

Oblikovanje programske potpore

2012./2013. grupa P01

Arhitektura programske potpore

Prof.dr.sc. Vlado Sruk



Sveučilište u Zagrebu

Fakultet elektrotehnike i računarstva

Zavod za elektroniku, mikroel., računalne i inteligentne sustave





- Definicija arhitekture programske potpore.
- Proces donošenja odluke izbora i oblikovanja arhitekture programske potpore.
- Kriteriji izbora arhitekture programske potpore.
- Struktura i sadržaj dokumenata oblikovanja arhitekture programske potpore.



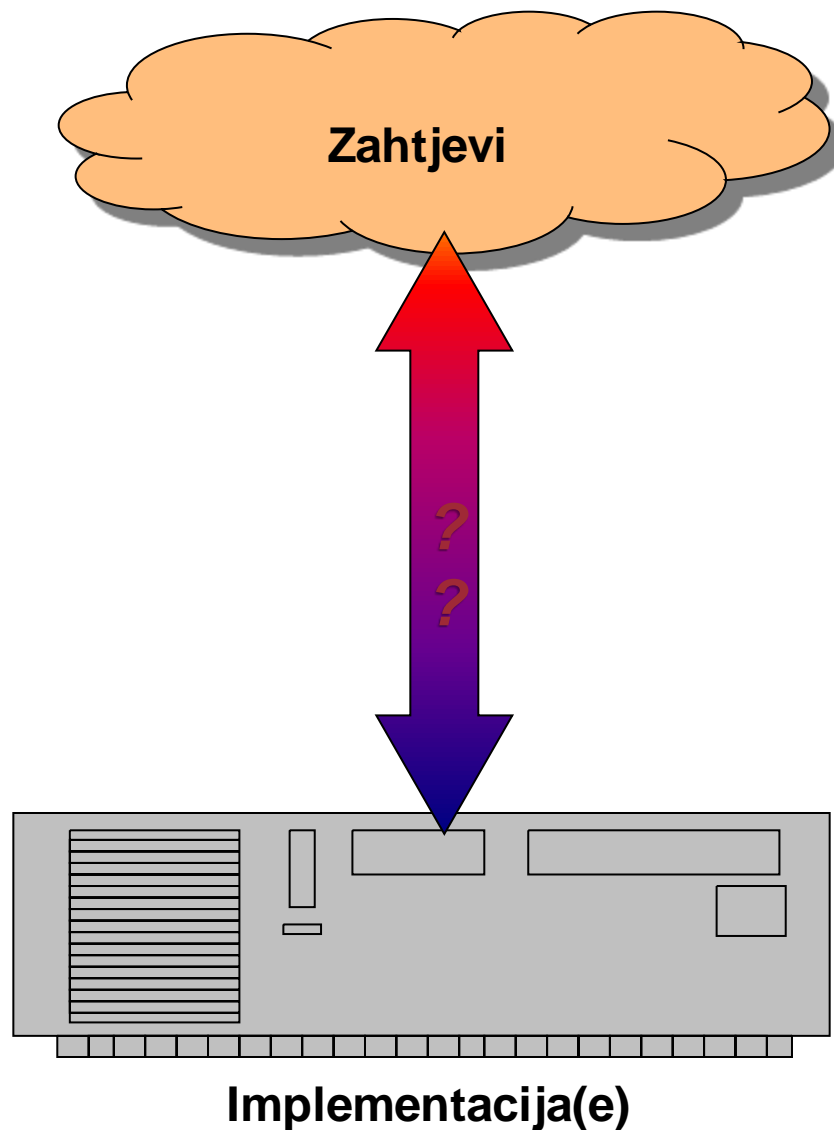
- Sommerville, I., ***Software engineering***, 8th ed., Addison-Wesley, 2007.
- Timothy C. Lethbridge, Robert Laganière, ***Object-Oriented Software Engineering: Practical Software Development using UML and Java***, Second Edition, McGraw Hill, 2001
- Mary Shaw, Paul Clements: ***The golden age of software architecture***, IEEE Software, vol 23, no 2, March/April 2006.
- *Software Engineering Institute*
 - Carnegie Mellon University, Pittsburgh, PA, USA
 - utemeljen 1984, zapošljava >300 ljudi (Pittsburgh, Pennsylvania, Arlington, Virginia, i Frankfurt)
 - <http://www.sei.cmu.edu/>





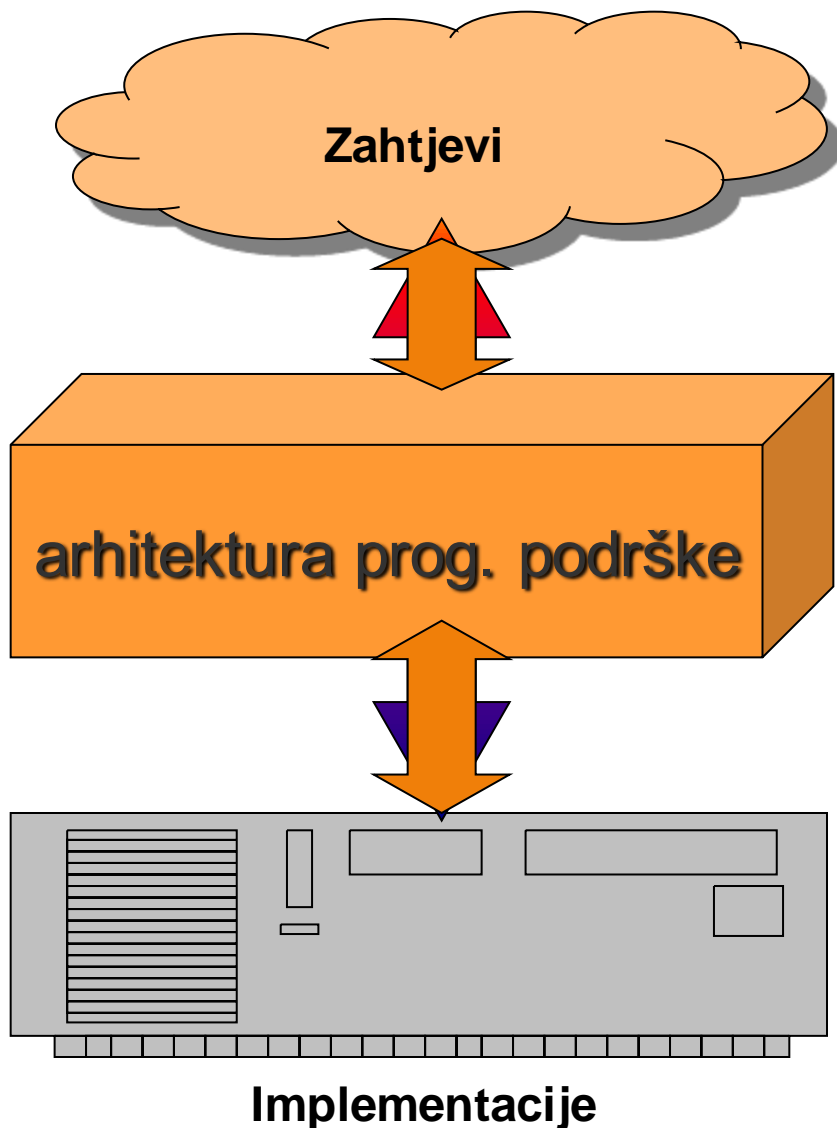
Od zahtjeva do koda

- Veliki raskorak između problema i rješenja





Uloga arhitekture prog. potpore



- Apstrakcija sustava na visokom nivou
 - komponente, konektori, ..
- Osnovni nositelj kvalitete sustava
- Strukturira razvojni projekt i samu programsku podršku
- Kapitalna investicija koja se može ponovno koristiti
- Osnova za komunikaciju dionika
- Izgrađuje se prije detaljne specifikacije



