMICĂ

Sistemul = colecţie de *entităţi de execuţie* numite *componente şi conectori* (care leagă componentele).

Reprezintă *structura* sistemului *la momentul execuţiei*.

Elementele primare:

Componenta:

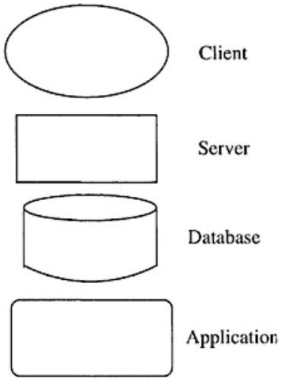
ex. obiect, colecţie de obiecte, proces.

Conectorul:

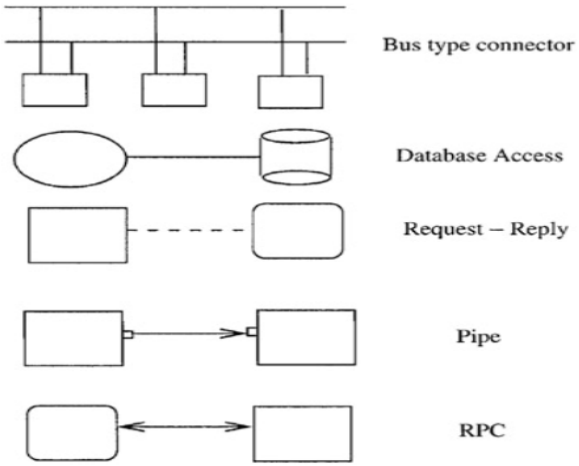
ex. conductă(pipe), socket,

date partajate, middleware.

notaţii



notaţii



PERSPECTIVE ARHITECTURALE

PERSPECTIVA ALOCĂRII

Modul de *alocare a unităţilor software la resurse* hardware, software sau umane.

Specifică relaţiile între elementele sistemului software şi elementele platformei pe care se va executa acesta, sau cu membrii echipei de dezvoltare.

Elementele primare:

Artefacte software: fişier executabil, fişier de date, fişier de configurare, etc.

Relaţii: proces – procesor, fişier de date – sistem de fişiere, etc.

PERSPECTIVE ARHITECTURALE

- ***Perspectiva statică*** - ilustrează *unităţile de implementare* ale sistemului (subsisteme, module), defineşte *interfeţele* acestora şi ilustrează *relaţiile* între ele.
- ***Perspectiva dinamică (a procesului)*** - ilustrează structura de procesare a unui sistem formată din *unităţi de execuţie* (componente) şi *relaţii* între acestea (conectori).
- ***Perspectiva alocării*** - ilustrează modul în care unităţile de implementare sunt distribuite pe elementele de infrastructură hardware (calculatoare) şi platforme software.

Pentru a ***documenta*** un proiect arhitectural ***din perspective diferite*** se utilizează *tipuri de vederi* corespunzătoare.

Evaluare formativă

1. Enumerați perspective fundamentale de reprezentare a arhitecturii software.

2. Realizați corespondența corectă între perspectivă și tipuri de elemente reprezentate ?

https://forms.gle/os8R5ZtVFaH9iffj9

PLAN CURS

Arhitectura software

Arhitectura și atributele de calitate

Perspective arhitecturale

Stiluri şi mecanisme arhitecturale

Proiectare arhitectură sistem

Evaluare alternative arhitecturale

Descriere arhitectură

STILURI ŞI MECANISME ARHITECTURALE
Exemple relevante

STILURI ARHITECTURALE (exemple)

Perspectiva dinamică:

- Arhitectură centrată pe date
- Arhitectură flux de date
- Arhitectură call-return
- Arhitectură OO
- Arhitectură bazată pe evenimente

Perspectiva statică

- Arhitectură multi-nivel

MECANISME ARHITECTURALE (categorii)

- Concurenţă
- Persistenţă
- Distribuire

STILURI ARHITECTURALE

Stil arhitectural - *model arhitectural generic*, specific unei anumite perspective arhitecturale.

Utilitate - punct de plecare în definirea arhitecturilor pentru sisteme particulare.

Exemple:

- Perspectiva statică : stilul mașină abstractă (layered)
- Perspectiva dinamică : pipe-and-filter, repository, client-server, event-based

Obs. Majoritatea sistemelor mari sunt eterogene şi nu urmează un singur stil arhitectural.

STILURI ARHITECTURALE

Def. ***Stil arhitectural***: *specializare* pentru *tipuri de elemente* şi *de relaţii*, împreună cu un set de *constrângeri* referitoare la modul *de utilizare* a acestora.

