Oblikovanje programske potpore

2014./2015.

Administracija predmeta, uvod i motivacija

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

Pregled tema

- Administracija
  - organizacija nastave
  - sadržaj kolegija
  - literatura
  - ocjenjivanje
- Problemi razvoja programske potpore
- Oblikovanje programske potpore
- Uvod u programsko inženjerstvo

ADMINISTRACIJA

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 i izvođači:
  - Vlado.Sruk@fer.hr (D-332) - nositelj
  - Nikola.Bogunovic@fer.hr (D-309)
  - Alan.Jovic@fer.hr (D-340)

Asistenti, informacije

- Asistenti:
  - Alan.Jovic@fer.hr (D-340)
  - Danko.Ivosevic@fer.hr (D-344, D-335)
  - Nikolina.Frid@fer.hr (D-335)
  - Marko.Horvat2@fer.hr (vanjski suradnik FER-a)

- Informacije:
  - Sve obavijesti u svezi predmeta biti će objavljene na web stranici: http://www.fer.unizg.hr/predmet/opp
  - Predavanja: Moodle https://moodle.fer.hr/course/view.php?id=21
  - Sve obavijesti vezane uz projekt bit će na web stranici projekta: http://www.fer.unizg.hr/predmet/opp/projekt

Predavanja i konzultacije

- Predavanja
- Grupa P01 (Alan Jović)
  - srijeda, 08-10, (14-16), B4
  - četvrtak, 10-12, (16-18), B2
- Grupa P02 (Vlado Sruk)
  - srijeda, 16-18, (10-12), B4
  - četvrtak, 12-14, D1
- Grupa P03 (Nikola Bogunović)
  - srijeda, 12-14, B3
  - četvrtak, 12-14, D2

- Konzultacije profesora i asistenata: FER web stranice: http://www.fer.unizg.hr/predmet/opp/konzultacije
  - najava e-pošta:
    - Naslov/Subject: [OPP] Konzultacije

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 (2 kontrolne točke)
- *Anketa:* središnji i završni upitnik
- *Kontinuirana provjera znanja:*
  - kratke provjere znanja, međuispit, završni ispit

Praktični rad tijekom semestra

- Projekt:
  - **timski rad** u grupi studenata tijekom cijelog semestra.
  - nastavnici određuju početnog koordinatora svakog tima i pridijeljenu temu/projekt.
  - za svaku tim bit će zadužen **jedan asistent** za konzultacije i provjeru znanja.
  - koordinator formira tim od 7 studenata.
  - ako do određenog roka nije formiran tim, to će učiniti nastavnici.
  - po oformljavanju tima – članovi samostalno određuju voditelja
    - to može ali ne mora biti početni koordinator.
    - Ako tim do određenog roka ne odredi voditelja, koordinator postaje voditelj i preuzima organizaciju rada i poštivanje rokova na projektu.
  - projekt se izvodi i dokumentira tijekom cijeloga semestra.
    - predložak obrasca dokumentacije
  - rad na projektu i dokumentiranje izvode se u **dva ciklusa**.
  - na kraju svakog od dva ciklusa asistenti provjeravaju znanje potrebno za izvedbu projekta, postupke ostvarivanja te cjelovitost i razumljivost dokumentacije. Druga provjera uključuje predaju implementiranog projekta.
  - projektna dokumentacija svih grupa na kraju prvog i drugog ciklusa bit će **javno dostupna** na stranicama predmeta
  - sve upute i rokovi bit će objavljeni na web stranicama 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, Moodle, web)
- A. Jović, M. Horvat, I. Grudenić: "UML dijagrami – Zbirka zadataka i riješenih primjera" (sveučilišni priručnik).
- A. Jović, M. Horvat, D. Ivošević, N. Frid: "Procesi programskog inženjerstva" (interna skripta)