- 1968 - **Conwayov zakon** Melvin Conway,
 - “It concerns the structure of organizations and the corresponding structure of systems (particularly computer software) designed by those organizations.”
- Krajem 1980 - istraživanja u području arhitekture programske podrške s ciljem analize velikih programskih sustava
- U počecima temeljeno na kvalitativnim opisima empirički promatranih organizacija sustava razvija se i obuhvaća formalne opise, alate i tehnike analize.
 - od interpretacije prakse
 - konkretni napuci za analizu, oblikovanje i razvoj složene programske podrške
 - danas predstavlja osnovni element oblikovanja i izrade programskih sustava

Primjer: OO arhitektura prog. potpore





Razvoj programske potpore



- 1950 – programiranje na bilo koji način
- 1960 – potprogrami i zasebno prevođenje dijelova (*engl. programming in the small*). (Code-and-fix ,spaghetti coding)
- 1970 – apstraktni tipovi podataka, objekti, skrivanje informacije (*engl. programming in the large*).
- 1980 – razvojne okoline, cjevovodi i filtri
- 1990 – objektno usmjereni obrasci (*engl. patterns*), integrirana razvojna okruženja.
- 2000 – arhitektura programske potpore, jezici za oblikovanje (npr. **UML**) i metode oblikovanja (npr. modelno oblikovanje arhitekture – *engl. model driven architecture MDA*).



- Proces identificiranja i strukturiranja podsustava koji čine cjelinu te okruženja za upravljanje i komunikaciju između podsustava.
 - rezultat procesa oblikovanja je opis/dokumentacija *arhitekture programske potpore*.
- 1969 NATO Software Engineering conference: I.P. Sharp:

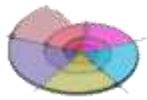
“I think we have something in addition to software engineering... This is the subject of software architecture. Architecture is different from engineering.”



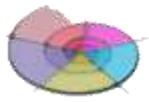
Prednosti definiranja arhitekture



- Smanjuje cijenu oblikovanja, razvoja i održavanja programskog produkta.
- Omogućuje ponovnu uporabu rješenja (*engl. **re-use***).
- Poboljšava razumljivost.
- Poboljšava kvalitetu produkta.
- Razjašnjava zahtjeve.
- Omogućuje donošenje temeljnih inženjerskih odluka.
- Omogućuje ranu analizu i uočavanje pogrešaka u oblikovanju.


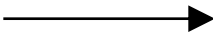


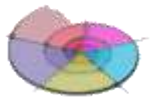
- “Just as good programmers *recognized useful data structures* in the late 1960s, good software system designers now *recognize useful system organizations*.”
 - garlan & Shaw: An Introduction to Software Architecture
- Dobar arhitekt:
 - razumije potrebe poslovnog modela i zahtjeve projekta.
 - svjestan različitih tehničkih pristupa u rješavanju danog problema.
 - evaluira dobre i loše strane tih pristupa.
 - preslikava potrebe i evaluirane zahtjeve u tehnički opis arhitekture programske potpore.
 - vodi razvojni tim u oblikovanju i implementaciji.
 - koristiti “meke” vještine kao i tehničke vještine.
- Pogled arhitekta na programsku potporu :
 - struktura kao skup implementacijskih zahtjeva
 - struktura i odnosi elemenata tijekom dinamičke interakcije
 - odnosi programskih struktura i okoline



Definicija arhitekture



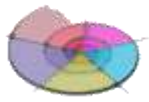
- Arhitektura programske potpore je struktura ili strukture sustava koji sadrži elemente, njihova izvana vidljiva obilježja i odnose između njih.
- Opis arhitekture programske potpore je skup dokumentiranih pogleda raznih dionika.
- **Pogled** predstavlja djelomično obilježje razmatrane arhitekture programa i dokumentiran je **dijagramom** (nacrtom) koji opisuje strukturu sustava i sadrži:
 - elementi/komponente: dijelovi sustava 
 - odnose između elemenata : topologija 
 - vanjski vidljiva obilježja
 - Veličina, performanse, sigurnost, API, ...



Definicija arhitekture



- “as *the size and complexity* of software systems increases, the design problem goes beyond the algorithms and data structures of the computation: *designing and specifying the overall system structure* emerges as a new kind of problem. *This is the software architecture level of design.*”
 - Garlan, 1992
- “The software architecture of a program or computing system is *the structure or structures of the system*, which comprise software components the externally visible *properties of those components*, and the *relationships* among them.”
 - Bass, Clements, and Kazman. *Software Architecture in Practice*, Addison-Wesley 1997
- *The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.*
 - The IEEE Standard for Architectural Description of Software-Intensive Systems (IEEE P1471/D5.3)



Model arhitekture

- Više pogleda u okviru nekog konteksta predstavljaju ***model arhitekture programske potpore***.
- Klasifikacija:
 - statičan strukturni model - Moduli
 - engl. ***module viewpoint***
 - pokazuju kompoziciju/dekompoziciju sustava
 - dinamički procesni model
 - engl. ***component and connector viewpoint***
 - komponente u izvođenju (engl. ***runtime***)
 - alocirani elementi
 - engl. ***allocation viewpoint***
 - Dokumentacija odnos programske potpore i razvojne/izvršne okoline
 - ...
- Arhitektura programske potpore opisuje se modelima koji svaki sadrži jedan ili više pogleda (dijagrama).



- Familije, *engl. Architecture style*
- U pojedinim modelima mogu se prepoznati često upotrebljavane forme i oblici
 - skupovi srodnih arhitektura.
 - u jednom programskom produktu može postojati kombinacija više stilova.
- Opisuju se:
 - rječnikom (tipovima komponenata i konektora).
 - topološkim ograničenjima koja moraju zadovoljiti svi članovi stila.
- Primjeri stilova:
 - protok podataka (*engl. data-flow*)
 - objektno usmjereni stil
 - repozitorij podataka
 - upravljan dogadajima
 - ...



■ **Koncepcijski** - *engl. Conceptual Architecture*

- usmjeravanje pažnje na pogodnu dekompoziciju sustava
- komunikacija s netehničkim dionicima (uprava, prodaja, korisnici)
- prikaz: UML arhitekturni dijagrami, Neformalna specifikacija komponenti CRC kartice

■ **Logički** - *engl. Logical Architecture*

- precizno dopunjena koncepcijska arhitektura
- detaljan nacrt pogodan za razvoj komponenti
- prikaz: UML arhitekturni dijagrami sa sučeljima, specifikacije komponenti i sučelja, komunikacijski dijagrami, potrebna objašnjenja diskusije

■ **Izvršni** - *engl. Execution Architecture*

- namijenjena distribuiranim i paralelnim sustavima
- pridruživanje procesa fizičkom sustavu