În perspectiva dinamică :

Descrie o categorie de sisteme software în termeni de:

- setul *tipurilor de componente* ce realizează funcţii necesare în sistem,
- setul *tipurilor de conectori* ce permit comunicarea, coordonarea şi cooperarea componentelor,
- *constrângerile* asupra integrării componentelor,
- *modele semantice* referitoare la proprietăţile de ansamblu ale sistemului.

Stabileşte *o organizare* de ansamblu a componentelor sistemului.

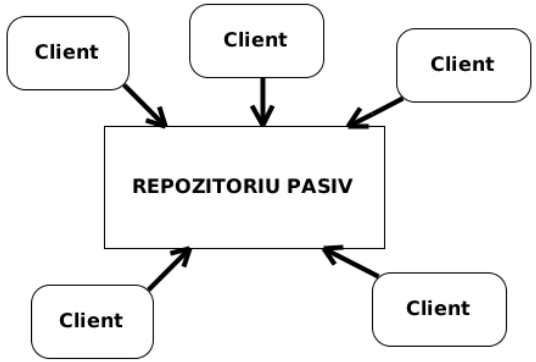
STILURI ARHITECTURALE
(perspectiva dinamică)

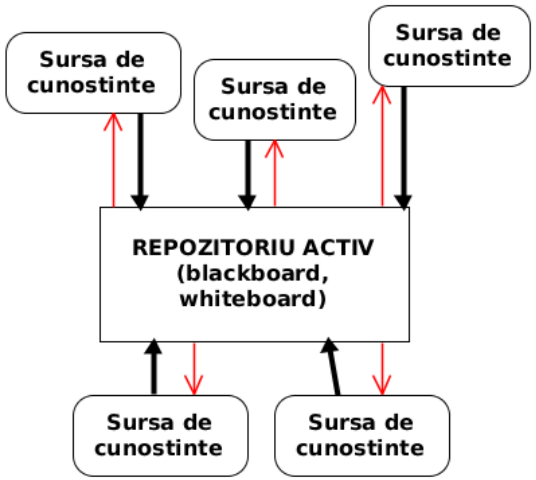
ARHITECTURĂ CENTRATĂ PE DATE (repository)

Componente:

- *Structură de date centrală*.
- Colecţie de *componente independente*, ce operează pe structura de date centrală.

Stilul *repository* este folosit în majoritatea cazurilor în care trebuie partajate cantităţi mari de date.





Exemple de aplicare:

Baze de date – repozitoriu pasiv

Inteligenţă artificială – repozitoriu activ

STILURI ARHITECTURALE
(perspectiva dinamică)

ARHITECTURĂ FLUX DE DATE

Datele: flux transferat prin *conducte*.

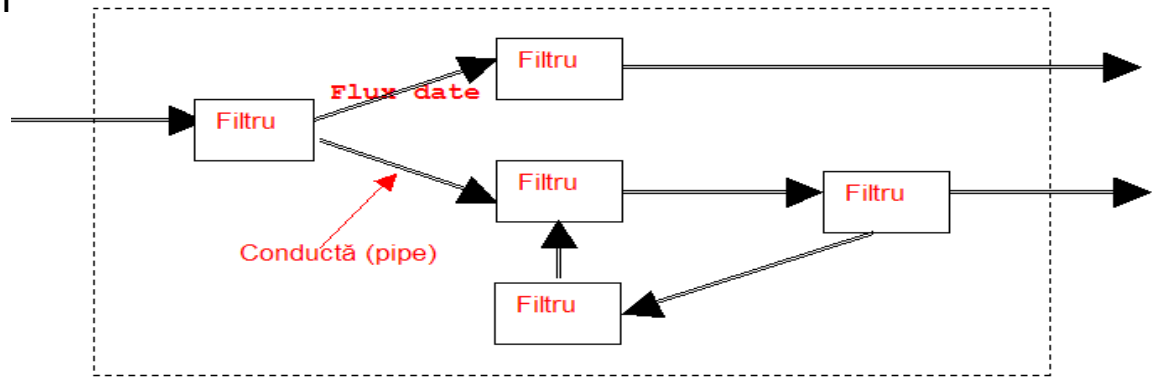
Prelucrările : componente independente

Componentele sunt procese ce realizează *transformări funcţionale* asupra *datelor din fluxul de intrare* şi trimit rezultatul acestora în *fluxul de ieşire*.

