Oblikovanje programske potpore

2012./2013. grupa P01

Uvod i motivacija

Prof.dr.sc. Vlado Sruk

Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva
Zavod za elektroniku, mikroel., računalne i inteligentne sustave

Pregled tema

- Uvod
- Sadržaj kolegija
- Literatura
- Administracija
  - Organizacija nastave
  - Ocjenjivanje
- Problemi razvoja programske potpore
- Oblikovanje programske potpore
- Uvod u programsko inženjerstvo

Oblikovanje programske potpore

- *engl. Software Design*

- Studij : Računarstvo
  - Redovni predmet za module
    - Obradba informacija i multimedijske tehnologije
    - Programsko inženjerstvo
    - Računalno inženjerstvo
    - Računarska znanost
    - Telekomunikacije i informatika

- Semestar : 5
- ECTS : 8

- Nositelji:
  - Nikola.Bogunovic@fer.hr (D-309) - koordinator
  - Vlado.Sruk@fer.hr (D-332)
  - Alan.Jovic@fer.hr (D-340)

Asistenti, informacije

- Asistenti:
  - Danko.Ivosevic@fer.hr (D-344)
  - Alan.Jovic@fer.hr (D-340)
  - Marko.Horvat@fer.hr (C02-02)

- Informacije:
- Sve obavijesti u svezi predmeta biti će objavljene na web stranici:
  - http://www.fer.unizg.hr/predmet/opp
- Vremenske aktivnosti:
  - Google kalendar
- Projekt
- Predavanja
  - Moodle
  - http://www.zemris.fer.hr/predmeti/opp/

Konzultacije

- Pauze predavanja
  - Srijeda, 08-10 (14-16), B4
  - Četvrtak, 10-12, (16-18) B2
- Utorak 10:00-10:30
  - najava e-pošta: vlado.sruk@fer.hr
    - Naslov/Subject: [OPP] Konzultacije
  - soba D-332 (D zgrada treći kat)
- Konzultacije asistenti: FER web stranice

Organizacija predmeta

- ***Predavanja*** su organizirana u dva ciklusa (7+6 tjedana) s 2 + 2 sata predavanja.
  - Temeljne cjeline:
  I. Uvod i motivacija, Procesi programskog inženjerstva
  II. Arhitekture programske potpore (modeli i izbor, UML ..)
  III. Validacija, verifikacija i ispitivanje programske potpore
- ***Samostalni rad***: Projekt (3 kontrolne točke)
- ***Anketa***: početni, središnji i završni upitnik
- ***Kontinuirana provjera znanja:***
  - Kratke provjere znanja, Međuispit, Završni ispit

Praktični rad tijekom semestra

- Projekt:
  - Timski rad u grupi od 10 studenata tijekom cijelog semestra.
  - Nastavnici određuju početnog koordinatora svakog tima i pridijeljenu temu-projekt.
  - Koordinator formira tim iz skupine svih upisanih studenata.
    - Ako do određenog roka nije formiran tim, to će učiniti nastavnici.
  - Tim samostalno određuje voditelja.
    - Ako tim do određenog roka ne odredi voditelja, koordinator postaje voditelj.
  - Voditelj preuzima organizaciju rada na projektu i odgovoran je za poštivanje rokova.
  - Projekt se izvodi i dokumentira tijekom cijeloga semestra.
  - Za svaki tim biti će zadužen jedan asistent za konzultacije i provjeru rada.
    - Provjera usvojenih znanja potrebnih za izvedbu projekta, postupke ostvarivanja te cjelovitost i razumljivost dokumentacije.
    - Druga provjera uključuje predaju implementiranog projekta.

Preporučena literatura

- ***Bilješke s predavanja***
- Nastavni sadržaji prezentacija s predavanja
  - 2-3 dana prije predavanja (FER web)
- Dodatni sadržaji
  - Članci, tehnička dokumentacija (FER web, web)
- Vlastite bilješke predavanja i diskusija
- A. Jović, M. Horvat, I. Grudenić: UML dijagrami – Zbirka zadataka i riješenih primjera.

Knjige:

- Lethbridge, T. C., Laganiere, R., ***Object-Oriented Software Engineering***, 2nd ed., McGraw-Hill, 2005.
- Sommerville, I., ***Software engineering***, 8th ed, Addison Wesley, 2007.
- .....
- Guide to the Software Engineering Body of Knowledge (SWEBOK)

- Maršić,I., ***Software engineering,*** Department of Electrical and Computer Engineering, Rutgers University, http://www.ece.rutgers.edu/~marsic/books/

- Schach, S. R.: Classical and Object Oriented Software Engineering, 7th edition, McGraw-Hill, 2007
- Pressman R.S.: Software Engineering – A Practitioner's Approach, McGraw-Hill, 2009
- Gamma E. et al: Design patterns: elements of reusable object-oriented software, Addison-Wesley, 1995
- Rambaugh J., Jacobson I. and Booch G.: *The Unified Modeling Language Reference Manual,* Addison-Wesley, 2005
- Fowler M.: *UML Distilled*, Addison-Wesley, 3rd edition, 2004
- ... link

Ocjenjivanje u kontinuiranoj nastavi

Aktivnost	max. 9
Projekt	max. 30
Međuispit	max. 25
Završni ispit	max. 36

- Uvjeti za izlazak na završni ispit:
  - ≥ 1/9 bodova iz aktivnosti.
  - ≥ 15/30 bodova iz projekta.
- *Neispunjavanje uvjeta ⇒ predmet se upisuje slijedeće godine*.

Formiranje ocjena

- Polaganje predmeta
  - na završnom ispitu postignuto ≥ 12 bodova
  - prag za prolaz: 50 bodova
- Formiranje ocjena
  - prema utvrđenim pragovima

Ocjena	Bodovi
Izvrstan (5)	≥86
Vrlo dobar (4)	≥72
Dobar (3)	≥60
Dovoljan (2)	≥50

Ocjenjivanje na ispitnim rokovima

- Uvjeti za izlazak na ispitni rok:
  - ≥ 1/9 bodova iz aktivnosti.
  - ≥ 15/30 bodova iz projekta.

Na ispitnom roku prenose se bodovi iz kontinuirane nastave:

	Maks. broj bodova:
Aktivnost:	9
Projekt (8+8+14):	30
Pismeni ispit na roku:	61

- Na pismenom ispitu potrebno je ostvariti barem 30 bodova.
- Polaganje predmeta
  - Na ispitu postignuto ***≥ 30 bodova***
  - prag za prolaz: ***50 bodova***
- Formiranje ocjena
  - **prema utvrđenim pragovima**

Studijski program Računarstvo

Akademski naziv: baccalaureus računarstva

Osnovni objekt proučavanja u računarstvu jest računalo kao univerzalni stroj za obradu informacija, kao i metode njegove primjene u drugim djelatnostima. Proučavanju računala pristupa se kroz cjelovito sagledavanje njegovih sklopovskih i programskih aspekata kao i njihovih međuzavisnosti. Računarstvo obuhvaća ***teoriju, metode analize i sinteze, projektiranje i konstrukciju, primjenu i djelovanje računalskih sustava***. Ovaj preddiplomski studijski program omogućava stjecanje kompetencija za analizu i rješavanje ***srednje složenih inženjerskih problema, za rad u timu, te za doprinos oblikovanju sustava i procesa s područja računarstva***, uz korištenje temeljnih znanja iz matematike, fizike, elektrotehnike i računarstva te ***suvremenih računarskih alata.***

O predmetu

- Ovaj predmet daje osnovna znanja i vještine nužne za postizanje kompetencija u programskom inženjerstvu, a posebice razumijevanje, evaluaciju i oblikovanje programskih sustava.
  - Generički modeli procesa programskog inženjerstva.
  - Inženjerstvo analize zahtjeva.
  - Koncepti programskih arhitektura i paradigme specifikacije.
  - Modeliranje ulazno/izlaznih i reaktivnih programskih sustava.
  - Modeliranje objektno usmjerenih sustava (UML).
  - Formalna specifikacija i verifikacija željenih obilježja programskih sustava.
  - Ispitivanje programskih sustava.
  - Automatizirani pomoćni postupci i alati u oblikovanju programskih sustava.