Problem dokumentata arhitekture

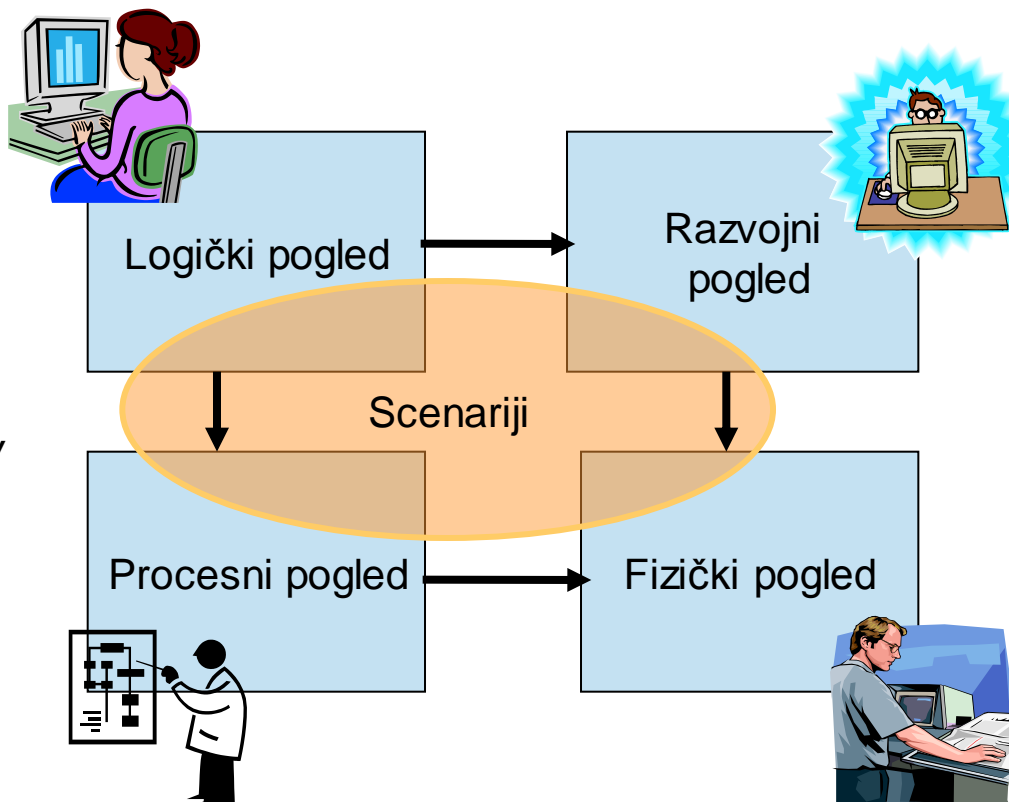


- Prenaglašavanje jedne dimenzije razvoja
- Programski inženjeri pokušavaju sveobuhvatno prikazati problem na jednom nacrtu
 - prekompleksan dokument
 - nerazumljiv dokument



Primjeri arhitekturnih pogleda

- 4+1 pogled
 - 1995 Kruchten, P.: **Architectural Blueprints—The “4+1” View Model of Software Architecture**, IEEE Software
- **Logički pogled** – *engl. Logic View*
 - ponašanje sustava i dekompozicija
- **Procesni pogled** – *engl. Process View*
 - opis procesa i njihove komunikacije
- **Fizički pogled** – *engl. Physical View*
 - instalacija i izvršavanje u mrežnom okruženju
- **Razvojni pogled** – *engl. Development View*
 - opisuje module sustava
- **Podatkovni pogled** – *engl. Data View*
 - tijek informacija u sustavu
-



Izvor: Kruchten P.: Architectural Blueprints—The “4+1” View Model of Software Architecture



Pogled: Komponente i konektori



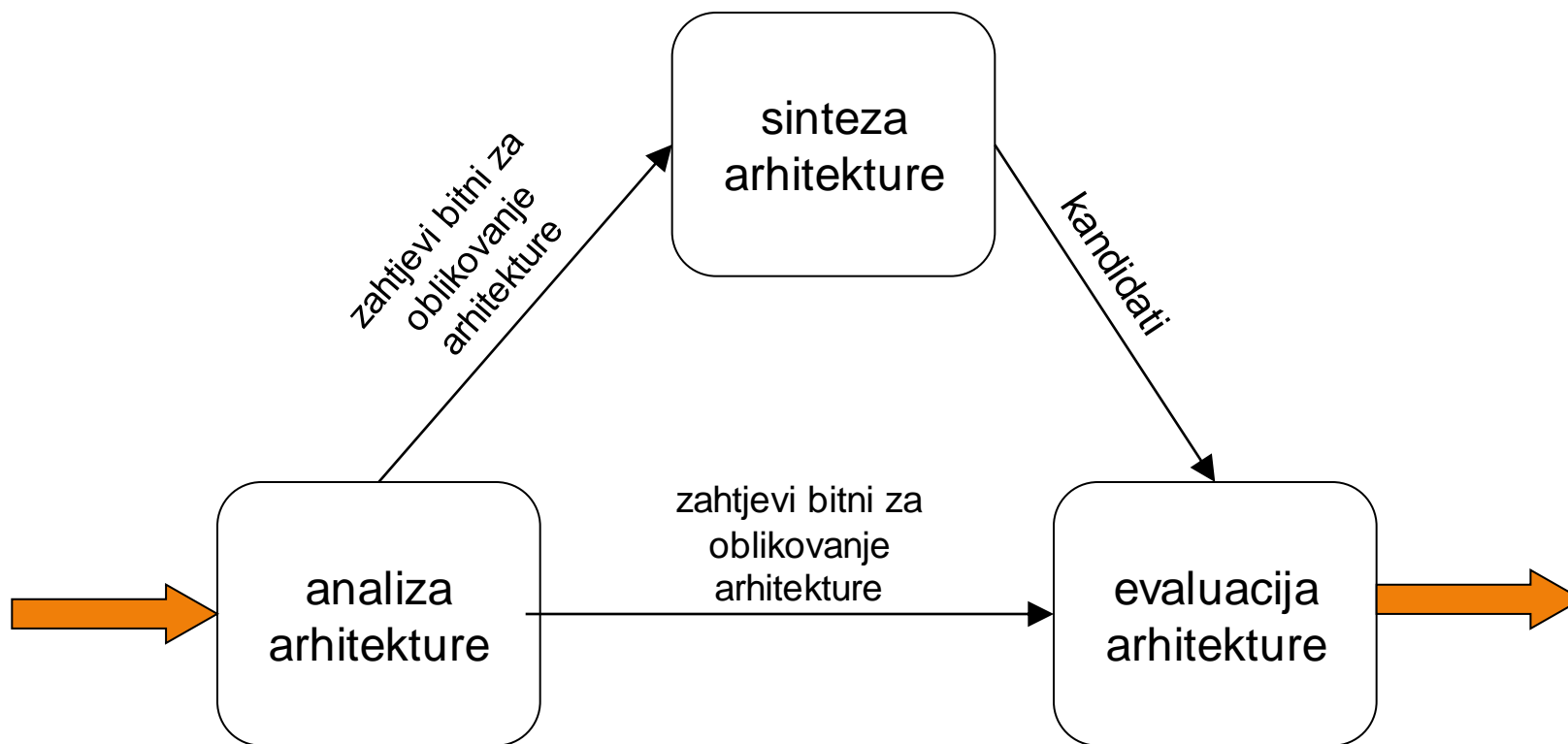
- *engl. Connector and Component*
- Opisuje ponašanje tijekom izvršavanja
- Dekompozicija sustava u komponente (Tipično hijerarhijska dekompozicija).
 - komponente:
 - osnovna jedinica izračunavanja i pohrane podataka
 - npr. objekti, procesi, klijent-poslužitelj.
 - konektori:
 - apstrakcija interakcije između komponenata
 - cjevovodi, repozitoriji, utičnice (*engl. socket*), udaljeni poziv procedure, posrednici (*engl. middleware*)
 - uporaba stila arhitekture:
 - oblikovanje kompozicije komponenata i konektora.
- Uvažiti ograničenja i moguće invarijante.