Exemple :

- *Batch-sequential* Procesare ***secvențială***: componentele transformă toate datele de pe intrare după care rezultatul este trimis spre prelucrare următoarei componente.
- *Pipe-and-filter* Procesare ***incrementală***: componentele aplicã transformările și transmit rezultatul componen

Exemplu : pipe-and-filter



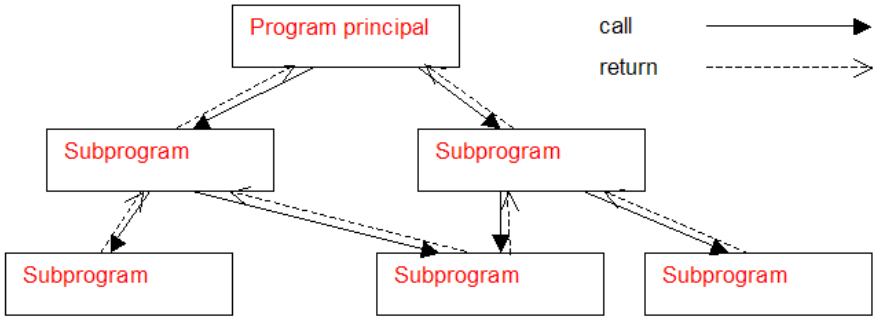
STILURI ARHITECTURALE
(perspectiva dinamică)

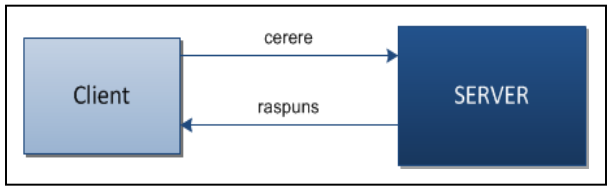
ARHITECTURĂ CALL-RETURN - structură uşor de modificat şi scalabilă.

- componentă (fir principal de control) ce execută operaţii de *invocare explicită*
- un set de componente către care se fac invocările.

Fiecare componentă poate, la rândul ei, să execute operaţii de invocare explicită către alte componente din set.

Fiecare componentă ce poate fi invocată prezintă unul sau mai multe *puncte de acces*.





Sub-stiluri:

- Arhitectură program principal – subprogram
- Client/server
- Peer-to-peer
- SOA

STILURI ARHITECTURALE
(perspectiva dinamică)

ARHITECTURĂ ORIENTATĂ OBIECT

Componentele : încapsulează date şi operaţiile de manipulare a acestora.

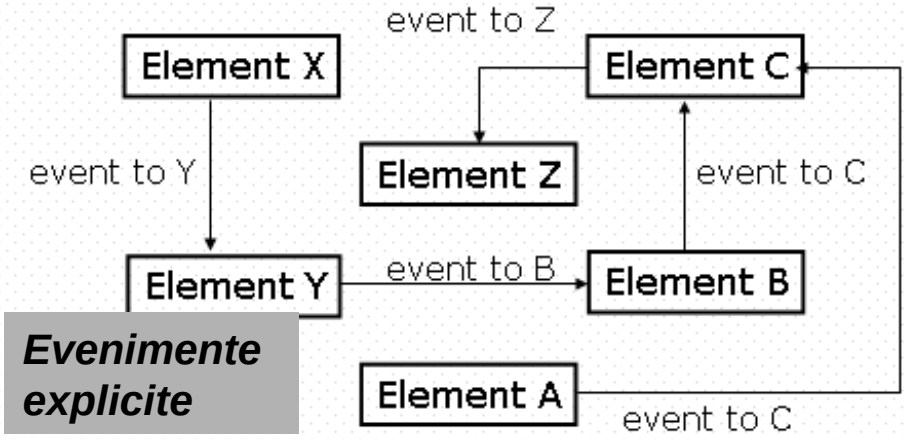
Comunicarea : transfer mesaje.