Tema

- Programi – glavna “industrija” današnjice
  - Primjena u poslovnom, znanstveno-istraživačkom, društvenom, ... životu
  - IT sustavi, prikupljanje, obradu, pohranjivanje, dohvaćanje, manipulaciju informacijama
  - Sve razvijene ekonomije ovisne o programskoj potpori
- Aplikacijski programi
- Sistemski programi
- Obožavanje informacijske tehnologije i programa
  - Promjena današnjeg društva
  - Standardizacija?

Razvoj programa

- 1965 Gordon Moore
  - Broj tranzistora po in^2 udvostručuje se svake godine
  - Danas: udvostručenje svakih 18 mjeseci
- Svojstva programa
  - ***Veličina***: broj linija koda - KLOC
  - ***Funkcionalnost***: funkcionalnost koja je na raspolaganju krajnjem korisniku
  - ***Složenost***: složenost problema, algoritamska složenost, strukturna složenost, spoznajna složenost
- Primjer:
  - 106 LOC, 80 upravljačkih jedinki, 5 sabirnica
- 2010 – tipični ugrađeni sustav 100 10^6 LOC

Proces izrade PP

- Pretpostavke:
  - Dobar proces -> dobra PP
  - Dobar proces -> manji rizik neuspjeha
- Upravljanje rizikom (*engl. Risk Management*)
  - Koji su problemi izrade PP?
  - Kako smanjiti rizik?

Problemi razvoja PP

- “The major cause of the ***software crisis*** is that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.”
  - – 1972 Edsger Dijkstra, The Humble Programmer
- Programi
  - Primjena u kritičnim područjima
    - ?
  - Kratko vrijeme izrade
  - Kvaliteta
- 28% uspješnih projekata
  - !?

Razvoj programske podrške

- Jedan od načina:

  “The amateur software engineer is always in ***search of magic***, some ***sensational method*** or tool whose application promises to render software development ***trivial***. It is the mark of the ***professional software engineer*** to know that no such panacea exists.”

  - Grady Booch, Object-Oriented Analysis and Design

Primjer povijesnih problema PP

- Signifikantan događaj (1985-1987)
- Therac-25: sustav za terapiju radioaktivnim zračenjem
  - elektroni ili fotoni do 25 MeV
  - 6 fatalnih pogrešaka (prekomjerne doze)
  - Najgori akcident u 40 godina terapije zračenjem
  - Nažalost nije i zadnji!!!
- Program:
  - DEC PDP 11 asembler
  - konkurentni procesi
  - “test-and-set” nije bila nedjeljiva operacija
  - višestruki upis u zajedničku varijablu
- Detaljnije:
  - http://computingcases.org/case_materials/therac/case_history/Case%20History.html

Software “HALL OF SHAME”

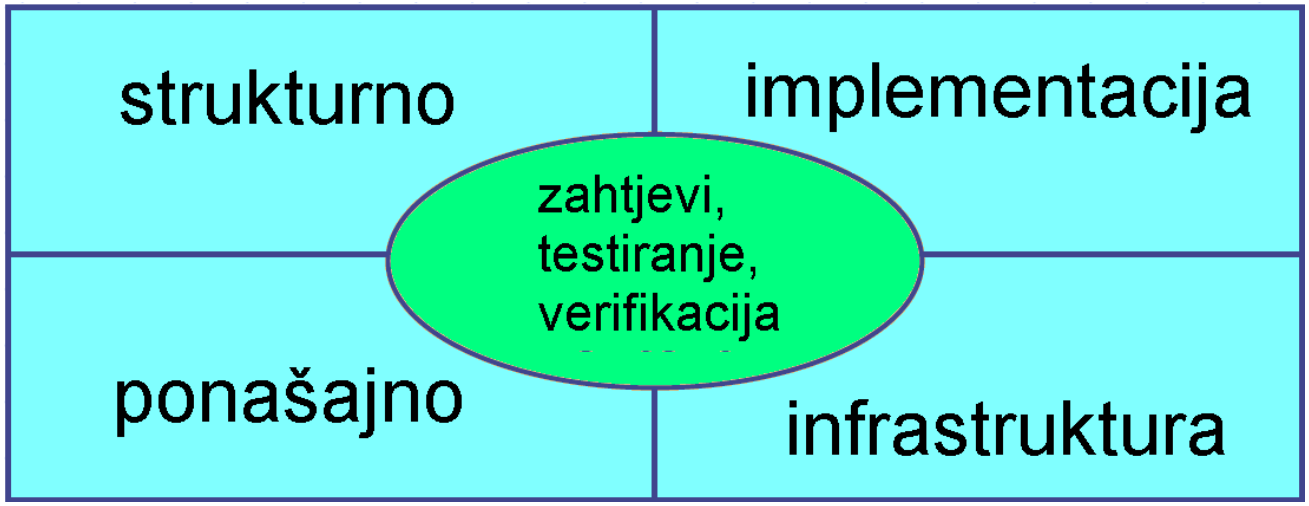
- 1993: London Stock Exchange ($ 600 M, canceled)
- 1995: American Airlines, (159 death)
- 1996: Arianespace (Ariane 5 rocket explosion, $350 M)
- 1997: ***Korean Jet crash, (225 death)***
- 1999: State of Mississippi (Tax system, $185 M loss)
- 2000: National Cancer Institute (pogrešan izračun doze-8 mrtvih/20 ozlijeđenih)
- 2001: ***Panama radiation overdose, (5 death)***
- 2001: Nike Inc. (Supply chain management, $100 M loss)
- 2002: McDonald’s (Purchasing system canceled, $170 M)
- 2003: ***North-eastern U.S. Power loss (3 death)***
- 2004: Hewlett-Packard (Management system, $160 M loss)
- 2004: Sainsbury food chain, UK ($527 M, canceled)
- 2004: Ford Motor Co. (purchasing, $400 M, canceled)
- 2004: Mars Spirit, NASA: "a serious software anomaly“
- 2005: UK Tax (soft errors resulted in $3.5 G (billion) overpay)
- 2005: FBI Trilogy ($105 M, canceled)
- . . .
- 2009: SQL injection errors - steal data on approximately 130 million credit and debit cards over several months
- 2010: German Bank Cards (30 M korisnika,…)
- 2010: Alaram Clock Bug in Mobile phones