PROCES IZBORA I EVALUACIJE ARHITEKTURE PROGRAMSKE POTPORE



Aktivnosti oblikovanja





Proces izbora i evaluacije



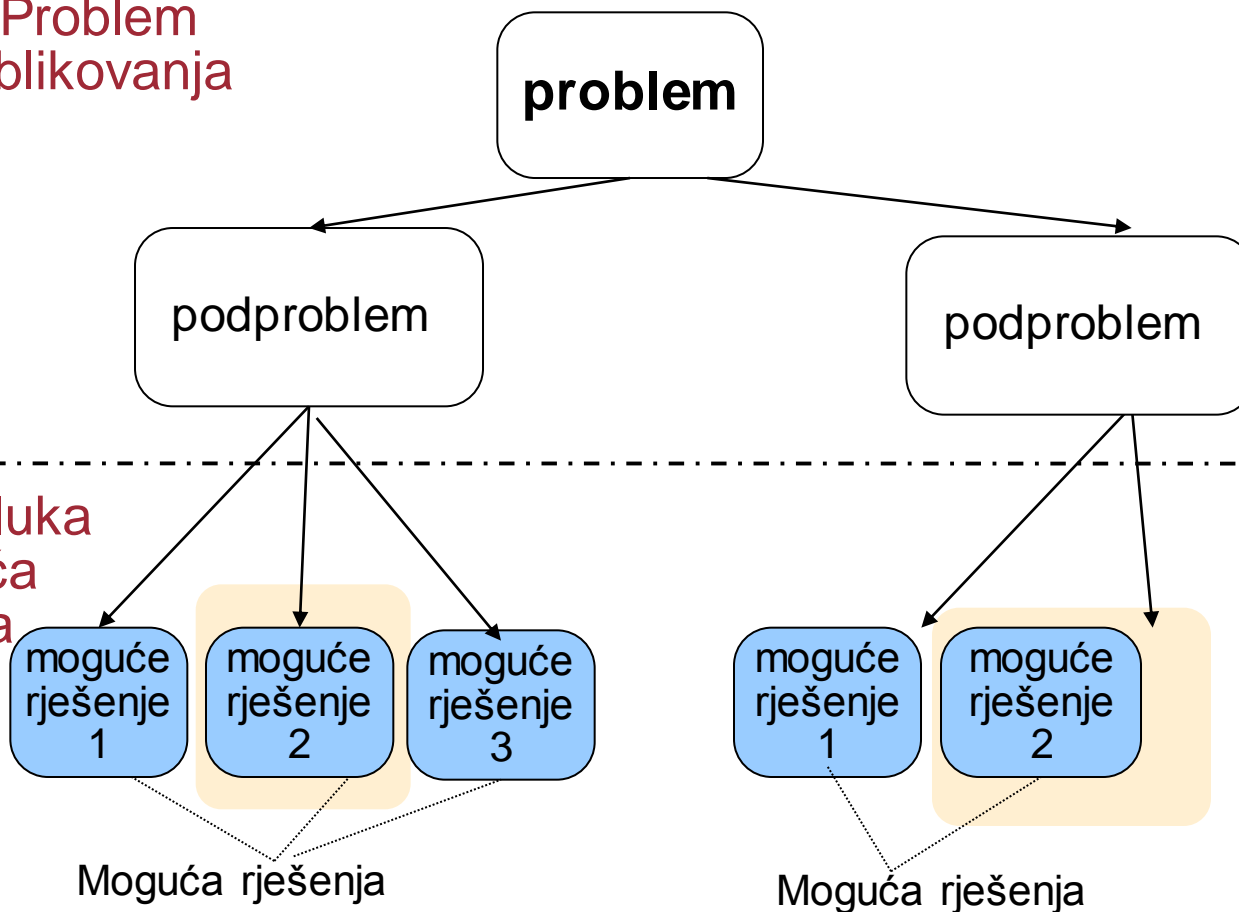
- Proces izbora i evaluacije arhitekture \equiv proces donošenja odluka
- Alternativni stilovi arhitekture programske potpore
 \Rightarrow Oblikovanje kao niz odluka
- Dizajner se sučeljava s rješavanjem niza problema (*engl. **design issues***)
 - podproblemi ukupnog problema
- Više inačica rješenja
 - *engl. **design options***
- Dizajner donosi odluke (*engl. **design decision***) za rješavanje problema
 - **odabir najbolje opcije između više mogućih rješenja problema**



Odluke

Problem
oblikovanja

Prostor odluka
– moguća
rješenja

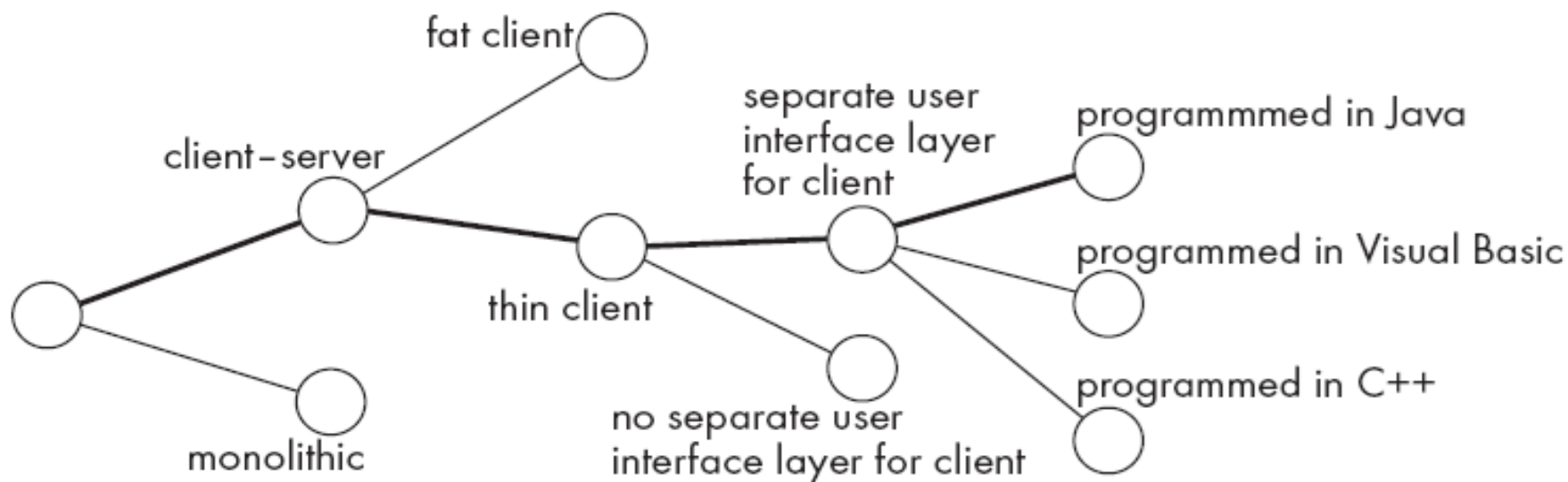




Prostor oblikovanja



- Prostor oblikovanja je **skup opcija** koja su na raspolaganju uporabom različitih izbora, *engl. design space*
- Primjer:



Izvor: Sommerville, I., Software engineering



Donošenje odluka



- Za donošenje odluka potrebna znanja
 - zahtjeva
 - trenutno oblikovana arhitektura
 - raspoloživa tehnologija
 - principi oblikovanja i najbolja praksa (*engl. **best practices***)
 - dobra rješenja iz prošlosti
- Zadaće donošenja odluka
 - postavljanje prioriteta sustava
 - dekompozicija sustava
 - definiranje svojstva sustava
 - postavljanje sustava u kontekst
 - integritet sustava
- Tehničke i netehničke pitanja su isprepletena!