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/

Dodatna preporučena literatura

- 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

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:
  - *≥ 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 pismenom ispitu postignuto *≥ 30 bodova*
  - prag za prolaz: *50 bodova*
- Formiranje ocjena
  - **prema utvrđenim pragovima jednakim kao i za kontinuiranu provjeru**

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

- Predmet “Oblikovanje programske potpore” daje osnovna znanja i vještine nužne za postizanje kompetencija u programskom inženjerstvu, a posebice razumijevanje, oblikovanje i vrednovanje programskih sustava.
  - generički modeli procesa programskog inženjerstva.
  - inženjerstvo analize zahtjeva.
  - koncepti programskih arhitektura i paradigme specifikacije.
  - modeliranje objektno usmjerenih sustava (UML).
  - modeliranje ulazno/izlaznih i reaktivnih programskih sustava.
  - ispitivanje programskih sustava.
  - formalna specifikacija i verifikacija željenih obilježja 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, obrada, pohranjivanje, dohvaćanje, upravljanje informacijama
  - sve razvijene ekonomije ovisne o programskoj potpori
- Primjenski programi
- Sistemski programi
- Obožavanje informacijske tehnologije i programa
  - promjena današnjeg društva
  - normiranje?

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:
  - automobil: 10^6 LOC, 80 upravljačkih jedinki, 5 sabirnica
- 2010 – tipični sustav 100 10^6 LOC

Primjer: OS

		SLOC
1993	Windows NT 3.1	$6\ 10^6$
1994	Windows NT 3.5	$10\ 10^6$
1996	Windows NT 4.0	$16\ 10^6$
2000	Windows 2000	$29\ 10^6$
2002	Windows XP	$40\ 10^6$
2003	Windows Server 2003	$50\ 10^6$
2007	Windows Vista	$\sim 50\ 10^6$
2009	Windows 7	$\sim 40\ 10^6$
2012	Debian 7.0	$419\ 10^6$

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: Alarm Clock Bug in Mobile phones

Uzrok?

Razrješavanje problema

- 1968 - software engineering
  - pokušaj odgovora na stanje razvoja kvalitetnih programa na vrijeme i uz zadana sredstva
- 1969 NATO, Glasgow