Razrješavanje problema

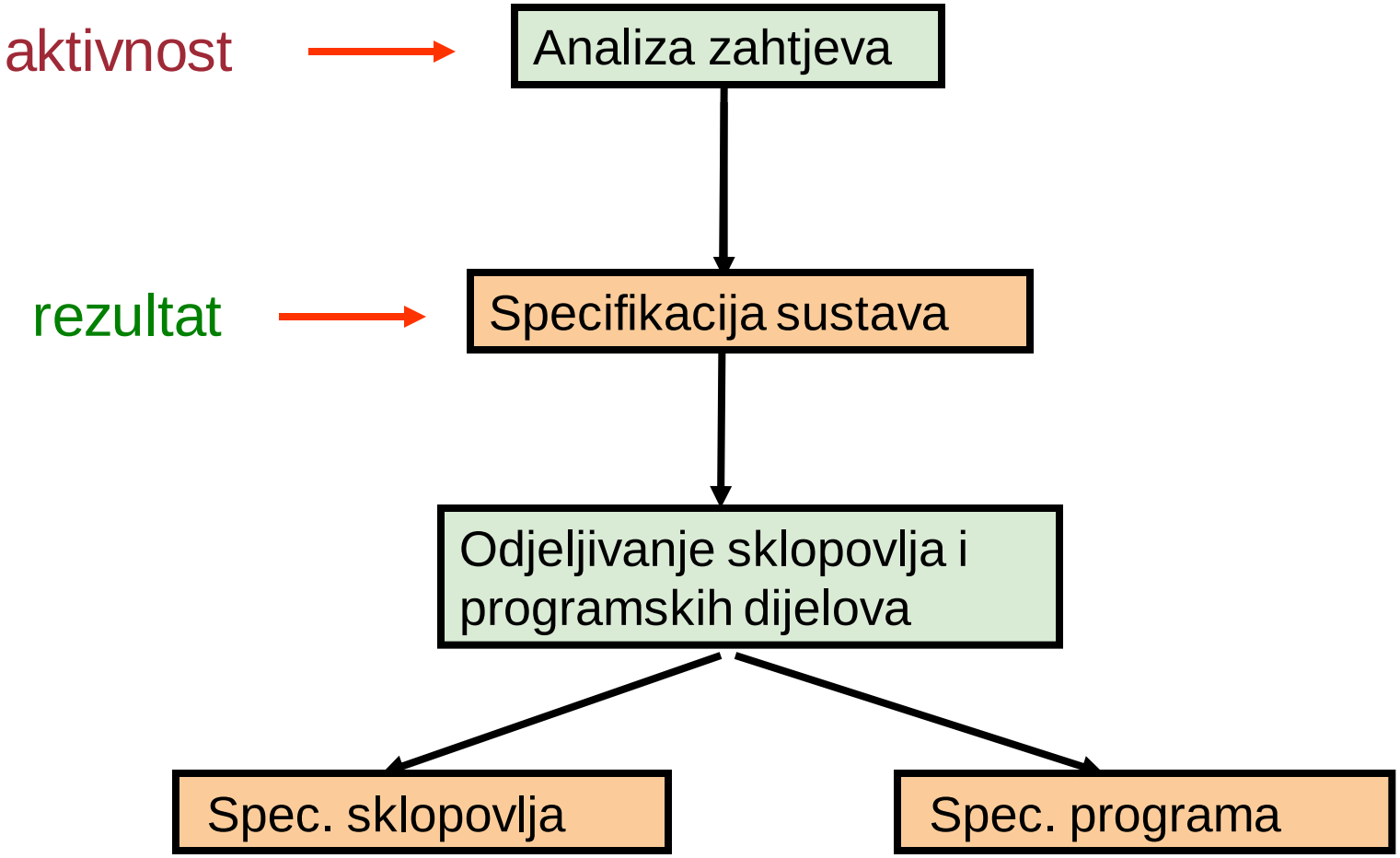
- 1968 - software engineering
  - Pokušaj odgovora na stanje razvoja kvalitetnih programa na vrijeme i zadanih sredstava
- 1969 NATO Glasgow

Oblikovanje programske podrške

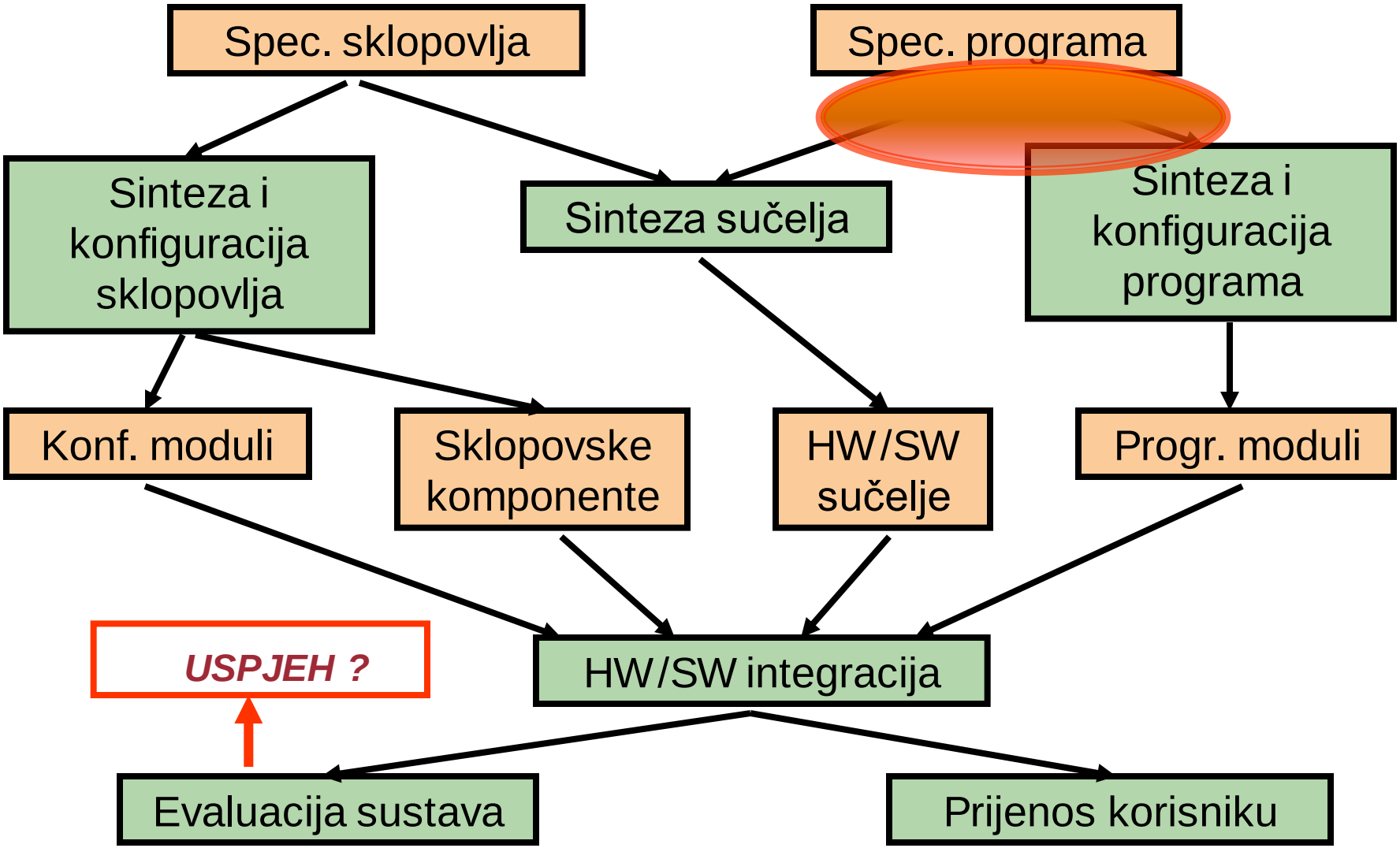
- *engl.* ***Software design***
- Stvaranje nacrta
- Precizan opis sustava s različitih pogleda



Tradicijski postupak oblikovanja sustava (1)



Tradicijski postupak oblikovanja sustava (2)



Nedostaci postupka oblikovanja sustava

- Analiza i oblikovanje:
  - Zahtjevi nisu formalizirani
    - “Formalni” = temeljeni na matematičkim teorijama (logika, teorija automata, teorija grafova, …).
    - nemoguće postići preciznost
    - neodređenost dovodi do nekompatibilnosti
  - Izgradnja “ad hoc”prototipa
    - neformalni prototipovi ne omogućuju dublju analizu
  - Procjena performansi
    - neformalni pristup daje pogrešne vrijednosti
  - Dijeljenje na sklopovski i programski dio
    - nije poduprto formalnim transformacijama i postupcima donošenja odluka
  - Dokumentacija se teško interpretira

Problemi ispitivanja sustava

- Ručna ispitivanja (testiranje, provjera):
  - Nedovoljno duboka
    - Složeno predvidjeti sve moguće interakcije
  - Cijena ispitivanja čini najveći dio cijene programske potpore.
  - Pokrivenost skupa testiranih stanja je ograničena.
  - Granični slučajevi (*engl. Corner cases*)
    - teško identificirati
    - nemodularno oblikovanje
- Vrijeme stavljanja proizvoda na tržište (*engl. time to market TTM*) smanjuje pokrivenost ispitanih stanja i podupire mentalitet "kolektivne krivnje".
- Provjere ispravnosti uporabom simulacija i ispitivanja:
  - ***mogu samo dokazati postojanje pogreške (****engl. bug****)***
  - ***ali nikada njeno nepostojanje*** (Dijkstra)!!!