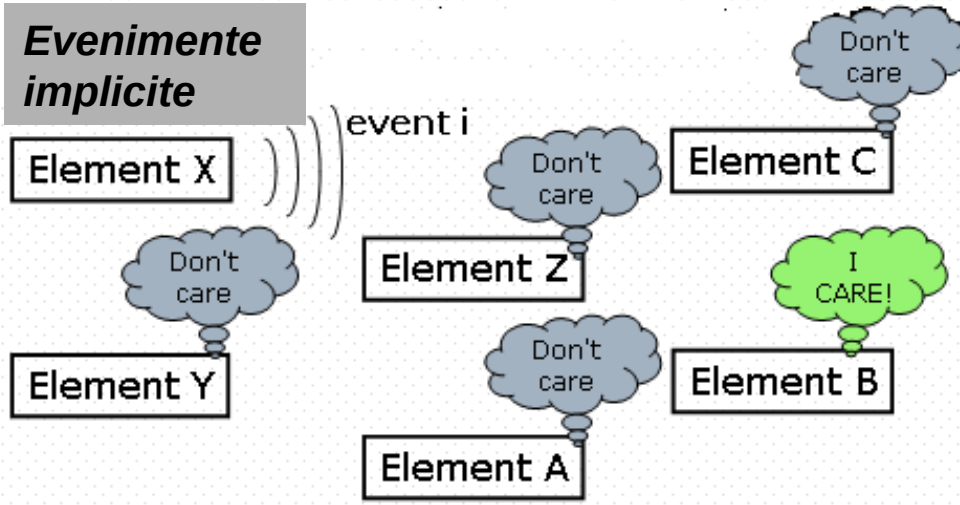
Avantaje
-obiectele pot fi *task-uri concurente* într-un sistem,
-obiectele pot avea *interfeţe multiple*,
-stilul permite *modificare transparentă a implementării*, fără efecte colaterale
-stilul oferă suport pentru o proiectare care să *descompună problemele* în colecţii de agenti ce interacţionează.

Dezavantaje
-interacţiunea cu un obiect presupune *cunoaşterea identităţii* acestuia, deci orice modificare de identitate induce modificări în obiectele ce-l invocă explicit.

STILURI BAZATE PE EVENIMENTE
(perspectiva dinamică)

Organizarea aplicaţiei în componente ce *comunică evenimente*, şi/sau *reacţionează la evenimente* generate de alte componente.





Exemple de aplicare:

- Biblioteci GUI – sisteme interactive
- Sisteme distribuite – av. decuplare şi reorganizare

STILURI ARHITECTURALE
(perspectiva statică)

ARHITECTURĂ MULTI-NIVEL (stratificată)

Presupune organizarea sistemelor pe nivele ierarhice, unde nivelul **n** oferă servicii nivelului **n+1** şi este client pentru serviciile nivelului **n-1.**

Obs. Opacitatea nivelelor poate fi totala sau parţiala.

Modulele : *maşini abstracte*, corespunzătoare unui nivel.

Relaţiile între module : *protocoale* ce definesc modul în care interacţionează nivelele, *API*-uri.

Stilul ofera suport pentru:

- proiectare bazată pe nivele de abstractizare multiple (rafinarea soluţiilor),
- suport pentru dezvoltare ulterioară,
- suport pentru reutilizare.

STILURI ARHITECTURALE

Fiecare stil arhitectural răspunde mai bine unui anume *set de cerinţe de calitate*.

Ex. Stilul pipe-and-filter răspunde bine cerinţelor de reutilizabilitate şi adaptabilitate, dar nu şi celor de performanţă sau de interactivitate.

Soluţia unei probleme la nivel arhitectural poate fi un stil sau o *combinaţie* de stiluri.

Uneori mai multe stiluri pot fi *variante* alternative ale soluţiei. Aceste variante vor fi proiectate arhitectural şi apoi evaluate în vederea *alegerii* arhitecturii finale.

MECANISME ARHITECTURALE

Mecanism arhitectural:

- Impune o *transformare a modelului* unei arhitecturi.
- Este centrat asupra unui *singur aspect* al arhitecturii.
- Impune o *regulă asupra arhitecturii*, descriind modul în care software-ul va gestiona un aspect al funcţionalităţii la nivelul infrastructurii.
- Adresează probleme de *comportament* în contextul arhitectural.

Un stil arhitectural se poate utiliza în conjuncţie cu unul sau mai multe mecanisme arhitecturale pentru a contura structura de ansamblu a unui sistem.

MECANISME ARHITECTURALE

Definesc o *abordare specifică* în realizarea unor caracteristici de *comportament* ale sistemului.

La proiectare, pentru fiecare mecanism arhitectural ales se va alege unul din mecanismele de proiectare existente pentru acesta.

Domenii (reprezentative) ale mecanismelor arhitecturale:

- Concurenţă
- Persistenţă
- Distribuire

MECANISME ARHITECTURALE

CONCURENŢĂ

Mecanisme pentru simularea şi gestionarea paralelismului activităţilor.

Exemple de ***mecanisme de proiectare***:

Managementul proceselor la nivelul sistemului de operare:

- Soluţie pentru concurenţă, planificare, comunicare între procese realizată de sistemul de operare şi utilizată de aplicaţie prin API-ul corespunzător.

Planificator de task-uri la nivelul aplicaţiei:

- Planificatorul
- Obiecte active ce implementează operaţia `tick()`.