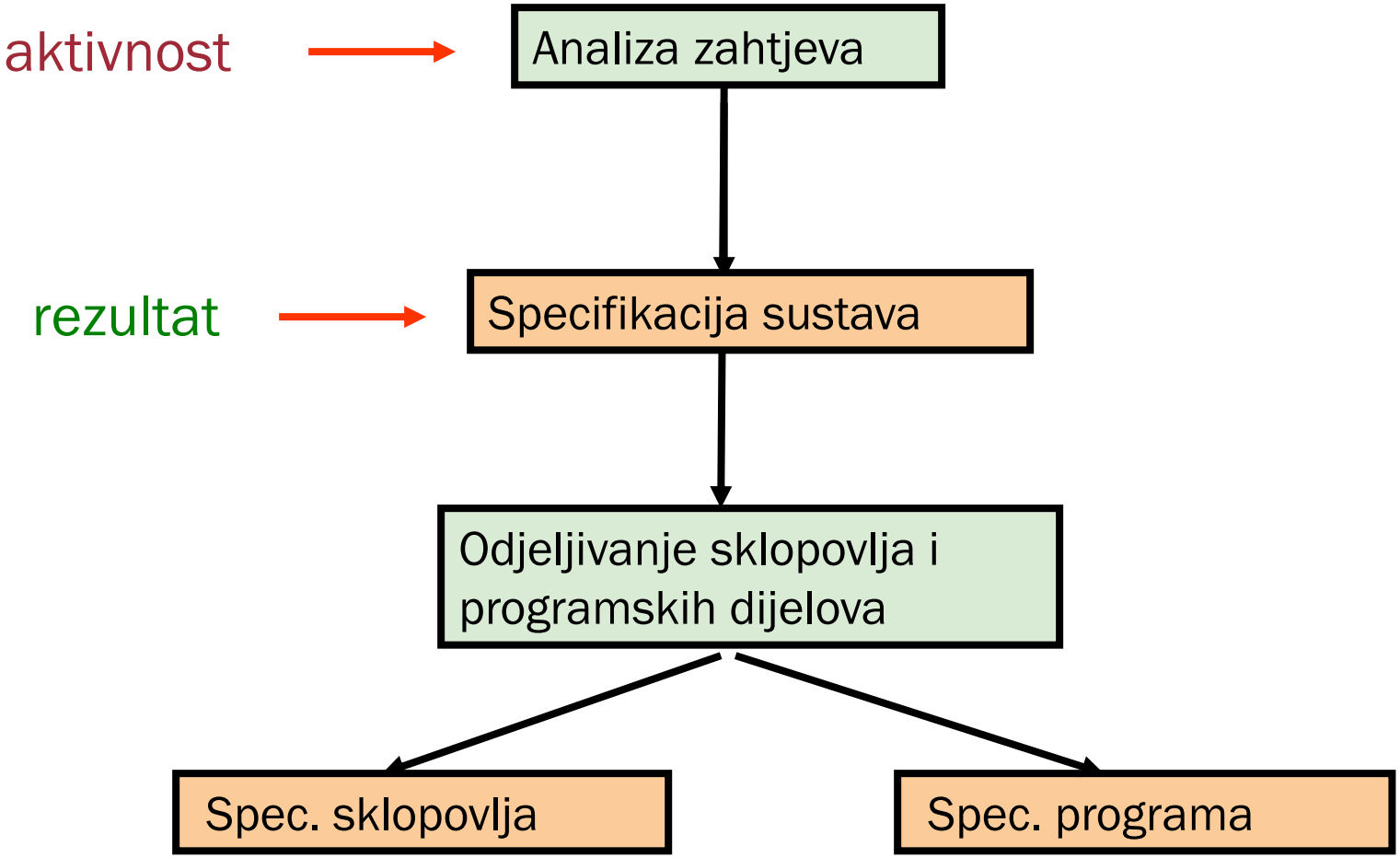
Proces izrade programske potpore

- Pretpostavke:
  - dobar proces -> dobra programska potpora
  - dobar proces -> manji rizik neuspjeha
- Upravljanje rizikom (*engl. Risk Management*)
  - koji su problemi izrade programske potpore?
  - kako smanjiti rizik?

