Druga nastavna cjelina:

ARHITEKTURA PROGRAMSKE POTPORE

Generičke aktivnosti u procesu programskog inženjerstva (SE):

- Specifikacija (temeljem analize zahtjeva)
- **Razvoj i oblikovanje**
- Validacija i verifikacija
- ~~Evolucija~~

1

Oblikovanje arhitekture programske potpore:

- To je proces **identificiranja** podsustava koji čine cjelinu te okruženja za upravljanje i komunikaciju između podsustava.
- Rezultat procesa oblikovanja je **opis** (**dokumentiranje**) **arhitekture programske potpore**.

Ciljevi ove i daljnjih prezentacija:

- Uvesti pojmove arhitekture programske potpore.
- Definirati kriterije za izbor arhitekture.
- Objasniti odluke u izboru i oblikovanju arhitekture programske potpore.
- Definirati strukturu i sadržaj dokumenata oblikovanja.
- Upoznati se s referentnim stilovima arhitekture programske potpore s naglaskom na njihove značajke.

2

Prednosti definiranja arhitekture:

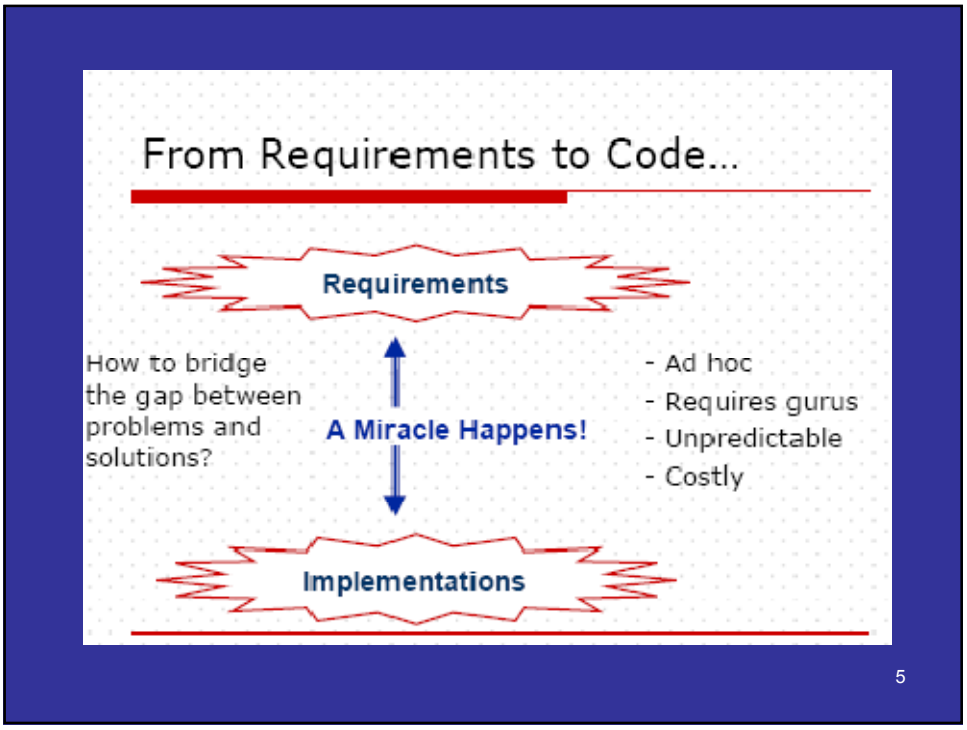
- Smanjuje cijenu oblikovanja, razvoja i održavanja programskog produkta.
- Omogućuje ponovnu uporabu rješenja (*engl. reuse*), što agilni pristup ne podržava.
- Poboljšava razumljivost.
- Poboljšava kvalitetu produkta.
- Razjašnjava zahtjeve.
- Omogućuje donošenje temeljnih inženjerskih odluka.
- Omogućuje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