Primjer okvira donošenja odluka



guide
architects



Meta-Architecture

- Architectural vision, principles, styles, key concepts and mechanisms
- *Focus: high-level decisions that will strongly influence the structure of the system; rules certain structural choices out, and guides selection decisions and tradeoffs among others*

Architecture

- Structures and relationships, static and dynamic views, assumptions and rationale
- *Focus: decomposition and allocation of responsibility, interface design, assignment to processes and threads*

Conceptual Architecture

Logical Architecture

Execution Architecture

guide
designers



Architecture Guidelines and Policies

- Use model and guidelines; policies, mechanisms and design patterns; frameworks, infrastructure and standards
- *Focus: guide engineers in creating designs that maintain the integrity of the architecture*



- *engl. Top-down design*
 - oblikuje najvišu strukturu sustava
 - postepeno razrađuj detalje
 - na kraju detaljne odluke:
 - Format podataka;
 - Uporaba/izbor algoritama

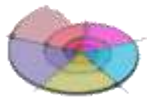


■ *engl. Bottom-up design*

- donošenje odluka o komponentama za ponovnu uporabu
- odluke o njihovoj uporabi za stvaranje komponenti više razine

■ *Hibridno oblikovanje*

- uporaba obje metode:
- oblikovanje od vrha prema dolje
 - Dobra struktura sustava
- oblikovanje od dna prema vrhu
 - Stvaranje komponenti pogodnih za ponovnu uporabu



- Oblikovanje arhitekture
 - podjela u podsustave i komponente
 - Način povezivanja?
 - Način međudjelovanja?
 - Sučelja?
- *Oblikovanje korisničkog sučelja*
- *Oblikovanje algoritma*
 - za izračunavanje, upravljanje ..
- *Oblikovanje protokola*
 - komunikacijski protokoli



Razvoj modela arhitekture



- Započni s grubom skicom arhitekture zasnovanoj na osnovnim zahtjevima i obrascima uporabe.
- Odredi temeljne potrebne komponente sustava.
- Izaberi između raznih stilova arhitekture
 - savjet: nekoliko timova nezavisno radi grubu skicu arhitekture, a potom se spoje najbolje ideje.
- Arhitekturu dopuni detaljima tako da se:
 - identificiraju osnovni načini komunikacije i interakcije između komponenata.
 - odredi kako će dijelovi podataka i funkcionalnosti raspodijeliti između komponenata.
 - pokušaj identificirati dijelove za ponovnu uporabu
 - vrati se na pojedini obrazac uporabe i podesi arhitekturu.



■ *Uporaba prioriteta i ciljeva za odabir alternativa*

1. pobroji i opiši alternative odluka oblikovanja
2. pobroji prednosti i nedostatke svake alternative u odnosu na prioritete i ciljeve
3. odredi sukob alternative koje su u sukobu s ciljevima
4. odaberi alternative koje najbolje zadovoljavaju ciljeve
5. prilagodi prioritete za daljine donošenje odluka

- Strukturne odluke (*engl. structural decisions*)
 - stvaranje podsustava, nivoa, komponenata ...
- Ponašajne odluke (*engl. behavioral decisions*)
 - formiranje interakcija u sustavu
- Odluke o svojstvima (naputci)
 - vodilje (*engl. design rules or guidelines*)
 - ograničenja (*engl. design constraints*)
- Izvršne odluke
 - poslovne odluke koje utječu na metodologiju razvoja, ljude, alate



Primjer prioriteta i ciljeva



- U području oblikovanja ***računalnog sustava:***
- ***Sigurnost/Security:*** Podaci se ne smiju moći dešifrirati poznatim tehnikama za < 100sati na 400Mhz Intel procesoru.
- ***Održavanje/Maintainability:*** Nema
- ***Performanse/CPU efficiency:*** Odziv < 1s na 400MHz Intel procesoru.
- ***Mrežno opterećenje/Network bandwidth efficiency:*** $\leq 8\text{KB}$ po transakciji.
- ***Memorijski resursi/Memory efficiency:*** $\leq 20\text{MB}$ RAM.
- ***Prenosivost/Portability:*** Windows XP, Linux



Analiza mogućih opcija



- Npr. 5 različitih opcija

	<i>Security</i>	<i>Maintainability</i>	<i>Memory efficiency</i>	<i>CPU efficiency</i>	<i>Bandwidth efficiency</i>	<i>Portability</i>
Algorithm A	High	Medium	High	Medium; DNMO	Low	Low
Algorithm B	High	High	Low	Medium; DNMO	Medium	Low
Algorithm C	High	High	High	Low; DNMO	High	Low
Algorithm D	—	—	—	Medium; DNMO	DNMO	—
Algorithm E	DNMO	—	—	Low; DNMO	—	—

Izvor: Sommerville, I., Software engineering



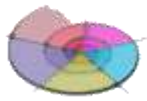
- *engl. Cost-Benefit Analysis*
- Utvrđivanje troškova novog sustava – *engl. Cost*
 - razvoj – *engl. capital expenditure - CAPEX*
 - redovni rad – *engl. operating expenditure - OPEX*
- Utvrđivanje koristi novog sustava – *engl. Benefits*
 - mjerljive i direktne - *engl. tangible*
 - teško mjerljive, posredne – *engl. intangible*
- Procjena CIJENE/troškova uključuje:
 - cijena rada programskog inženjera, uključujući održavanja
 - cijena uporabe razvojne tehnologije
 - cijena krajnjih korisnika i potpore
- Procjena koristi/dobiti uključuje:
 - uštedu vremena programskog inženjera
 - dobrobiti mjerene kroz povećanu prodaju ili ostale financijske uštede



- Za osiguranje održavanja i pouzdanosti oblikovana arhitektura mora biti stabilna
 - dodavanje novih karakteristika jednostavno s minimalnim promjenama u arhitekturi



- Započeti izradom kostura
 - koristiti osnovne zahtjeve i oblikovne obrasce
 - odrediti neophodne osnovne komponente
 - odabrati jedan od uzoraka arhitekture
- *Preporuka:*
 - koristiti različite timove
 - uzeti najbolje ideje



- Poboljšanje arhitekture
 - utvrđivanje osnovnih načina interakcije komponenti i odgovarajućih sučelja
 - odluka o raspoređivanju dijelova podataka i funkcionalnosti po komponentama
 - utvrđivanje mogućnosti ponovne uporabe postojećih arhitektura, mogućnost izgradnje nove arhitekture
- Posudba oblikovnih obrazaca i prilagodba arhitekture za njihovo ostvarenje
- Zrelost arhitekture



KRITERIJI ZA IZBOR ARHITEKTURE



Principi oblikovanja

1. **Podijeli i vladaj** – engl. *Divide and conquer*
2. **Povećaj koheziju** - engl. *Increase cohesion where possible*
3. **Smanji međuovisnost** - engl. *Reduce coupling where possible*
4. **Zadrži (višu) razinu apstrakcije** - engl. *Keep the level of abstraction as high as possible*
5. **Povećaj ponovnu uporabivost** - engl. *Increase reusability where possible*
6. **Povećaj uporabu postojećeg** - engl. *Reuse existing designs and code where possible*
7. **Oblikuj za fleksibilnost** - engl. *Design for Flexibility*
8. **Planiraj zastaru** - engl. *Anticipate Obsolescence*
9. **Oblikuj za prenosivost** - engl. *Design for Portability*
10. **Oblikuj za ispitivanje** - engl. *Design for Testability*
11. **Oblikuj konzervativno** - engl. *Design Defensively*
12. **Oblikuj po ugovoru** - engl. *Design by Contract*