Tradicijsko oblikovanje programske potpore

- Formulacija zahtjeva (engl. requirements)
  - govori što bi sustav trebao raditi.
- Specifikacija i analiza.
- Oblikovanje
  - kako će sustav ispuniti ciljeve
- Kodiranje
  - stvarno programiranje
- Ispitivanje modula.
- Integracija i ispitivanje sustava.

Principi savladavanja problema složenosti

- razdor problem - implementacija
- ***Dekompozicija*** - “podijeli pa vladaj”
- ***Inkrementalno poboljšanje***
- ***Ponovno korištenje dijelova*** (*engl. design reuse*)
- ***Odvojiti podprobleme***:
  - logički - fizikalni
  - logički - vremenski slijed
  - funkcije - komunikacije
- ***Apstrakcija*** - eliminirati nepotrebne detalje
  - uvođenje modela.
- ***Formalizacija*** - matematički određena semantika!!!

Metodologije oblikovanja zasnovane na modelima

- *engl. Model-driven Development Methods*

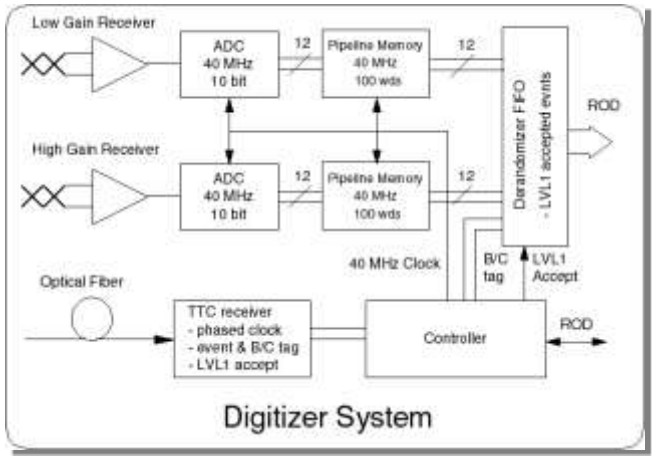
- Slijedećih nekoliko slika preuzeto od:
- Bran Selic

- "IBM Distinguished Engineer" u IBM Rational.
  - Honorarni profesor, Carleton University in Ottawa, Canada.
  - Preko 30 godina iskustva u oblikovanju i implementaciji velikih industrijskih programskih sustava.
  - Predvodnik metodologije oblikovanja zasnovanog na modelima.
  - Predsjeda OMG (Object Management Group) timu odgovornog za standard "unificirani jezik za modeliranje" UML 2.0 (*engl, Unified Modeling Language*).

Model

- Inženjerski model:
- Sažeta reprezentacija sustava koja naglašava značajna svojstva iz nekog pogleda na sustav

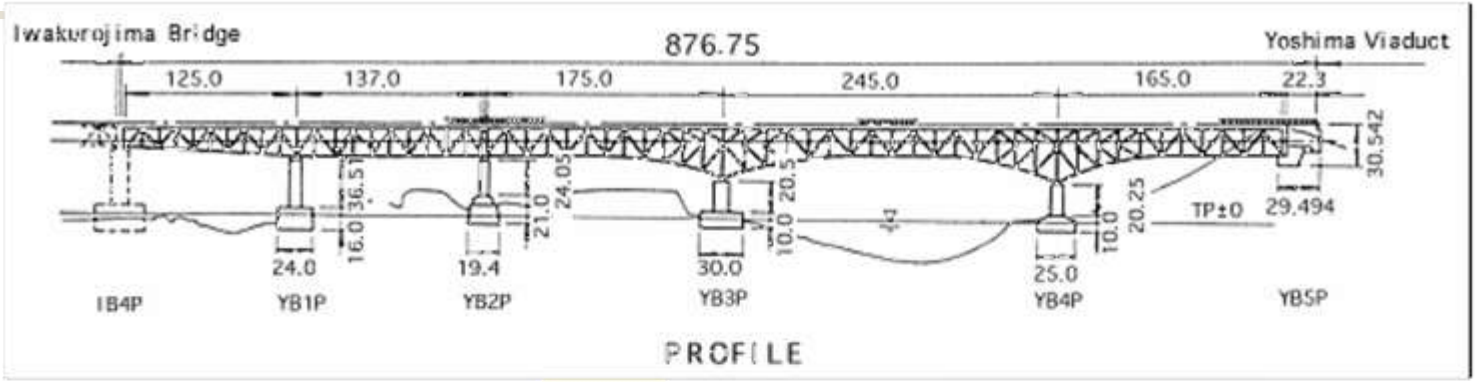
modelirani sustav



Funkcionalni model

- Ne prikazuje sve odjednom
- Upotrebljava oblik prezentacije (notacije) jednostavno razumljiv za traženu namjenu

Primjer modela



Problem?

Differences **due to:**

- Idiosyncrasies of actual construction materials
- Construction methods
- Scaling-up effects
- Skill sets/technologies
- Misunderstandings

Can lead to serious errors and discrepancies in the realization

Uporaba modela

- Razumijevanje složenih sustava
  - Specifikacija zahtjeva
  - Projektiranje
  - Rano otkrivanje pogrešaka
    - analiza, simulacija
  - Olakšano komuniciranje
- Osnova za implementaciju

- Svojstva modela
  - Apstrakcija
  - Razumljivost
  - Točnost
  - Predvidljivost
    - daje odgovor na pitanja o promatranom sustavu
  - Jednostavnost

Utjecaj promjena okoline na razvoj PP

- Promjene koje utječu na razvoj programske potpore:
  - Sklopovlje: brzina, pouzdanost, cijena, …
  - Načini i svojstva povezivanja računala
  - Timski rad
  - Razvoj sučelja čovjek - računalo
  - Napredak razvojnih alata
  - Složenost aplikacija
- Ne mijenja se:
  - nivo apstrakcije programiranja

Problem pisanja ispravnih programa

- SLOŽENOST (*engl. COMPLEXITY*)

- Nivo složenosti modernih programa dosegao je razinu gdje se može mjeriti s biološkim sustavima!!
  - sustavi sustava s vise desetaka milijuna linija koda

- Klasifikacija složenosti:
  - Osnovna složenost
    - ovisi o problemu
    - ne može se eliminirati tehnologijom ili tehnikama
  - Nenamjerna (*engl. Accidental complexity*)
    - uvedena uporabom alata ili tehnika
- Problem razvoja programske potpore je prekomjerne nenamjerne složenosti

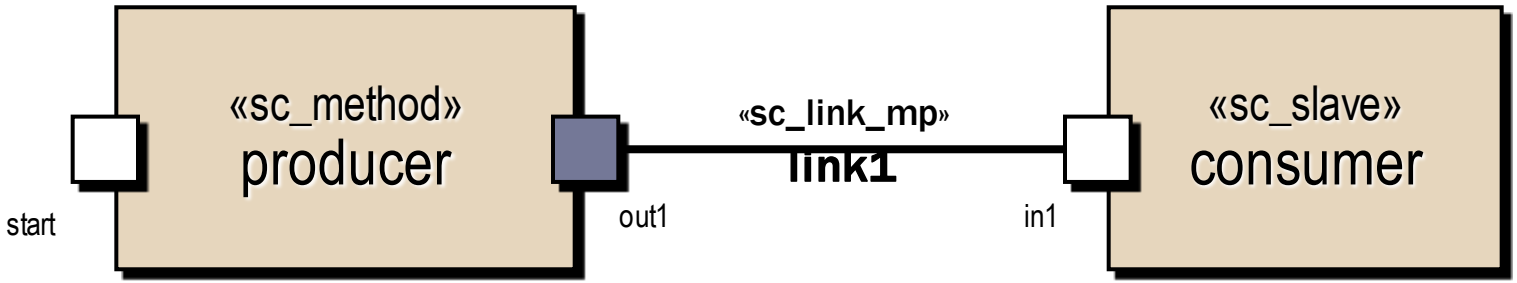
Primjer programa

```
SC_MODULE(producer)
{
sc_outmaster<int> out1;
sc_in<bool> start; // kick-start
void generate_data ()
{
for(int i =0; i <10; i++) {
out1 =i ; //to invoke slave;}
}
SC_CTOR(producer)
{
SC_METHOD(generate_data);
sensitive << start;}};
SC_MODULE(consumer)
{
sc_inslave<int> in1;
int sum; // state variable
void accumulate (){
sum += in1;
cout << "Sum = " << sum << endl;}
SC_CTOR(consumer)
{
SC_SLAVE(accumulate, in1);
sum = 0; // initialize
};
SC_MODULE(top) // container
{
producer *A1;
consumer *B1;
sc_link_mp<int> link1;
SC_CTOR(top)
{
A1 = new producer("A1");
A1.out1(link1);
B1 = new consumer("B1");
B1.in1(link1);}};
```