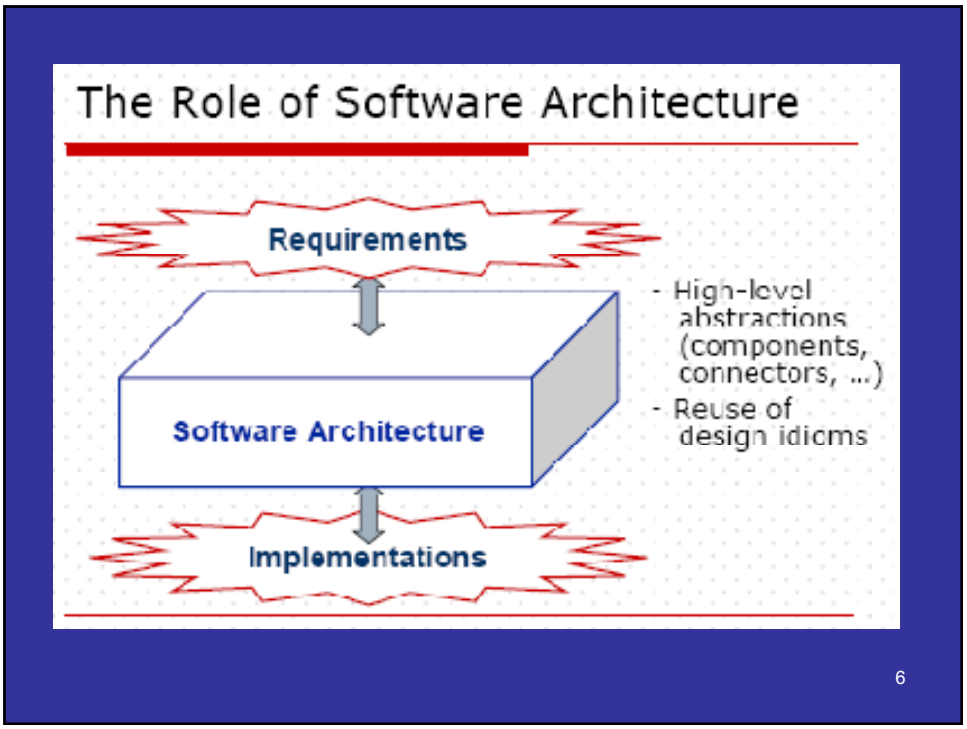
3

Izvori:

- David Garlan, Carnagie Mellon University (slides)
- Ian Sommerville, Software Engineering, 8th, ed.
- T.C. Lethbridge, R-Laganiere, Object-Oriented Software Engineering, 2nd. ed.

4


From Requirements to Code...
Requirements
How to bridge the gap between problems and solutions?
A Miracle Happens!
- Ad hoc
- Requires gurus
- Unpredictable
- Costly
Implementations
5


The Role of Software Architecture
Requirements
Software Architecture
Implementations
- High-level abstractions (components, connectors, ...)
- Reuse of design idioms
6

ULOGA ARHITEKTA:

- Razumjeti potrebe poslovnog modela i zahtjeve projekta.
- Biti svjestan različitih tehničkih pristupa u rješavanju danog problema.
- Evaluirati dobre i loše strane tih pristupa.
- Preslikati potrebe i evaluirane zahtjeve u tehnički opis arhitekture programske potpore.
- Voditi razvojni tim u oblikovanju i implementaciji.
- Koristiti "meke" vještine (manje formalne) kao i stroge tehničke vještine.

7

Što je arhitektura ?

Arhitektura programske potpore je **struktura ili strukture** sustava koji sadrži elemente, njihova izvana vidljiva **obilježja** i **odnose** između njih.

Koje vrste struktura ? (ima ih više)

- Moduli (pokazuju statičku kompoziciju/dekompoziciju sustava).
- Dinamičku kompoziciju, t.j. komponente u izvođenju (*engl. runtime*).
- Alocirane elemente (npr. datoteke).
- …

Svaka vrst strukture čini jedan **arhitekturni pogled**.

Opis arhitekture sadrži **višestruke poglede**.

8

Primjer pogleda: Komponente i Konektori

- Dekompozicija sustava u komponente.

  Komponente:

  osnovna jedinica izračunavanja i pohrane podataka (npr. klijent-uslužitelj)

  Tipično hijerarhijska dekompozicija.

- Konektori:

  Apstrakcija interakcije između komponenata (npr. poziv procedure).

- Uporaba nekog stila arhitekture na pogled komponenata i konektora usredotočuje se na:

  Oblikovanje kompozicije komponenata i konektora.