1: Podijeli i vladaj

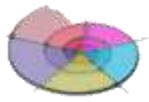
- *engl. Divide and conquer*
- Jednostavniji rad s više malih dijelova
 - odvojeni timovi rade na manjim problemima
 - omogućava specijalizaciju
 - manje komponente povećana razumljivost
 - olakšana zamjena dijelova
 - bez opsežne intervencije u cijeli sustav



Primjeri programskih sustava



- distribuirani sustavi – klijenti i poslužitelji
- podjela sustava u podsustave
- podjela podsustava u pakete
- podjela paketa u razrede



2: Povećanje kohezije



- *engl. Increase cohesion where possible*
- Podsustav ili modul ima veliku koheziju ako grupira međusobno povezane elemente, a sve ostalo stavlja izvan grupe
- Olakšava razumijevanje i promjene u sustavu
 - klasifikacija
 - Funkcijska – *engl. Functional*;
 - Razinska – *engl. Layer*;
 - Komunikacijska – *engl. Communicational*
 - Sekvencijska – *engl. Sequential*;
 - Proceduralna - *engl. Procedural*;
 - Vremenska – *engl. Temporal*;
 - Korisnička - *engl. Utility*



- *engl. **Functional cohesion***
- Kod koji obavlja pojedinu operaciju je grupiran, sve ostalo izvan
 - npr. Modul obavlja jednu operaciju, vraća rezultat bez popratnih efekata
- Prednosti:
 - olakšano razumijevanje
 - povećana ponovna uporabljivost modula
 - lakša zamjena
- Nefunkcionalna kohezija:
 - modul mijenja bazu podatka, kreira datoteku, interakcija s korisnikom

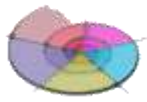


Razinska kohezija

- Kohezija u istoj razini, *engl. Layer cohesion*
- Svi resursi za pristup skupu povezanih servisa na jednom mjestu, sve ostalo izvan
 - razine formiraju hijerarhiju
 - Viša razina može pristupiti servisima niže razine
 - Niža razina ne pristupa višoj
 - skup procedura kojima pojedina razina omogućava pristup servisima naziva se aplikacijsko programsko sučelje – *engl. application programming interface (API)*
 - moguća zamjena pojedine razine bez utjecaja na ostale (više ili niže) razine



- *engl. Communicational cohesion*
- Svi moduli koji pristupaju ili mijenjaju određene podatke su grupirani, sve ostalo izvan
 - razred ima dobru komunikacijsku koheziju
 - Sadrži sve systemske usluge neophodne za rad sa podacima
 - Ne obavlja ništa drugo osim upravljanja podacima
 - prednost:
 - Prilikom promjene podatka sav kod na jednom mjestu



Sekvencijska kohezija



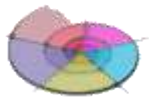
- engl. *Sequential cohesion*
- *Grupiranje procedura u kojoj jedna daje ulaz slijedećoj, sve ostale izvan*
 - postizanje sekvencijske kohezije provodi se nakon svih prethodnih kohezijskih tipova.



Proceduralna kohezija



- engl. *Procedural cohesion*
- *Procedure koje se upotrebljavaju jedna nakon druge*
 - ne moraju razmjenjivati informacije
 - slabija od sekvencijske



Vremenska kohezija



- engl. *Temporal Cohesion*
- *Operacije koje se obavljaju tijekom iste faze rada programa su grupirane, sve ostalo izvan*
 - podizanje sustava ili inicijalizacija
 - slabije od proceduralne kohezije



Kohezija pomoćnih programa



- *engl. **Utility cohesion***
- Povezani pomoćni programi (*engl. utilities*) koji se *logički ne mogu smjestiti u ostale grupe*
 - procedura ili razred koji je široko primjenjiv za različite sustave
 - npr. **java.lang.Math**



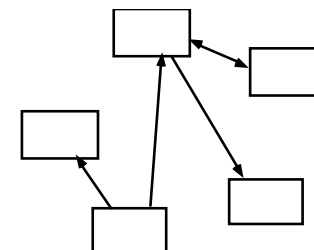
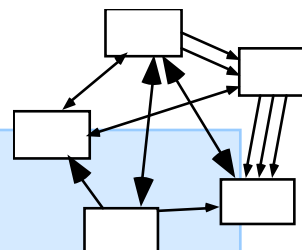
3: Smanjivanje međuovisnosti



- *engl. Reduce coupling where possible*
- *Povezivanje se javlja u slučaju međuovisnosti modula*
 - Međuovisnost \Rightarrow promjene na jednom mjestu zahtijevaju i promjene drugdje
 - Kod velike međuovisnosti teško je jasno raspoznati rad komponente.

- Tipovi međuovisnosti:

- Međuovisnost sadržaja – *engl. Content*
- Opća međuovisnost – *engl. Common*
- Upravljačka međuovisnost – *engl. Control*
- Međuovisnost u objektnom oblikovanju – *engl. Stamp*
- Podatkovna međuovisnost – *engl. Data*
- Povezivanje poziva procedura - *engl. Routine Call*
- Međuovisnost tipova - *engl. Type use*
- Međuovisnost uključivanjem - *engl. Inclusion/Import*
- Vanjska međuovisnost - *engl. External* _





Međuovisnost sadržaja



- *engl. Content coupling*
- Jedna komponenta prikriveno mijenja interne podatke druge komponente
- OO oblikovanje:
 - za smanjivanje međuovisnosti sadržaja koristi se enkapsulacija svih instanci varijabli
 - Deklarirati ih kao *private*
 - Osigurati *get* i *set metode*
 - najgori oblik međuovisnosti sadržaja javlja se kod izravne promjene varijable od instancirane varijable (višerazinska međuovisnost).



- *engl. Common coupling*
- Pri uporabi globalne varijable
 - sve komponente koje ju upotrebljavaju postaju povezane
 - slabiji oblik međuovisnosti je pristupanje varijabli preko podskupa klase sustav
 - Npr. Java package
 - globalne varijable mogu biti prihvatljive za postavljanje tipičnih vrijednosti sustava (*engl. default*).



- *engl. Control coupling*
- Izravna kontrola rada druge procedure uporabom zastavice ili naredbe
 - za promjenu potrebno mijenjati obje procedure
 - izbjegavanje
 - uporabom polimorfnih operacija u objektnom pristupu. Tijekom rada određuje se koju proceduru treba pozvati i kako se ona izvodi.
 - *look-up tablice*
 - Pridruživanje naredbi odgovarajućoj metodi



- *engl. Stamp coupling; Data-structured coupling*
- Javlja se kada složene podatkovne strukture dijeli više razreda a koristi samo jedan dio
 - npr. jedna razred deklariran kao tip argumenta metode
 - jedan razred upotrebljava drugi te na taj način otežava promjene sustava
 - Ponovna uporaba jednog razreda zahtijeva i ponovnu uporabu drugog
- načini smanjenja međuovisnosti
 - Uporaba sučelja kao argumenta tipa
 - Prijenos jednostavnijih varijabli