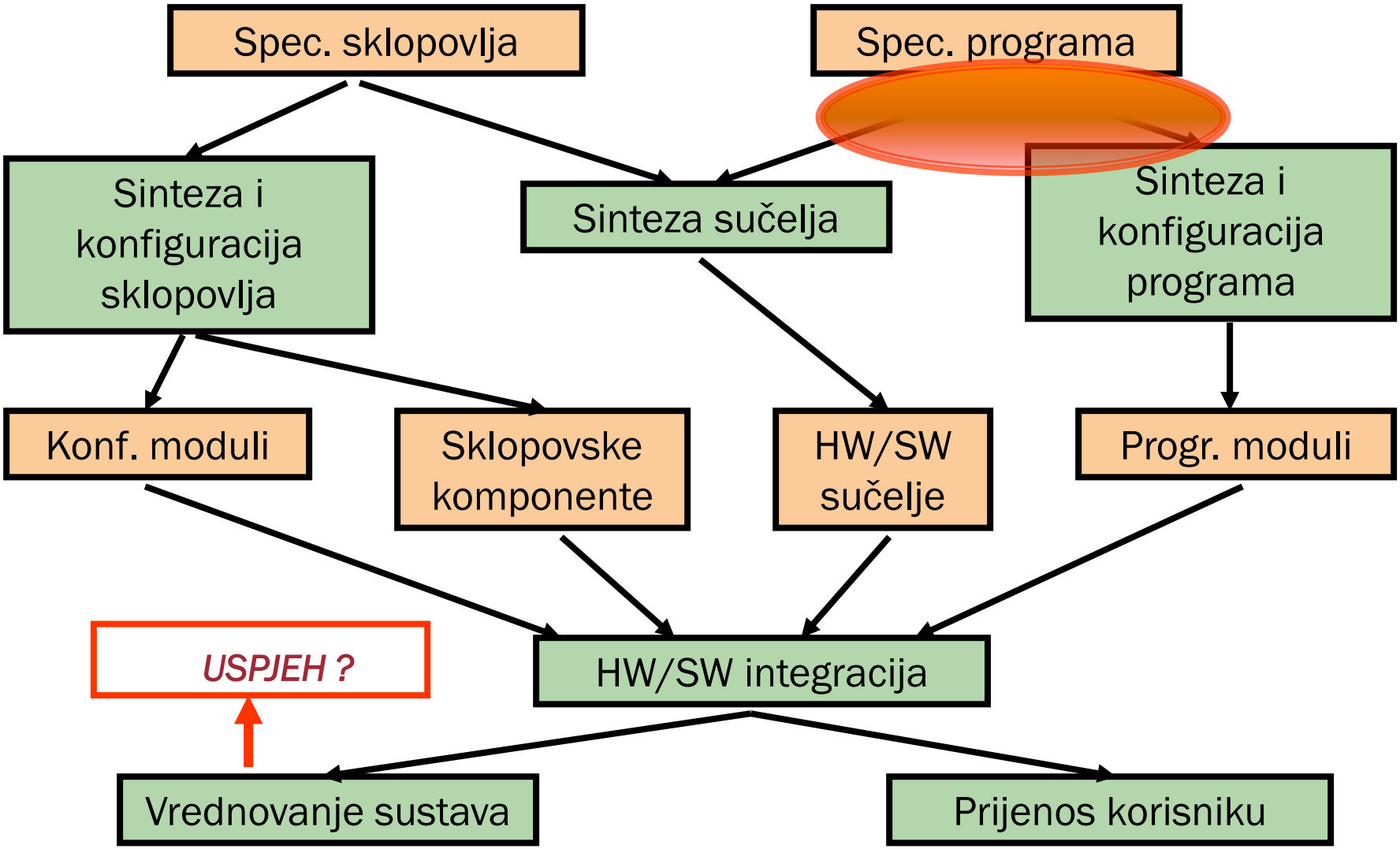
Tradicijsko oblikovanje programske potpore

- 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.

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

Problem ispitivanja sustava

- Ručna ispitivanja (testiranje, provjera):
  - nedovoljno sveobuhvatna
    - 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)!!!

OBLIKOVANJE PROGRAMSKE POTPORE

Principi savladavanja problema složenosti

- *Razdor* problem – implementacija

Kada se javlja?

- *Dekompozicija* - “podijeli pa vladaj”
- *Inkrementalno poboljšanje*
- *Ponovno korištenje dijelova* (*engl. design reuse*)
- *Odvojiti potprobleme*
- *Apstrakcija* - eliminirati nepotrebne detalje
  - uvođenje modela
- *Formalizacija* - matematički određena semantika!

Metodologije oblikovanja zasnovane na modelima

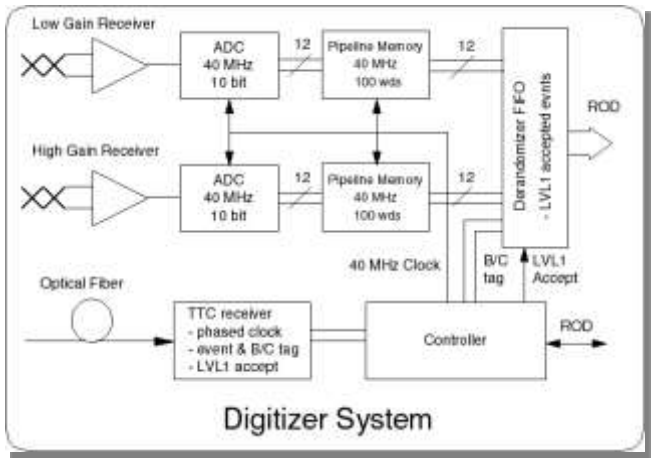
- *engl. Model-driven Development Methods*