  Respektirati ograničenja i invarijante.

9

Stilovi (familije) arhitektura programske potpore

Skupovi srodnih arhitektura. U jednom programskom produktu može postojati kombinacija više stilova.

Opisuju se:

Rječnikom (npr. tipovima komponenata i konektora).

Topološkim ograničenjima koja moraju zadovoljiti svi članovi familije (stila).

Primjeri stilova:

- protok podataka (*engl. data-flow*)
- objektno usmjereni stil
- repozitorij podataka
- arhitektura upravljana događajima
- …

10

Arhitektura programske potpore u kontekstu:

1950 – programiranje na bilo koji način

1960 – subrutine i posebno prevođenje dijelova (*engl. programming in the small*).

1970 – apstraktni tipovi podataka, objekti, skrivanje informacije (*engl. programming in the large*).

1980 – razvojne okoline, "cjevovodi i filtri" - UNIX

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*).

2000 + stalna potreba za oblikovanjem stabilne arhitekture (nove značajke mogu jednostavno dodavati uz vrlo male izmjene). Time se osigurava veća pouzdanost i mogućnost jednostavnog održavanja sustava.

11

PROCES OBLIKOVANJA (ARHITEKTURE) PROGRAMSKE POTPORE

(osnovice za donošenje ključnih odluka)

12

Oblikovanje kao niz odluka

- Osoba zadužena za oblikovanje suočena je s nizom pitanja.
- Ta pitanja su podproblemi cjelokupnog oblikovanja.
- Svako pitanje ima nekoliko mogućih odgovora (opcija).
- Osoba zadužena za oblikovanje mora donijeti niz odluka kao odgovora na moguće alternativne opcije.

Da bi donio odluke programski inženjer koristi znanja o:

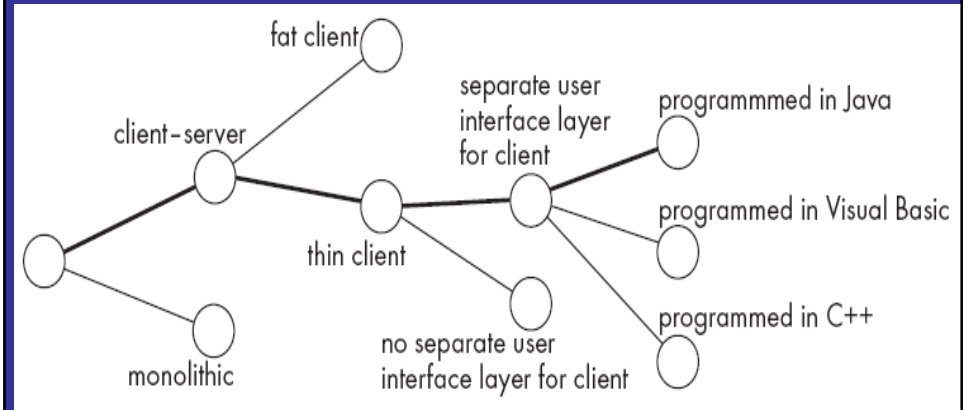
- Korisnički zahtjevi i zahtjevi sustava (rezultat inženjerstva zahtjeva).
- Rezultati dosadašnjeg procesa oblikovanja.
- Obilježja dostupne tehnologije.
- Principi i najbolja praksa programskog inženjerstva.
- Oblikovanja koja su u prošlosti bila uspješna.

13

Prostor oblikovanja

To je prostor svih mogućih oblikovanja sustava koji nastaje temeljem mogući alternativnih odluka.

Npr.



14

Oblikovanje odozgo prema dolje ili odozdo prema gore

Odozgo-prema-dolje

- Najprije se oblikuje struktura sustava na najvišoj apstraktnoj razini.
- Nakon toga se postepeno spušta na detaljne odluke o konstrukcijama nižih razina.
- Konačno se dosegnu detaljne odluke kao:
  - formati određenih podatkovnih jedinica
  - izbor individualnih algoritama

Odozdo-prema-gore

- Najprije se donose odluke o izboru funkcionalnih dijelova najniže razine koji se mogu višestruko koristiti.
- Nakon toga se odlučuje kako se ti dijelovi integriraju u konstrukcije viših razina.