- *engl. Data coupling in OO design*
- Javlja se kada je tip metode argumenta primitiv ili jednostavnog razreda
 - međuovisnost se povećava s većim brojem argumenata metode
 - Sve metode koje ju koriste moraju prenijeti sve argumente
 - smanjenje nepotrebne uporabe argumenata smanjuje stupanj međuovisnosti
 - neophodan kompromis podatkovne i međuovisnosti podatkovnih struktura u objektnom oblikovanju
 - Povećanje međuovisnosti jednog tipa smanjuje drugi



Povezivanje poziva procedura



- *engl. Routine call coupling*
- Javlja se kada procedura ili metoda u OO sustavu poziva drugu
 - procedure time postaju povezane jer ovise o ponašanju druge
 - Uvijek prisutno u sustavima
 - česta uporaba niza dvije ili više procedura/metoda
 - Smanjenje povezivanja postiže se pisanjem jedinstvene procedure koja obuhvaća željeni niz



Međuovisnost tipova



- *engl. Type use coupling*
- Javlja se kada modul koristi podatkovni tip definiran u drugom modulu
 - kada razred deklarira instancu varijable ili lokalna varijabla ima tip drugog razreda
 - posljedica takve međuovisnosti je potreba za promjenom svih korisnika neke definicije pri njezinoj promjeni
 - izbjegavanje: Deklarirati tip varijable kao najopćenitiji razred ili sučelje koje sadrži zahtijevane operacije



Međuovisnost uključivanjem



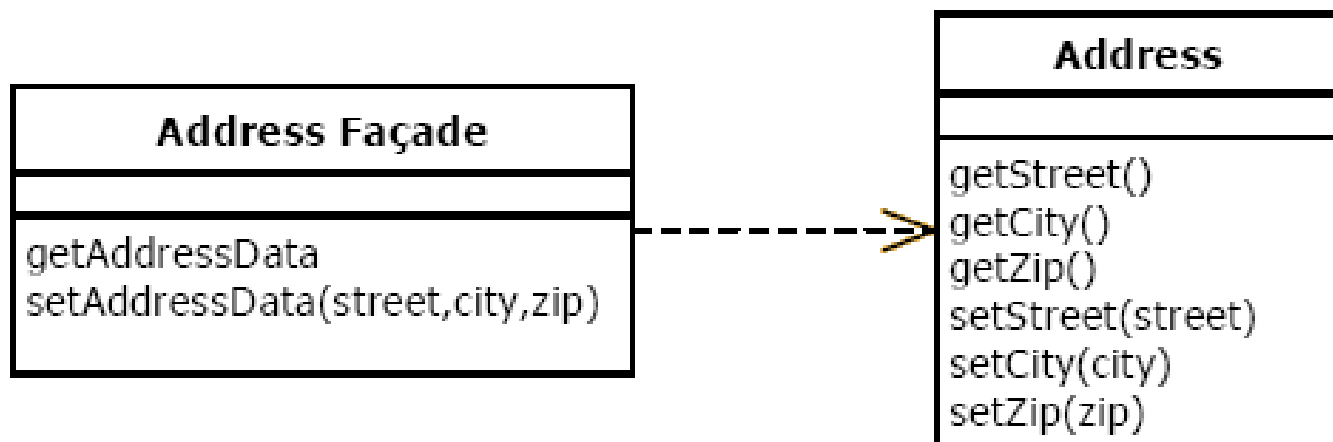
- *engl. Inclusion or import coupling*
- Javlja se kada komponenta importira paket
 - java
- Jedna komponenta uključuje drugu
 - C++
 - komponenta koja importira/uključuje drugu komponentu izložena je sadržaju komponenti koje je importirala/uključila
 - ako importirana/uključena komponenta nešto promjeni može doći do konflikta s komponentom koja importira/uključuje
 - ime nekog elementa uključene komponente može biti u konfliktu s imenom definiranim u komponenti koja uključuje



Vanjska međuovisnost



- *engl. External coupling*
- Predstavlja ovisnost modula o OS, biblioteci, HW, ...
 - minimizirati broj mjesta na kojima se javlja takva povezanost
 - u objektnom pristupu oblikovati malo sučelje prema vanjskim komponentama *engl. The Façade design pattern*





4: Zadrži razinu apstrakcije

- *engl. Keep the level of abstraction as high as possible*
- Osigurati da oblikovanje omogući sakrivanje ili odgodu razmatranja detalja, te na taj način smanji složenost
 - dobra apstrakcija podrazumijeva skrivanje informacija (*engl. information hiding*)
 - predstavlja temelj Objektno usmjerenog pristupa.
 - 1972. Parnas D.L.: *On Criteria to be Used in Decomposing Systems Into Modules.*
 - omogućava razumijevanje suštine podsustava bez poznavanja nepotrebnih detalja.

- *engl. Abstraction and classes (OO design)*
- Razredi/Klase su podatkovne apstrakcije koje sadrže proceduralne apstrakcije (metode)
 - razina apstrakcije se povećava definiranjem privatnih varijabli
 - one su nedostupne izvan dosega
 - apstrakcija se poboljšava smanjivanjem broja javnih metoda
 - superrazredi/Nasljeđivanja i sučelja (*engl. Superclass, interface*) povećava razinu apstrakcije
 - atributi i asocijacije predstavljaju podatkovne apstrakcije
 - dvije vrste varijabli instancija
 - metode predstavljaju proceduralne apstrakcije
 - Apstrakcija se povećava sa smanjivanjem broja argumenata procedura (metoda).



5: Povećaj ponovnu uporabivost



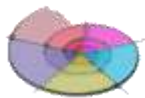
- *engl. Increase reusability where possible*
- Oblikovanje različitih aspekata sustava tako da može pridonijeti ponovnoj uporabi
 - poopćavanje oblikovanja u što većoj mjeri
 - uporaba prethodnih principa oblikovanja **Povećaj koheziju, Smanji povezanost, Zadrži razinu apstrakcije**
 - oblikuj sustav tako da sadrži kopče/sučelje (*engl. hooks*) koje omogućava pristup u program dodatnom korisničkom kodu
 - Korisnik vidi kopče kao otvore u kodu koji su dostupni u trenutku pojave nekog događaja ili zadovoljenja nekog uvjeta.
 - maksimalno pojednostavi oblikovanje



6:Povećaj uporabu postojećeg



- *engl. Reuse existing designs and code where possible*
- Princip je komplementaran povećanju ponovne uporabivosti.
 - što veća aktivna ponovna uporaba komponenti
 - smanjuje trošak i povećava stabilnost sustava
 - Korištenje prethodnih investicija
 - Kopiranje i umetanje, “kloniranje” (*engl. cloning*) se ne razmatra kao uporaba postojećeg dijela koda



7: Oblikuj za fleksibilnost



- *engl. Design for flexibility*
- Aktivno predviđaj buduće moguće promjene i provedi pripremu za njih
 - smanji povezivanje (međuviznost) i povećaj koheziju
 - stvaraj apstrakcije
 - ne upotrebljavaj izravno umetanje podataka ili konfiguracija u izvorni programski kod (*engl. hard code*)
 - ostavi otvorene opcije za eventualne modifikacije
 - upotrjebljavaj postojeći kod
 - Za kod kojeg radiš teži da zadovolji što lakšu i veću ponovnu uporabu



8: Planiraj zastaru

- *engl. Anticipate obsolescence*
- Planiraj promjene u tehnologiji ili okolini na taj način da program može raditi ili biti jednostavno promijenjen
 - izbjegavaj uporabu novih tehnologija ili njihovih novih inačica (*engl. release*)
 - izbjegavati knjižnice namijenjene specifičnim okolinama
 - izbjegavaj nedokumentirane ili rijetko upotrebljavane dijelove biblioteka
 - izbjegavaj SW/HW bez izgleda za dugotrajniju podršku
 - uporaba tehnologija i jezika podržanih od više dobavljača



9: Oblikuj za prenosivost



- *engl. Design for Portability*
- Omoguće rad na što većem broju različitih platformi
 - izbjegavaj specifičnosti neke okoline
 - Npr. specifičnosti okoline razvojnog okruženja



10: Oblikuj za ispitivanje



- *engl. **Design for Testability***
- Olakšaj ispitivanje
 - oblikuj program za automatsko testiranje
 - Omogući odvojeno pokretanje svih funkcija uporabom vanjskih programa (npr. bez grafičkog sučelja)
 - Npr. U Javi, u svakom razredu stvori `main()` metodu za jednostavnije ispitivanje drugih metoda.