MECANISME ARHITECTURALE

PERSISTENŢĂ

- Date ce supravieţuiesc procesului ce le-a creat.
- Memorate în baze de date sau în fişiere.
- Accesibile ulterior altor procese.

Exemple de ***mecanisme de proiectare :***

Sisteme de gestiune a bazelor de date:

- Aplică arhitecturii aplicaţiei capabilitatea de memorare şi extragere date a unui SGBD.

Persistenţă la nivelul aplicaţiei:

- Construieşte în arhitectura aplicaţiei capabilitatea de persistenţă.

MECANISME ARHITECTURALE

DISTRIBUIRE

Elementele problemei:

- modul de interconectare a entităţilor
- natura comunicării

Definesc maniera de comunicare a sistemelor sau componentelor într-un mediu distribuit.

Exemplu de ***mecanism de proiectare***:

Broker :

Intermediar al comunicării între o componentă client şi o componentă server.

Evaluare formativă

1. Realizați corespondența corectă între stil și forme de comunicare între componente.

2. Realizați corespondența corectă între mecanism și categoria căreia îi aparține.

3. Ce relație există între stiluri arhitecturale și mecanisme arhitecturale.

https://forms.gle/jeFoUuGqiQHmxCG39

ANALIZA MODELULUI ARHITECTURAL

Analiza ***controlului*** :

- Modul de realizare a controlului.
- Existenţa unei ierarhii de control şi rolul componentelor în cadrul acesteia.
- Mecanismul de transfer al controlului între componente.
- Partajarea controlului între componente.
- Topologia controlului.
- Tipul de control: sincronizat sau asincron.

ANALIZA MODELULUI ARHITECTURAL

Analiza ***datelor*** :

- Modul de comunicare a datelor între componente.
- Flux continuu vs. obiecte trimise sporadic sistemului.
- Flux de date vs. date partajate.
- Existenţa şi rolul componentelor de date (ex. blackboard).
- Modul de interacţiune a componentelor funcţionale cu componentele de date.
- Componente de date pasive vs. componente de date active.
- Interacţiunea datelor cu controlul, în cadrul sistemului.