Vidljivost strukture?

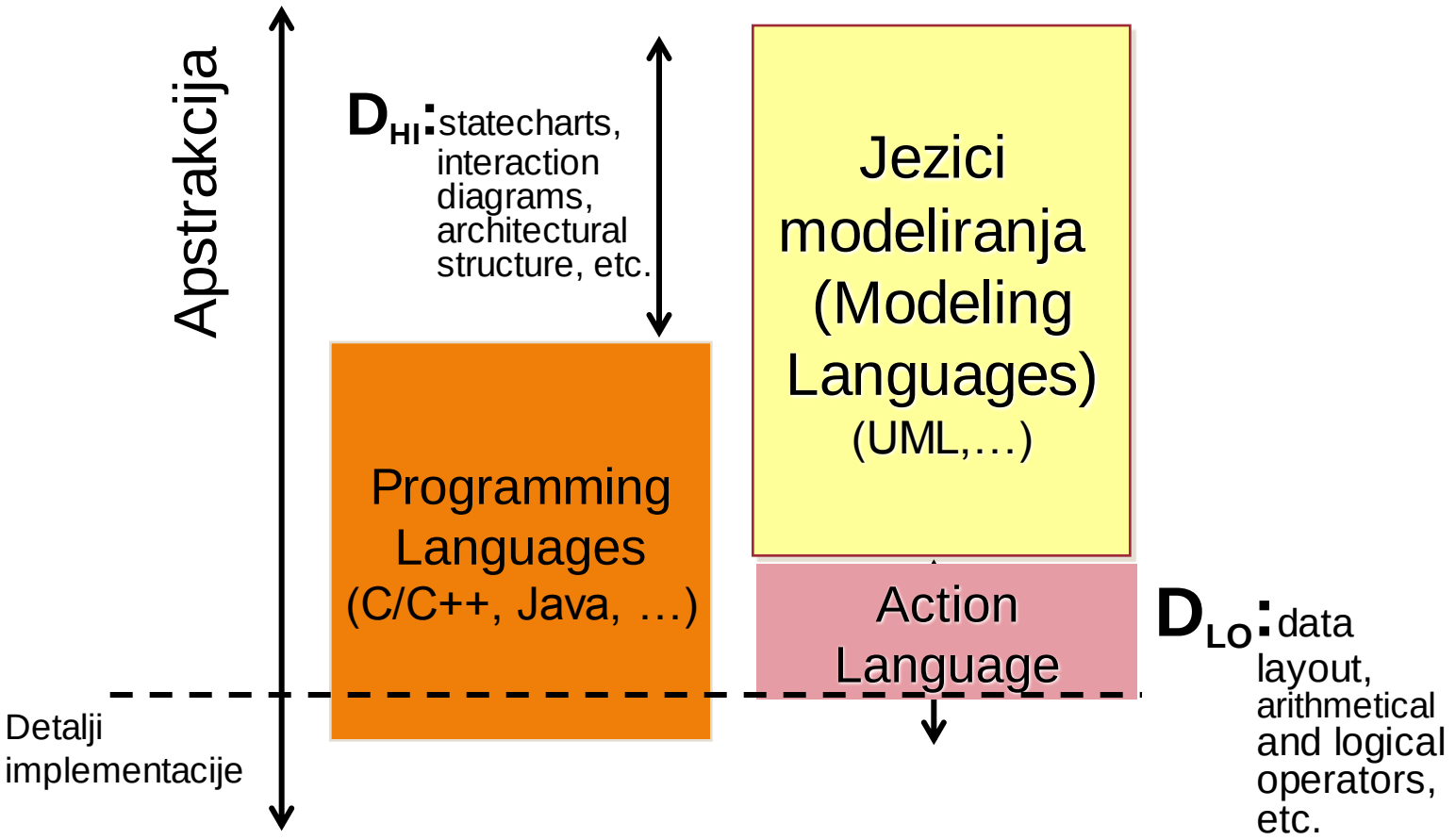
Model programa

- Model:



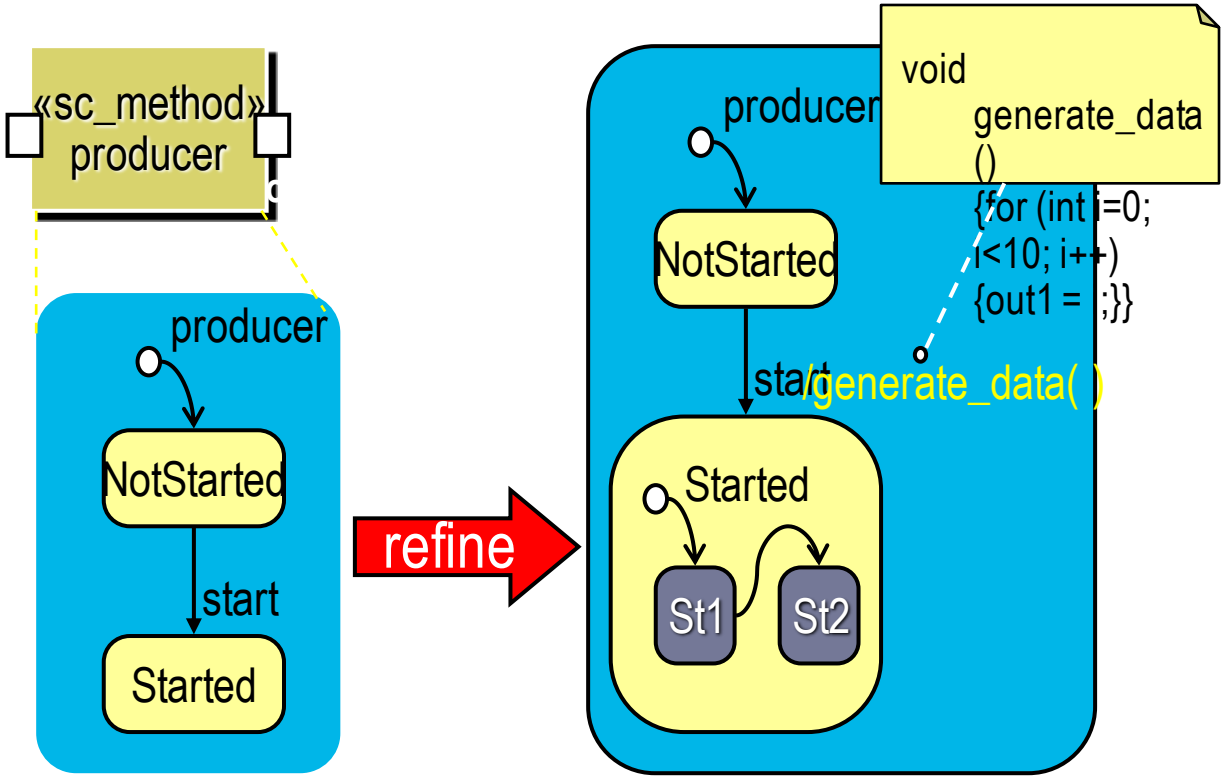
- Koliko je koristan?