15

Najčešće se koristi mješavina oba pristupa:

- Pristup odozgo prema dolje daje sustavu dobru strukturu.
- Pristup odozdo prema gore omogućuje kreaciju i uporabu komponenata za ponovno korištenje.

Različiti aspekti oblikovanja:

- Oblikovanje arhitekture (dijeljenje u podsustave i komponente, odluke kako se povezuju pojedini dijelovi, odluke o interakciji pojedinih dijelova, odluke o sučeljima pojedinih dijelova). O tome će biti riječi u ovom kolegiju.
- Oblikovanje korisničkog sučelja.
- Oblikovanje algoritama (izračunskih mehanizama).
- Oblikovanje komunikacijskih protokola.

16

Razvoj modela arhitekture

- **Počni grubom skicom arhitekture zasnovanoj na osnovnim zahtjevima i obrascima uporabe.**
- **Odredi temeljne potrebne komponente sustava.**
- **Izaberi između raznih stilova arhitekture (to će se razmatrati u nastavku).**
- **Savjet: neka nekoliko timova nezavisno izrade 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 (engl. *reuse*).**
  - **vrati se na pojedini obrazac uporabe i podesi arhitekturu.**

17

Uporaba prioriteta i ciljeva za odluku između pojedinih opcija u oblikovanju

Korak 1: Izlistaj i opiši opcije za donošenje odluka u oblikovanju.

Korak 2: Izlistaj prednosti i nedostatke svake opcije s obzirom na ciljeve i prioritete.

Korak 3: Odredi koja opcija onemogućuje ostvarivanje jednog ili više ciljeva.

Korak 4: Izaberi opciju koja najbolje pokriva ciljeve.

Korak 5: Podesi prioritete za slijedeći odluku u procesu oblikovanja.

18

Uporaba analize cijena-korist

(*engl. cost- benefit analysis*)

za odluku između pojedinih opcija arhitektura.

Za izračun cijene zbroji:

- **Inkrementalnu cijenu rada u programskom inženjerstvu, uključujući održavanje.**
- **Inkrementalnu cijenu tehnologije za postupak oblikovanja.**
- **Inkrementalnu cijenu koju će morati platiti krajnji korisnik i osoblje za održavanje tijekom uporabe programskog produkta.**

Za izračun koristi zbroji:

- **Inkrementalno smanjeno vrijeme programskih inženjera tijekom oblikovanja.**
- **Inkrementalna novčana korist od povećanje prodaje ili financijska korist korisnika.**

19

PRINCIPI I SMJERNICE U OBLIKOVANJU ARHITEKTURE PROGRAMSKE POTPORE

(služe i za evaluaciju pogodnosti pojedinih stilova arhitekture za određenu primjenu)

20

Design Principle 1: Divide and conquer

(Podijeli i vladaj)

Trying to deal with something big all at once is normally much harder than dealing with a series of smaller things

- Separate people can work on each part.
- An individual software engineer can specialize.
- Each individual component is smaller, and therefore easier to understand.
- Parts can be replaced or changed without having to replace or extensively change other parts.

Design Principle 2: Increase cohesion where possible

(Povećaj koheziju)

A subsystem or module has high cohesion if it keeps together things that are related to each other, and keeps out other things

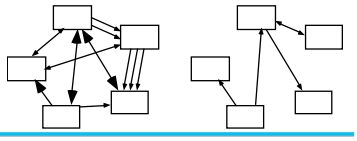
- This makes the system as a whole easier to understand and change.

Design Principle 3: Reduce coupling where possible

(Smanji međuovisnost)

***Coupling* occurs when there are *interdependencies* between one module and another**

- When interdependencies exist, changes in one place will require changes somewhere else.
- A network of interdependencies makes it hard to see at a glance how some component works.



Design Principle 4: Keep the level of abstraction as high as possible

(Povećaj razinu apstrakcije)

Ensure that your designs allow you to hide or defer consideration of details, thus reducing complexity

- A good abstraction is said to provide *information hiding*
- Abstractions allow you to understand the essence of a subsystem without having to know unnecessary details

Design Principle 5: Increase reusability where possible

(Oblikuj sustav tako da se omogući ponovna uporabivost njegovih dijelova)

Design the various aspects of your system so that they can be used again in other contexts

- Generalize your design as much as possible
- Follow the preceding three design principles
- Design your system to contain hooks **(kopče)**
- Simplify your design as much as possible

(A hook offers the opportunity to execute one or more user-supplied program fragments at some designated place in a program. From a user's point of view, hooks may be regarded as openings of a program, which are effective when some event happens, or if some condition is fulfilled.)