- Sljedećih 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

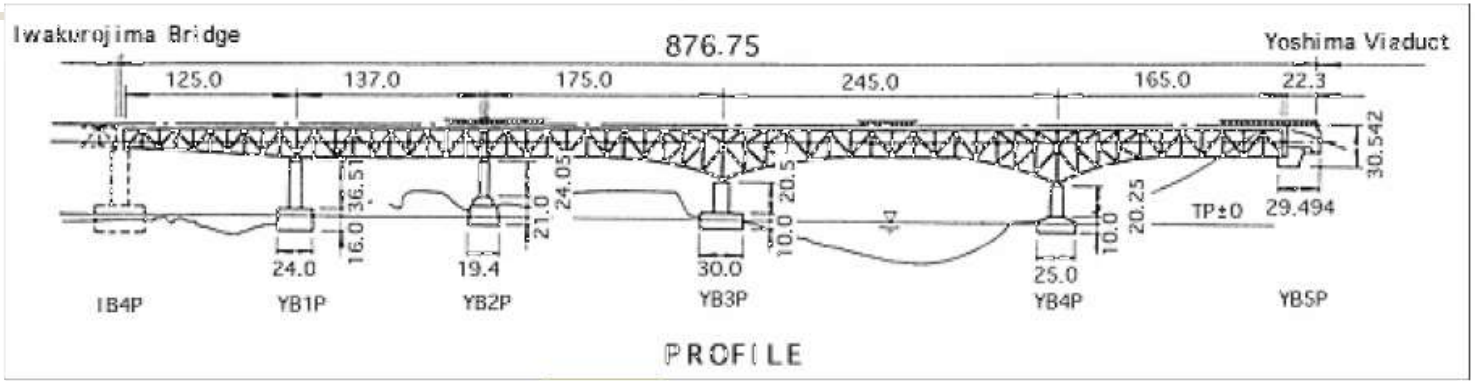
modelirani sustav



funkcionalni model

- Ne prikazuje sve odjednom
- Upotrebljava oblik prezentacije (notacije) jednostavno razumljiv za područje primjene

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 prekomjerna nenamjerna složenost