Programiranje i modeliranje



Razvoj modela

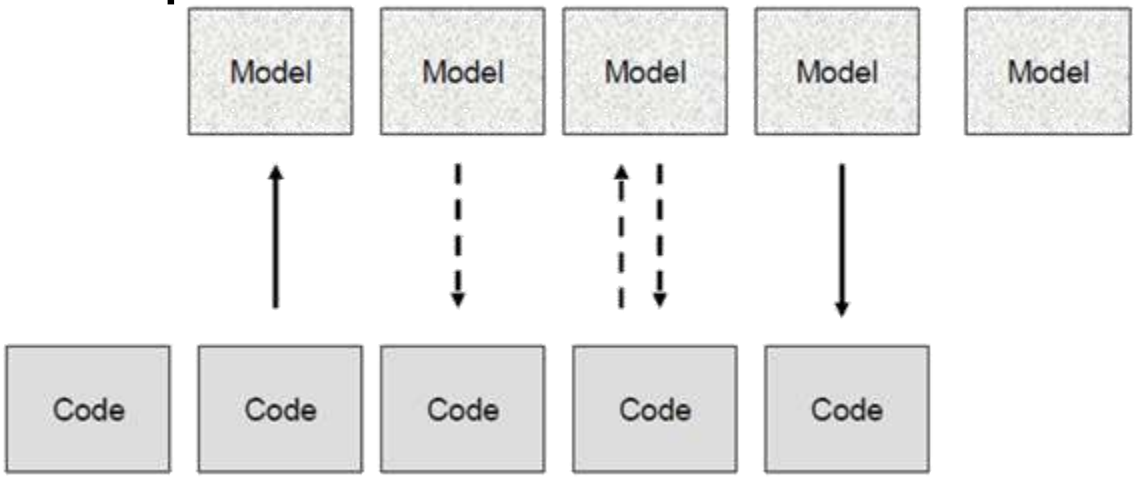
- U oblikovanju programske potpore modeli se mogu nadograđivati sve do potpune specifikacije
  - model postaje sustav kojeg je u početku samo modelirao!!



Osobito svojstvo programa

- Programi imaju jedinstveno svojstvo koje omogućava razvoj apstraktnih modela u potpunu implementaciju bez promjene inženjerske okoline, metoda ili alata.
- Omogućavaju generiranje apstraktnih modela izravno i automatizirano iz implementacije
  - to osigurava potpunu točnost modela programa
  - model ≡ sustav
- Možemo li to iskoristiti?

Oblikovanje PP zasnovano na modelima

- *engl. Model-Driven Software Development*
- Postavljaju model u centar razvojnog procesa PP
  - metode: apstrakcije i automatizacija
- Povijest uporabe modela:
- 

- Cilj: povećanje razine apstrakcije, poboljšanje komunikacije, produktivnost, održavanje ....

Neobično svojstvo programa

što je najznačajnije?

Posljedice

- Programska potpora je manje ovisna o okolini
  - ne potpuno
- Individualne mogućnost dolaze do izražaja
  - produktivnost pojedinca različita
  - kvaliteta ?
  - inteligencija?
- Brz razvojni ciklus

Poboljšanja

Kako oblikovati programsku potporu da se smanji vjerojatnost neuspjeha?

1. Uvesti inženjerski propisane ***postupke*** (procedure) u proces oblikovanja programske potpore.
   precizno definirati faze procesa oblikovanja – tko radi što i kada.

2. Svaku fazu procesa ***dokumentirati*** (po mogućnosti na standardan i formalan način).

3. U oblikovanje uvesti ***analizu i izbor stila arhitekture*** programske potpore te pripadne modele.

4. Programsku potporu oblikovati u manjim cjelinama (***komponentama***).

5. Uz tradicijsko ispitivanje (testiranje) uvesti ***formalne metode provjere*** (verifikacije) modela programske potpore.

Proces oblikovanja programske potpore

- Ovaj dio problema oblikovanja programske potpore analizira i predlaže rješenja disciplina koju nazivamo ***Programsko inženjerstvo*** (*engl. Software engineering*).
- Nema jedinstvenog i standardnog prijedloga
  - Mogu se izlučiti neke generičke aktivnosti u svim prijedlozima.
  - Osnove najpopularnijih pristupa razmotrit će se u nastavku predavanja.
- Posebice je bitno razmotriti postupke precizne specifikacije zahtjeva koje korisnik postavlja na sustav.
  - ***Inženjerstvo zahtjeva*** (*engl. Requirements engineering*).

Uloga dokumentiranja

- U procesu oblikovanja programske potpore mogu se uočiti pojedine faze.
- Rezultat svake ***faze oblikovanja*** je ***dokument***.
  - dokumenti koji specificiraju zahtjeve korisnika,
  - dokumenti koji specificiraju arhitekturu sustava (uz navođenje argumenata zašto je izabrana),
  - dokument kodiranja, dokument ispitivanja i sl.
- Rezultat oblikovanja projekta na ovom kolegiju je također dokument koji ima tri dijela (tri faze).
- Dokumenti se uobičajeno podvrgavaju ocjeni i reviziji (promjeni).

Arhitekture programske potpore

- Temeljem izlučivanja zahtjeva ***analizira*** se nekoliko stilova arhitekture programske potpore kao bi se došlo do optimalnog izbora obzirom na konkretnu primjenu.
- Stilovi arhitekture ne mogu se analizirati na razini programskog koda, već na apstrakcijama te arhitekture – modelima.
- Svi stilovi arhitekture nisu jednako detaljno podržani modelima.
  - Neki stilovi još nemaju standardne modele.
- Najdetaljnije je podržan stil arhitekture koji nazivamo Objektno usmjeren pristup; OO (*engl. Object oriented*)
  - u fokusu ovoga kolegija.

Uloga komponenata

- Oblikovanje programske potpore slijedi princip “podijeli pa vladaj”.
- U fokusu oblikovanja su manji dijelovi sustava – komponente, te njihovo ponovno i višestruko korištenje (*engl. reuse*).
- Princip višestrukog korištenja u oblikovanju programske potpore manifestirao se tijekom vremena kroz:
  - programske jezike, knjižnice procedura i objekata, ponovno korištenje manjih dijelova arhitekture (arhitekturni obrasci – *engl. architectural patterns*) ili većih dijelova (radni okviri – *engl. frameworks*).

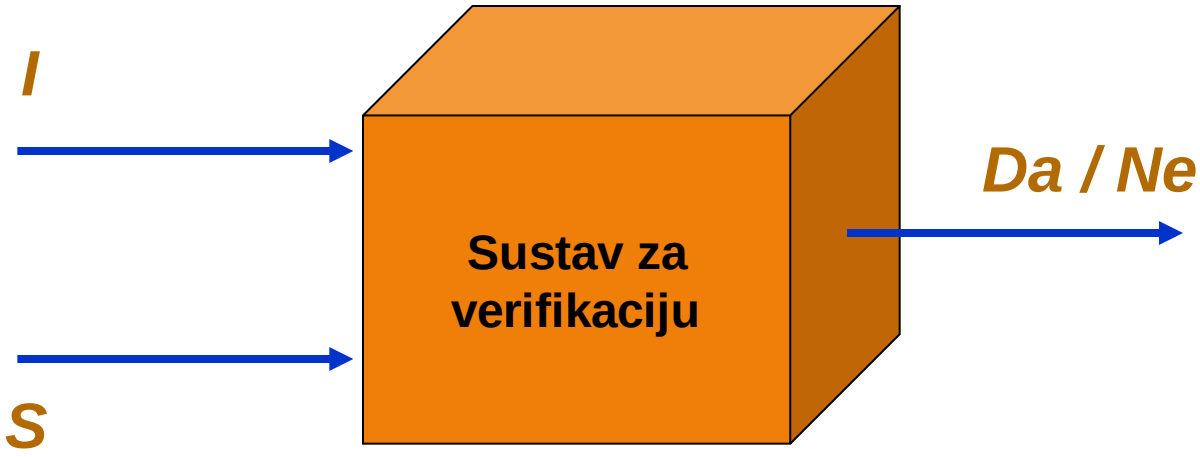
Formalne metode provjere

- Temeljem procesa oblikovanja programske potpore generira se *formalna specifikacija i formalni model*
  - S – specifikacija, što sustav treba raditi
    - ***formalna specifikacija***
  - I – implementacija, kako sustav radi
    - ***formalni model***
- Cilj formalnih metoda provjere:

  Odredi da li: *I* ***odgovara*** *S*.
- Značenje "Odgovara":
  - ekvivalencija, simulacijske relacije, logička zadovoljivost, logička implikacija, implementacijske relacije, …

Formalna verifikacija

- Postupak provjere da ***formalni model*** izvedenog sustava ***(I),*** odgovara ***formalnoj specifikaciji (S)*** s matematičkom izvjesnošću



- Primjer ulazno izlaznog programa:
  - Specifikacija (S) – željena transformacija ulaza (npr. y = x2).
  - Model implementacije (I) – matematički opis izvedbe
    npr. y = 1 + 3 + … (2x – 1)

Primjer reaktivnih programa

- trajna interakcija s okolinom (operacijski sustavi, upravljanje avio letovima, upravljanje nuklearnim reaktorom, …)
  - *S*, *I*: formalizacija njihovog ponašanja tijekom vremena!
- Primjer:
  - Opis ponašanja: “Nikada se ne smije dogoditi da su vrata otvorena dok se lift kreće”,
  - (otvorena_vrata $\wedge$ lift_se_kreće) mora uvijek biti neistinito.
  - Primjer specifikacije *S* Emerson - Clarke notacijom**:**

    AG $\neg$(otvorena_vrata $\wedge$ lift_se_krece)
  - ***Modeli implementacije:***

    Regularni jezici

    Strojevi s konačnim brojem stanja

Značaj FV

- Često se oglašava da FV daje apsolutnu sigurnost u ispravnost sustava (programa/sklopovlja)

iako

- ***FV garantira izvan svake sumnje samo ispravnu relaciju između formalnih (matematičkih) objekata I i S***

- ***FV ne osigurava ispravnu relaciju između stvarne implementacije i specifikacije***

- Temeljno:
  - FV radi s MODELIMA (apstrakcijama) stvarne implementacije

Razlozi neispravnosti postupka i rezultata FV

1. Razlika između matematičkih objekata i stvarnosti
   - model *I* nije obuhvatio bitna obilježja stvarne implementacije (npr. ograničena duljina riječi, precizna informacija o vremenskim odnosima i sl.).
   - Model *S* pretpostavlja suviše jaka ograničenja okoliša.
2. Neadekvatna specifikacija željenih obilježja
   - *S* ne opisuje intencije korisnika.
   - *S* je nekompletan ili neispravan (opisuje krivo ponašanje).

Okosnice za FV

- Algoritamska verifikacija
  - provjera modela, *engl. model checking*
  - Istraživačka tehnika za strojeve s *konačnim* brojem stanja.
  - Pobrojavanje (numeriranje i simboličke varijante)
- Postupci zasnovana na dokazivanju teorema
  - *engl. Automatic Theorem Prover - ATP*
  - Primjenjivi na sustave s *beskonačnim* brojem stanja.
  - Zahtijevaju interakciju korisnika (ekspertno znanje)

FV u praksi

- Poluvodička industrija (ASICS i VHICS):
  - formiranje istraživačkih i razvojnih odjela, uvođenje FV
- Komunikacijski protokoli
  - često su formalno verificirani
- Sigurnosno-kritični sustavi
  - zahtijevaju potvrdu FV
- Programsko inženjerstvo
  - slabo prihvaćanje tehnika FV

Zašto postupci FV nisu široko prihvaćeni

- Ne uči se na sveučilištima. ☺
- Usredotočenost na ulazno/izlazne scenarije.
- Tradicija (nedovoljno matematike u računarstvu).
- “Formalisti” su također krivi jer:
  - Mnogi nemaju iskustva na stvarnim sustavima.
  - Reklamiraju i ono što se sa FV ne može postići.
  - Često se ekstrapolira (“silver bullets” traže vrijeme).

- Formalna verifikacija (FV) dio je područja kojega nazivamo Formalne metode (FM).

- Pitanje:
  - Da li se kakvoća i pouzdanost programske potpore povećava uvođenjem formalnih metoda u proces oblikovanja?

Spas u formalnim metodama?

- US Computer Science and Technology Board
  - FM trebaju biti ***ključna inženjerska vještina*** u računarstvu koja povećava ***kakvoću programske potpore i istovremeno*** smanjuje vrijeme stavljanja produkta na tržište (*engl. time to market*), te se mora uključiti u obrazovanje CS i EE.
- Treba razlikovati ***matematičke osnovice***
  - teorija domena, teoriju čvrste točke, propozicijska i predikatna logika, teorija automata, teorija grafova i sl.
- od ***formalnih postupaka***
  - Z metoda, provjera modela, funkcijsko programiranje i sl.

Primjer:

- 2007 Turing Award Winners Announced for their groundbreaking work on ***Model Checking***

- Edmund M. Clarke, E. Allen Emerson, and Joseph Sifakis are the recipients of the 2007 A.M. Turing Award for their work on an ***automated method for finding design errors in computer hardware and software***. The method, called Model Checking, is the most widely used technique for detecting and diagnosing errors in complex hardware and software design. It has helped to improve the reliability of complex computer chips, systems and networks.

Trend i problemi

- CrossTalk, The Journal of Defense Software Engineering, January, 2008
- Dr. Robert B.K. Dewar, AdaCore Inc.
  Dr. Edmond Schonberg, AdaCore Inc.
  Computer Science Education: Where Are the Software Engineers of Tomorrow?

  It is our view that Computer Science (CS) education is neglecting basic skills, in particular in the areas of programming and formal methods. We consider that the general adoption of Java as a first programming language is in part responsible for this decline. We examine briefly the set of programming skills that should be part of every software professional's repertoire.
- Computer science professors at New York University, say that today's computer science graduates are not equipped with the skills they need to work well in the current software industry, leading to the conclusion that "***we are training easily replaceable professionals***."
- They recommend an approach to CS education characterized by the earlier introduction of formal methods such as model checking.