PLAN CURS

Arhitectura software

Arhitectura și atributele de calitate

Perspective arhitecturale

Stiluri şi mecanisme arhitecturale

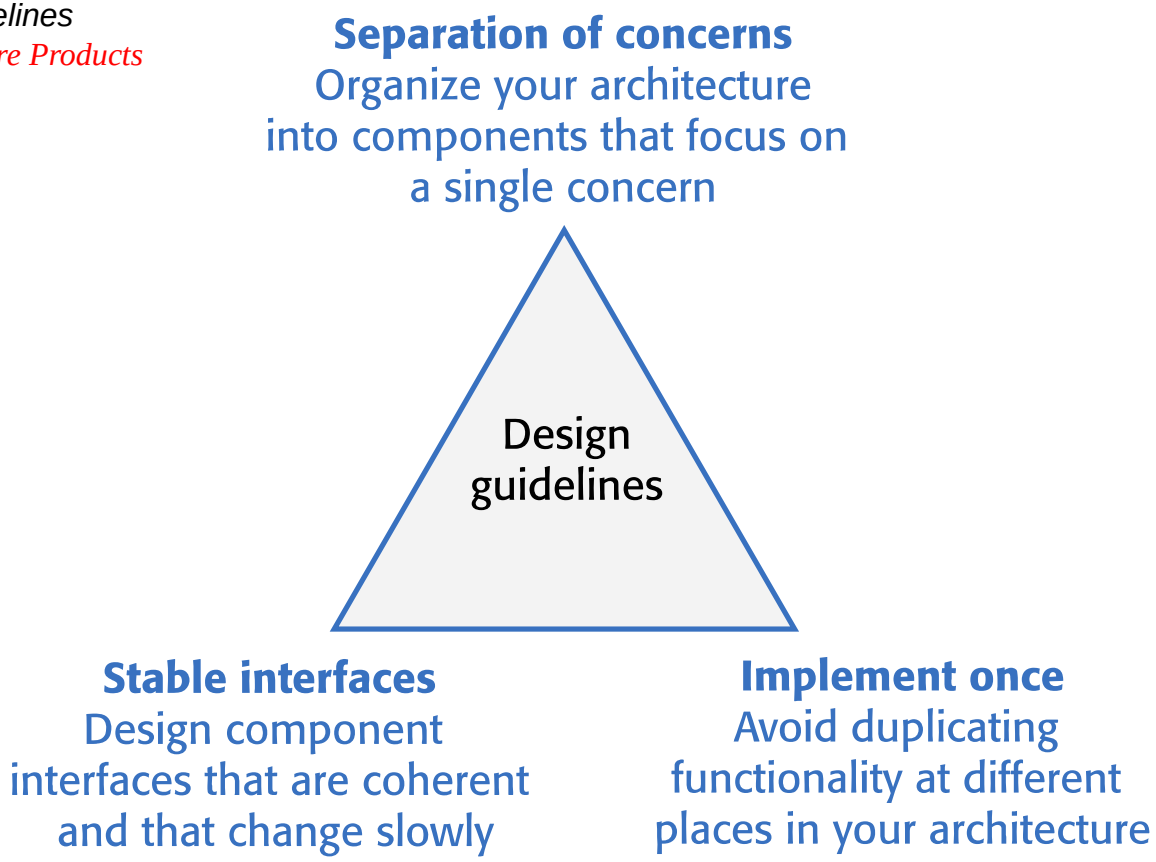
Proiectare arhitectură sistem

Evaluare alternative arhitecturale

Descriere arhitectură

INDICAȚII DE PROIECTARE PENTRU ARHITECTURI SOFTWARE

Figure 4.8 Architectural design guidelines
Ian Sommerville – Engineering Software Products



PROIECTARE ARHITECTURĂ SISTEM

1. Reprezentarea sistemului în context și definire interfețe externe
2. Partiţionare sistem şi definire interfeţe interne
3. Proiectare arhitectură date
4. Rafinarea arhitecturii în componente
5. Descrierea de instanţieri ale sistemului

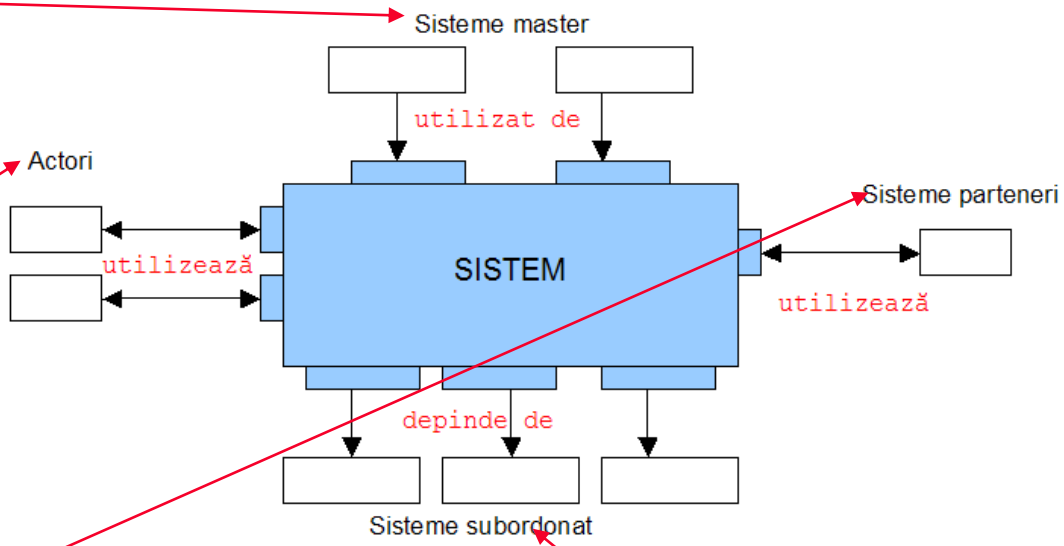
Proiectare arhitectură sistem
1. REPREZENTAREA SISTEMULUI ÎN CONTEXT

ACD (architectural context diagram)

CONTEXTUL = entităţile externe sistemului, cu care acesta interacţionează.

Utilizează sistemul ca parte a unei scheme de procesare de nivel mai înalt

Entităţi (persoane, dispozitive) ce interacţionează cu sistemul prin producere sau consum de informaţii.



Sisteme cu care interacţiunea este de tip peer-to-peer (reciproc producere şi consum de informaţii).