Design Principle 6: Reuse existing designs and code where possible

(Uporabi postojeće gotove komponente)

Design with reuse is complementary to design for reusability

- Actively reusing designs or code allows you to take advantage of the investment you or others have made in reusable components
  - —*Cloning* should not be seen as a form of reuse

(Clonning = ponovno pisanje istih linija koda)

Design Principle 7: Design for flexibility

(Oblikuj za fleksibilnost)

Actively anticipate changes that a design may have to undergo in the future, and prepare for them

- Reduce coupling and increase cohesion
- Create abstractions
- Do not hard-code anything
- Leave all options open
  - —Do not restrict the options of people who have to modify the system later
- Use reusable code and make code reusable

Design Principle 8: Anticipate obsolescence

(Anticipiraj zastaru)

Plan for changes in the technology or environment so the software will continue to run or can be easily changed

- Avoid using early releases of technology
- Avoid using software libraries that are specific to particular environments
- Avoid using undocumented features or little-used features of software libraries
- Avoid using software or special hardware from companies that are less likely to provide long-term support
- Use standard languages and technologies that are supported by multiple vendors

Design Principle 9: Design for Portability

(Oblikuj za prenosivost)

Have the software run on as many platforms as possible

- Avoid the use of facilities that are specific to one particular environment
- E.g. a library only available in Microsoft Windows

Design Principle 10: Design for Testability

(Oblikuj za jednostavno ispitivanje)

Take steps to make testing easier

- Design a program to automatically test the software
  - —Ensure that all the functionality of the code can by driven by an external program, bypassing a graphical user interface

Design Principle 11: Design defensively

(Oblikuj konzervativno)

Never trust how others will try to use a component you are designing

- Handle all cases where other code might attempt to use your component inappropriately
- Check that all of the inputs to your component are valid: the *preconditions*
  (preduvjeti)

Design Principle 12: Design by contract

(Oblikuj ugovorno)

A technique that allows you to design defensively in an efficient and systematic way

- Key idea
  - —each method has an explicit *contract* with its callers
- The contract has a set of assertions that state:
  - —What *preconditions* the called method requires to be true when it starts executing
  - —What *postconditions* the called method agrees to ensure are true when it finishes executing
  - —What *invariants* the called method agrees will not change as it executes

DOKUMENTIRANJE ARHITEKTURE - motivacija

- Potrebno zbog rane analize sustava.
- Temeljni nositelj obilježja kvalitete.
- Ključ za održavanje, poboljšanja i izmjene nakon puštanja u rad.
- Dokumentacija govori umjesto arhitekta danas i nakon 20 godina (ukoliko je sustav oblikovan, održavan i mijenjan sukladno dokumentaciji).
- U praksi je danas dokumentacija nejednoznačna i često kontradiktorna. Najčešće samo pravokutnici i linije koje mogu npr. 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, …

33

DOKUMENTIRANJE ARHITEKTURE:

(Struktura dokumenta oblikovanja)

1. Svrha (koji sustav ili dio sustava ovaj dokument opisuje te označi referenciju prema dokumentu zahtjeva na koji se ovaj dokument oslanja - slijedivost).
2. Opći prioriteti (opiši prioritete koji su vodili proces oblikovanja).
3. Skica sustava (navedi opis s najviše razine promatranja kako bi čitatelj razumio osnovnu ideju).
4. Temeljna pitanja u oblikovanju (diskutiraj osnovne probleme koji se moraju razriješiti, navedi razmatrana alternative rješenja, konačnu odluku i razloge za njeno donošenje).
5. Detalji oblikovanja (koji u dokumentu još nisu razmatrani).

Izbjegavaj: a) opis informacije koja je očigledna, b) informacije koja bi bolje pristajala komentiranju koda, c) pisanje detalja koji se automatizirano mogu izvući iz koda.

34