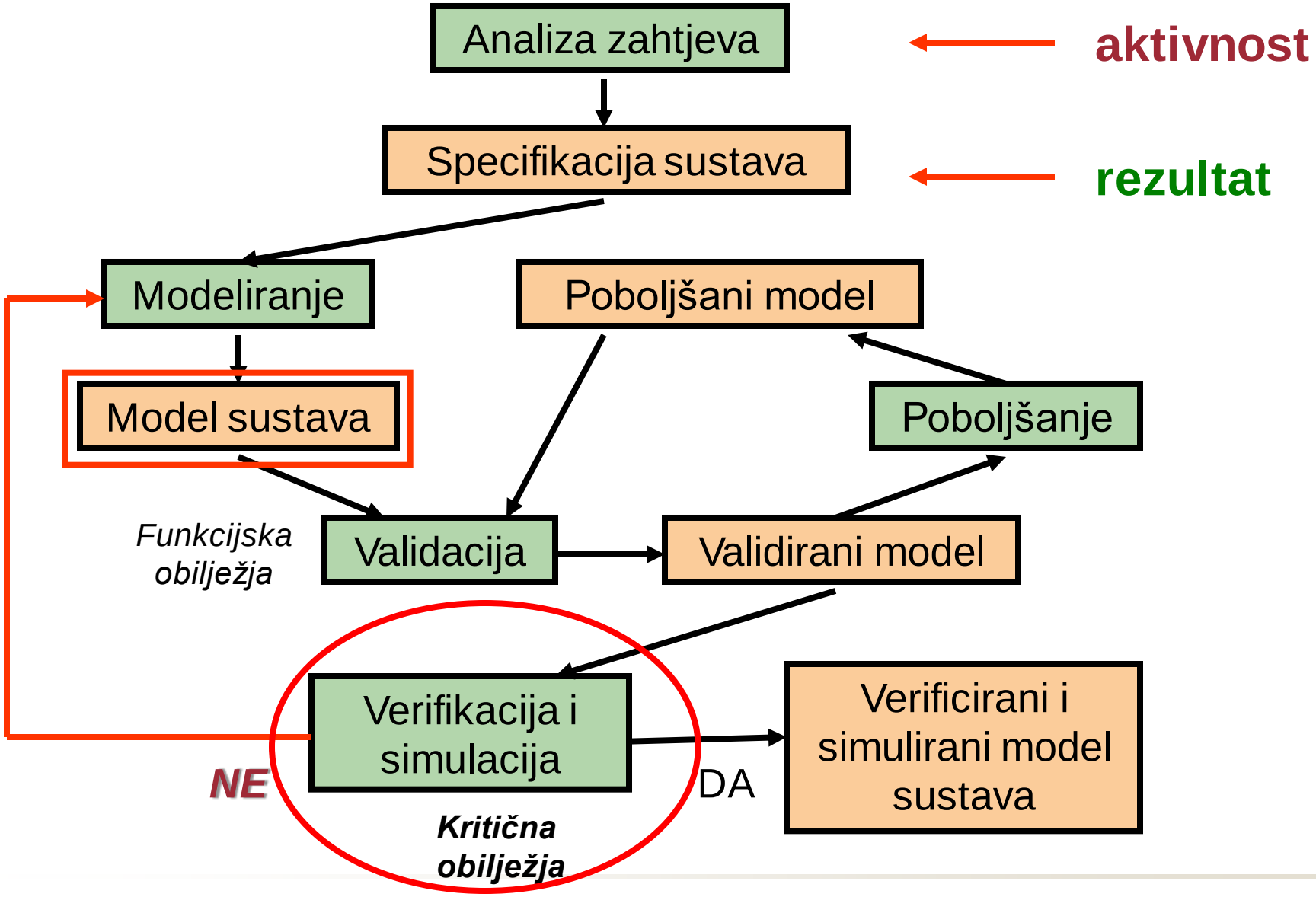
Trend i problemi

- Diskusija...
- Bachelor of Fine Arts in Software Development
- ”Imagine instead an undergraduate curriculum that consists of 1/3 liberal arts, and 2/3 software development work. The teachers are experienced software developers from industry. The studio operates like a software company. You might be able to major in Game Development and work on a significant game title, for example, and that's how you spend most of your time, just like a film student spends a lot of time actually making films and the dance students spend most of their time dancing.“
  - Joel Splosky

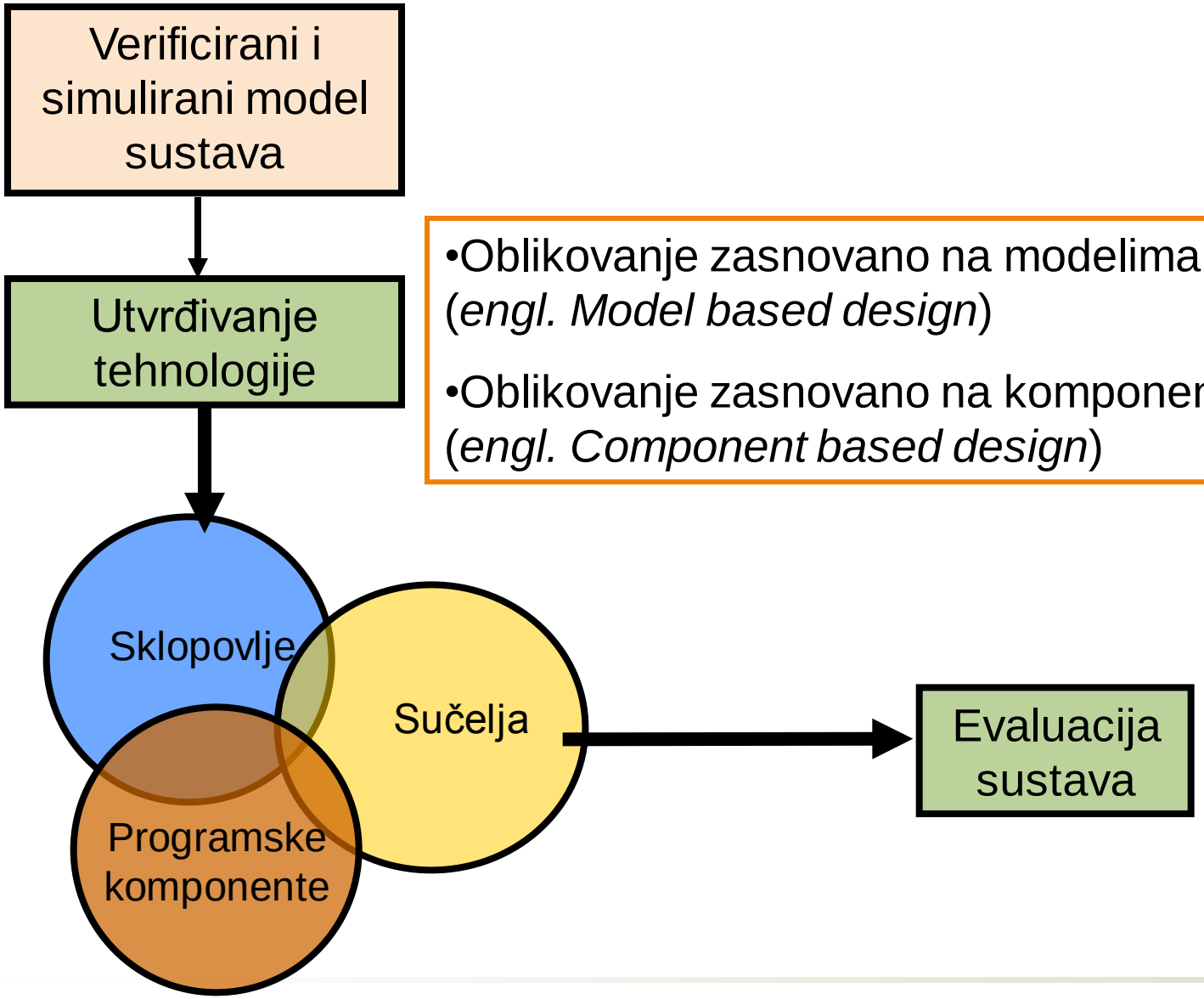
Analiza postojećeg stanja

- Temeljem analize nedostataka tradicijskog postupka oblikovanja sustava i izvedenih zaključaka slijedi predloženi moderan postupak.

Moderan postupak oblikovanja sustava (1)



Moderan postupak oblikovanja sustava (2)



Validacija i verifikacija

- Validacija:
  - Da li smo napravili ispravan sustav ?
  - Zadovoljava li izgrađeni sustav funkcijske zahtjeve?
  - (*engl. "Are we building the right system?"*)

- Verifikacija:
  - Da li smo ispravno napravili sustav ?
  - Zadovoljava li sustav zahtjeve na ispravan način, t.j. odsustvo kvarova?
  - (*engl. "Are we building the system right?"*)

Sažetak:

- Metodologija oblikovanja programske potpore kontinuirano se razvija i mijenja od 50-tih godina prošlog stoljeća.
- Postoji ozbiljna kriza razvoja programske potpore
- Osnovni cilj programskog inženjerstva je oblikovanje programske potpore sa smanjenom vjerojatnošću neuspjeha
- Oblikovanje programske potpore zasnovano na modelima široko prihvaćeno i podržano alatima

Diskusija