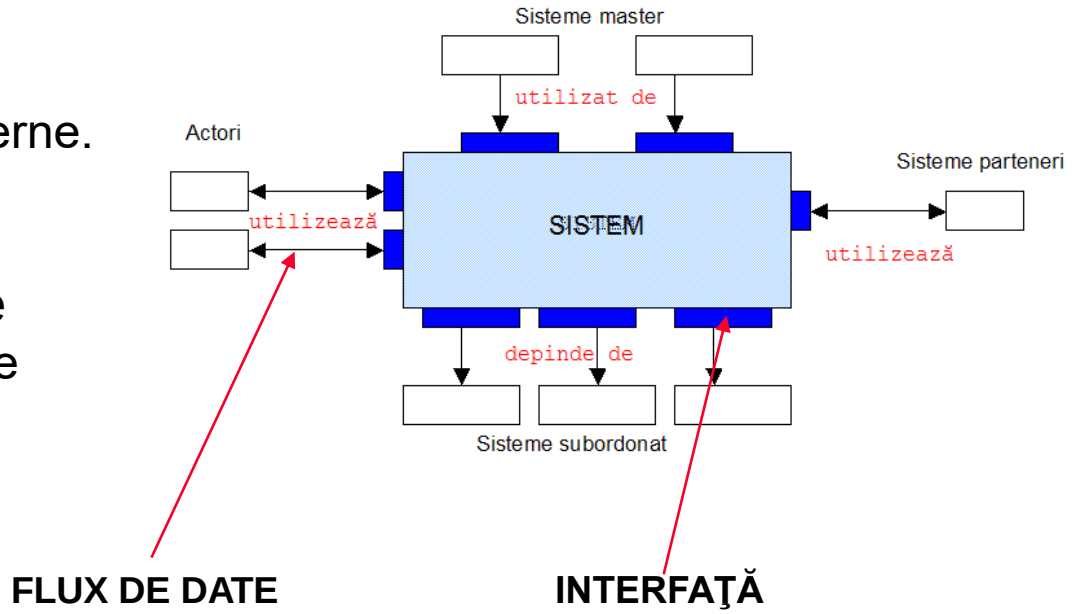
Utilizate de sistem; oferă date sau procesări necesare îndeplinirii funcţionalităţii sistemului.

Proiectare arhitectură sistem
1. REPREZENTAREA SISTEMULUI ÎN CONTEXT

OPERAŢII NECESARE

•Specificare interfeţe externe.

•Identificarea fluxurilor de date de intrare şi de ieşire în/din sistem.



Proiectare arhitectură sistem
2. PARTIŢIONARE SISTEM ŞI DEFINIRE INTERFEŢE INTERNE

Obs. Recomandare : combinare abordare top-down cu abordare bottom-up şi reconcilierea problemelor decoperite.

Ghid de organizare pe nivele de abstractizare:

- Creare nivele de abstractizare prin descompunere ierarhică a sistemului şi subsistemelor sale.
- Limitarea numărului de elemente la fiecare nivel (5 – 50).
- Fiecare element trebuie să aibă un scop coerent.
- Elementele vor fi încapsulate, iar detalii interne vor fi ascunse în spatele interfețelor.

Proiectare arhitectură sistem
2. PARTIŢIONARE SISTEM ŞI DEFINIRE INTERFEŢE INTERNE

Strategia de partiţionare – bazată pe o *descompunere dominantă*.

Exemple de descompuneri dominante:

- **Funcţionalitate** – grupare funcţii înrudite.
- **Arhetipuri** – tipuri remarcabile din domeniu.
- **Stil arhitectural** – mapare pe elementele definite de stilul arhitectural ales.
- **Atribute de calitate** (ADD) – identificarea atributelor de calitate importante şi alegerea sau crearea unui şablon (stil arhitectural, şablon de proiectare, şablon specific domeniului) care să le îndeplinească.
- **Reutilizare** – organizare în jurul unor elemente existente (ex. bază de date, componentă cumpărată (COTS), cod din proiecte anterioare) şi adăugare de conectori şi adaptoare.

Proiectare arhitectură sistem
3. PROIECTARE ARHITECTURĂ DATE

Proiectare structuri de date de nivel înalt:

- baze de date

Utilizate pentru operaţii curente, tranzacţionale, asupra datelor.

- depozite de date (data warehouse)

Baze de date independente, de dimensiuni mari, cu organizare specifică, utilizate pentru operaţii de analiză a datelor.

Proiectare arhitectură sistem
4. RAFINAREA ARHITECTURII ÎN COMPONENTE

Surse :

- Domeniul aplicaţiei : reprezentat în diagrama claselor de analiză sau în modelul fluxului de date.

⇒ componente ce implementează funcţiile business

- Domeniul infrastructurii:

⇒ ex. componente pentru management memorie, componente de comunicare, sisteme de gestiune a bazelor de date, componente pentru management task-uri.

- Interfeţele externe: specificate în reprezentarea sistemului în context.

⇒ componente care procesează fluxul de date prin interfeţe.

Reprezentare:

ex. UML - diagramă de componente

Proiectare arhitectură sistem
5. DESCRIERE DE INSTANŢIERI ALE SISTEMULUI

Instanţiere = aplicarea arhitecturii la o problemă specifică.