11: Oblikuj konzervativno



- *engl. Design Defensively*
- Ne koristiti pretpostavke kako će netko koristiti oblikovanu komponentu
 - obradi sve slučajeve u kojima se komponenta može neprikladno upotrijebiti
 - provjeri valjanost ulaza u komponentu provjerom definiranih pretpostavki
 - Pretjerano obrambeno oblikovanje –često dovodi do nepotrebnih provjera



12. Oblikuj po ugovoru

- *engl. Design by contract*
- Tehnika koja omogućava efikasan i sistematski pristup konzervativnom oblikovanju
 - osnovna ideja
 - Sve metode imaju ugovor s pozivateljima
 - ugovaratelj ima skup zahtjeva:
 - **Preduvjete** koje mora ispuniti pozvana metoda kada započinje izvođenje
 - **Uvjete** koje pozvana metoda mora osigurati kod završetka izvođenja
 - **Invarijante** na koje *pozvana metoda neće djelovati pri izvođenju*



DOKUMENTI OBLIKOVANJA ARHITEKTURE PROGRAMSKE POTPORE



- Zašto dokumentirati?
 - zabilježiti odluke arhitekta. Kompletnost i jednoznačnost.
 - priopćavanje arhitekture. Komunikacija dionika.
- Potrebno zbog rane analize sustava.
- Temeljni nositelj obilježja kvalitete.
- Ključ za održavanje, poboljšanja i izmjene PP
- Dokumentacija ne zastarijeva
 - dugoročno govori umjesto arhitekta (ukoliko je sustav oblikovan, održavan i mijenjan sukladno dokumentaciji).
- U praksi je danas dokumentacija često nejednoznačna i kontradiktorna.
 - najčešće samo pravokutnici i linije koje mogu značiti: A šalje upravljačke signale do B, A šalje podatke do B, A šalje poruku do B, A kreira B, A dobavlja vrijednost od B, ...

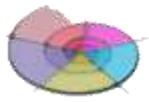
- Dokumenti oblikovanja pomoć izradi boljih dizajna
 - traže izričitost i obrađuju najvažnija pitanja prije faze implementacije
 - omogućuju vrednovanje oblikovanja i poboljšanje
 - sredstvo komunikacije
 - Tima za implementaciju
 - Budućih osoba zaduženih za održavanje, razvoj ili promjene
 - Osobama uključenih u oblikovanje povezanih sustava ili podsustava
 - pisati ih s gledišta čitatelja
 - upotrebljavati standardnu organizaciju

- Referentna specifikacija – *engl. Reference Specification*
 - potpuni skup dokumentiranih pokretača arhitekture (*engl. architecture drivers*), pogleda te pomoćne dokumentacije poput matrica odluka.
- Pregled za upravo - *engl. Management Overview*
 - pregled visokog nivoa, vizija sustava, poslovni motivi, koncepti arhitekturnih dijagrama, poveznice poslovne i tehničke strategije
- Dokumentacija komponenti - *engl. Component Documents*
 - za komponente se osigurava pogled na nivou sustava (*engl. Logical Architecture Diagram*), specifikacija komponente, specifikacija sučelja te dijagrami suradnje



Struktura dokumenta oblikovanja

- A. Svrha - *engl. Purpose*
 - koji sustav ili dio sustava ovaj dokument opisuje
 - te označi poveznice prema dokumentu zahtjeva na koji se ovaj dokument oslanja - sljedivost
- B. Opći prioriteti - *engl. General priorities*
 - opiši prioritete koji su vodili proces oblikovanja (npr. 4+1 pogled ...)
- C. Skica sustava - *engl. Outline of the design*
 - navedi opis sustava s najviše razine promatranja kako bi čitatelj razumio osnovnu ideju
- D. Temeljna pitanja u oblikovanju - *engl. Major design issues*
 - diskutiraj osnovne probleme koji su se morali razriješiti
 - navedi razmatrane opsijska rješenja, konačnu odluku i razloge za njeno donošenje
- E. Detalji oblikovanja - *engl. Other details of the design*
 - predoči sve ostale detalje koji su čitatelju zanimljivi a u dokumentu još nisu razmatrani



Pisanje dokumentacije



- Preporuka u sadržaju dokumenta oblikovanja:
 - izbjegavati dokumentiranje informacija očitih iskusnim programerima i dizajnerima
 - ne pisati detalje koji su dio komentara koda
 - ne pisati detalje koji su vidljivi u strukturi koda
- Arhitektura programske podrške mora rezultirati dokumentacijom koja ima slijedeća svojstva:
 - ***dobra***
 - Tehnički ispravna i jasno prezentirana
 - ***ispravna***
 - Doseže potrebe i ciljeve ključnih dionika
 - ***uspješna***
 - Upotrebljava se u stvarnom razvoju sustava kojim se postižu strateške prednosti



Diskusija



10 čimbenika neuspješnosti projekta

- Nekompletni zahtjevi i specifikacije
- Nedostatni angažman korisnika
- Nedostatak resursa
- Nerealna očekivanja
- Nedostatak potpore rukovoditelja
- Promjene zahtjeva i specifikacija
- Nedostatak planiranja
- Nepotrebni
- Nedostatak IT rukovoditelja
- Tehnološka nepismenost

10 čimbenika uspješnosti projekta

- Uključivanje korisnika
- Potpora rukovoditelja
- Jasno postavljene zahtjevi
- Prikladno planiranje
- Stvarno postavljena očekivanja
- Kraće projektne točke – *engl. smaller project milestones*
- Kompetentni ljudski resursi
- Vlasništvo
- Jasna vizija i ciljevi
- Puno rada



- Parkirni aparat je namijenjen prodaji parkirnih karata na samom parkiralištu.
 - na prednjoj strani uređaja nalazi se LCD zaslon na kojem je iskazano točno vrijeme, datum i parking zona u kojoj se nalazi uređaj, te pisačem za ispis karata.
 - predviđen je da radi potpuno samostalno.
 - ukoliko aparat otkrije bilo kakav problem sa svojim radom uključuje se proces dojave GSM modulom.
 - uređaj se napaja preko akumulatora koji mu daje autonomiju rada od 20 do 30 dana bez dopunjavanja preko solarnih ćelija ili iz električne mreže.
 - opremljen je čitačem papirnatih i kovanih novčanica (prima papirnatih novčanice i vraća kovanice).
 - naplata parkinga omogućena je uporabom bankovnih kartica do 50 kn bez autorizacije, a za veće iznose s autorizacijom te je u tu svrhu opremljen odgovarajućim čitačem magnetskih i pametnih kartica.
- Odgovarajućim obrascem uporabe i sekvencijskim dijagramom opišite plaćanje parkinga bankovnim karticama.
- Odgovarajućim obrascem uporabe i sekvencijskim dijagramom opišite gotovinsko plaćanje parkinga.