Scop:

- analiza arhitecturii pentru a demonstra validitatea structurii şi componentelor,
- rafinarea arhitecturii.

PLAN CURS

Arhitectura software

Arhitectura și atributele de calitate

Perspective arhitecturale

Proiectare arhitectură date

Stiluri şi mecanisme arhitecturale

Proiectare arhitectură sistem

Evaluare alternative arhitecturale

Descriere arhitectură

EVALUARE ALTERNATIVE ARHITECTURALE

ATAM – architecture trade-off analysis method

http://www.sei.cmu.edu/publications/documents/00.reports/00tr004.html

1. Colectare scenarii
2. Extragere cerinţe, constrângeri şi descriere context
3. Descrierea stilurilor şi şabloanelor arhitecturale alese pe baza scenariilor şi cerinţelor ⇒ mai multe arhitecturi candidat
4. Evaluarea individuală (izolată) a fiecărui *atribut de calitate*
5. *Identificarea senzitivităţii* atributelor de calitate în raport cu diferite variante arhitecturale, pentru un stil arhitectural specific.
6. Critica arhitecturilor candidat (pas.3) utilizând analiza de senzitivitate (pas.5).

Fiabilitate, performanţă, securitate, mentenabilitate, flexibilitate, testabilitate, portabilitate, reutilizabilitate, interoperabilitate,etc.

PROCEDURA:

1. Modificare arhitectură.
2. Măsurare variaţie atribut de calitate.

PRELUCRARE REZULTAT

1. Eliminarea unor arhitecturi.
2. Modificarea şi reprezentarea mai detaliată a celor rămase.
3. Reluarea aplicării ATAM.

EVALUARE ALTERNATIVE ARHITECTURALE

COMPLEXITATE ARHITECTURALĂ

Rezultă din fluxul informaţional şi din fluxul de control din sistem.

Evaluare complexitate considerând *dependenţele* dintre componentele arhitecturii.

Tipuri de dependenţe:

- dependenţe de partajare
- dependenţe de flux
- dependenţe de restricţionare

Producători / consumatori în competiţie la aceleaşi resurse.

Între producători şi consumatori ai aceluiaşi flux.

Constrângeri asupra fluxului de control relativ la un set de activităţi.
(ex. excludere mutuală)

PLAN CURS

Arhitectura software

Arhitectura și atributele de calitate

Perspective arhitecturale

Proiectare arhitectură date

Stiluri şi mecanisme arhitecturale

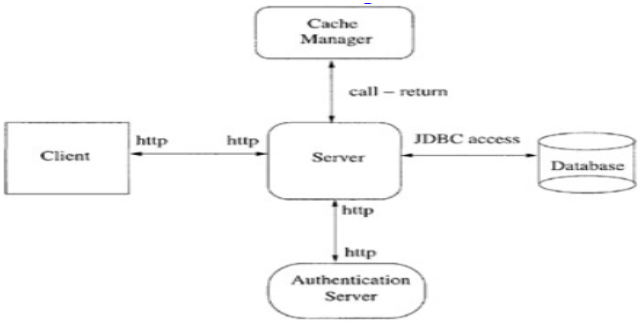
Proiectare arhitectură sistem

Evaluare alternative arhitecturale

Descriere arhitectură software

DESCRIERE ARHITECTURĂ SOFTWARE

- Diagrame specifice
- Diagrame UML
  - diagrame de clase, de pachete
  - diagrama de componente
  - diagrama de instalare (deployment)



- Instrumente
  - Ex. Rational Software Architect (support for model-driven development)
- www.ibm.com/software/awdtools/architect/swarchitect

- Limbaje pentru descriere arhitecturală (ADL)

  Sintaxă şi semantică pentru operaţii de:
  - descompunere arhitecturală în componente
  - compunere componente în blocuri arhitecturale
  - reprezentarea mecanismelor de conectare (interfeţe) între componente.

Ex. AADL, Acme, UniCon, Rapide

http://www.aadl.info, http://www.cs.cmu.edu/~acme/, http://www.cs.cmu.edu/~UniCon/, http://poset.stanford.edu/rapide/

Evaluare formativă

1. Enumerați principalele etape în proiectarea arhitecturii unui sistem software.

2. De ce este importantă crearea și evaluarea mai multor variante ale arhitecturii unui sistem software ?

https://forms.gle/emZJtt3uXXhqmAXG9

BIBLIOGRAFIE

Roger S. Pressman, **Software Engineering. A Practitioner's Approach** ed.6, McGraw-Hill International Edition, 2005

Fairbanks, G. **Just Enough Software Architecture, A Risk Driven Approach**, Marshall&Brainerd, 2010