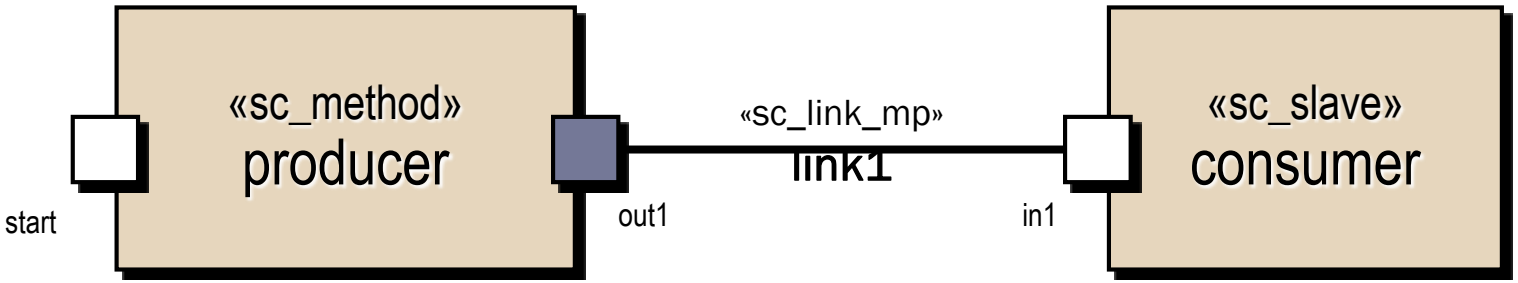
Primjer programa – SystemC

```
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?

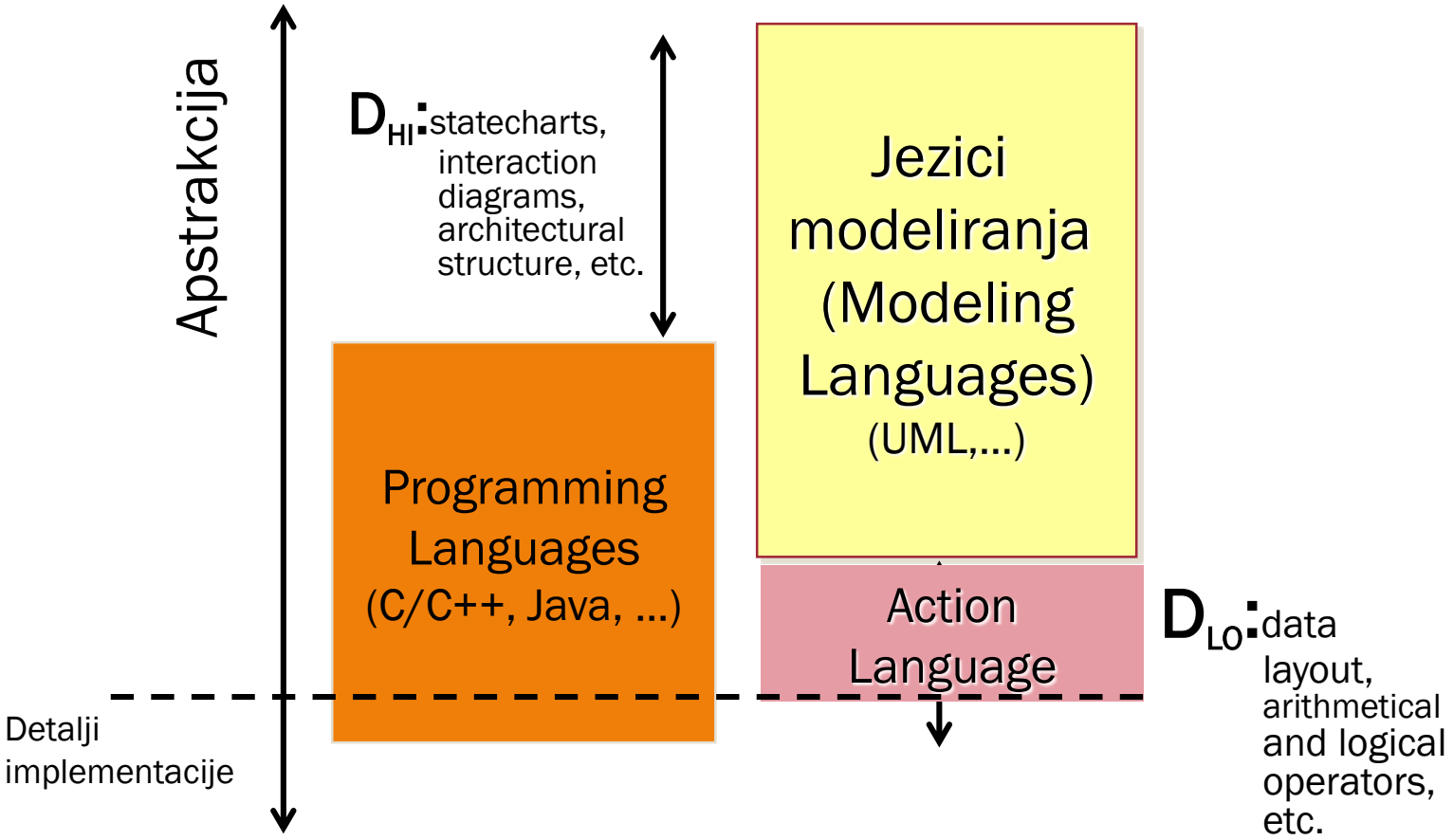
Model programa

- Model:



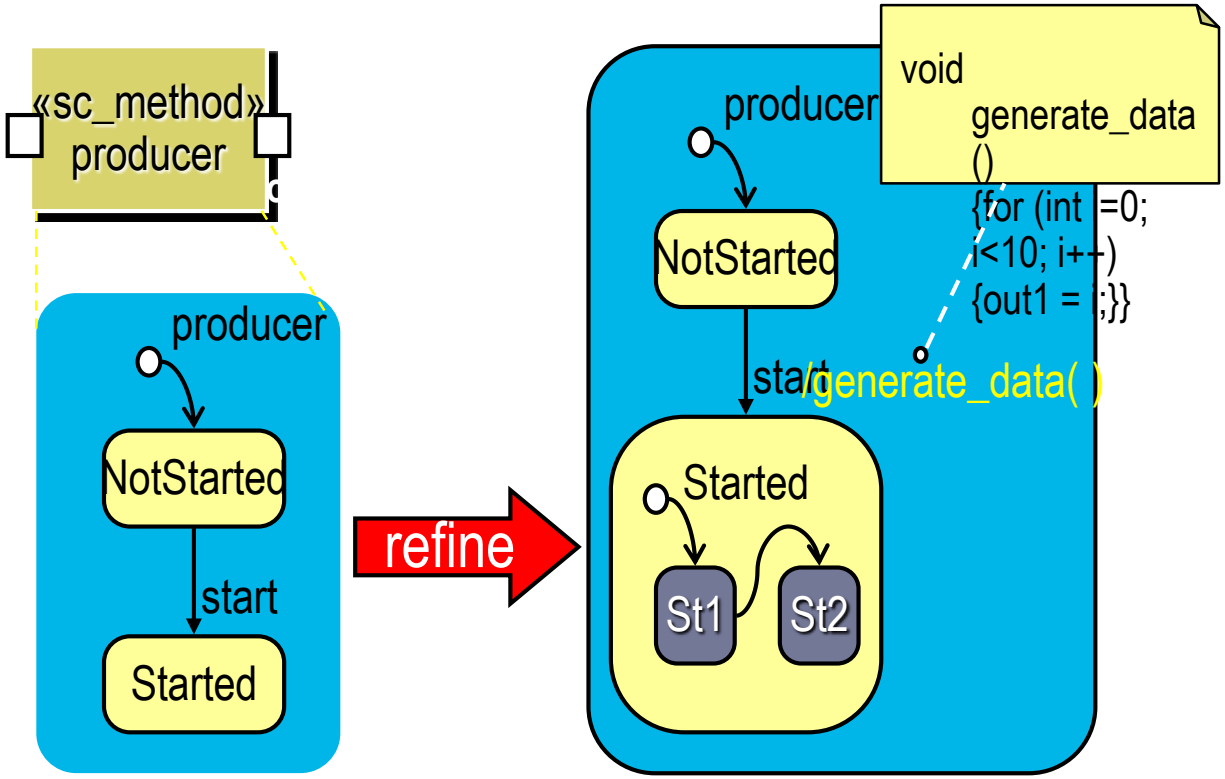
- Koliko je koristan?

Programiranje i modeliranje



Razvoj modela

- U oblikovanju programske potpore modeli se mogu nadograđivati (rafinirati) 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.
- Moguće generiranje apstraktnih modela izravno i automatizirano iz implementacije
  - to osigurava potpunu točnost modela programa
  - model ≡ sustav
- Možemo li to iskoristiti?

Neobično svojstvo programa

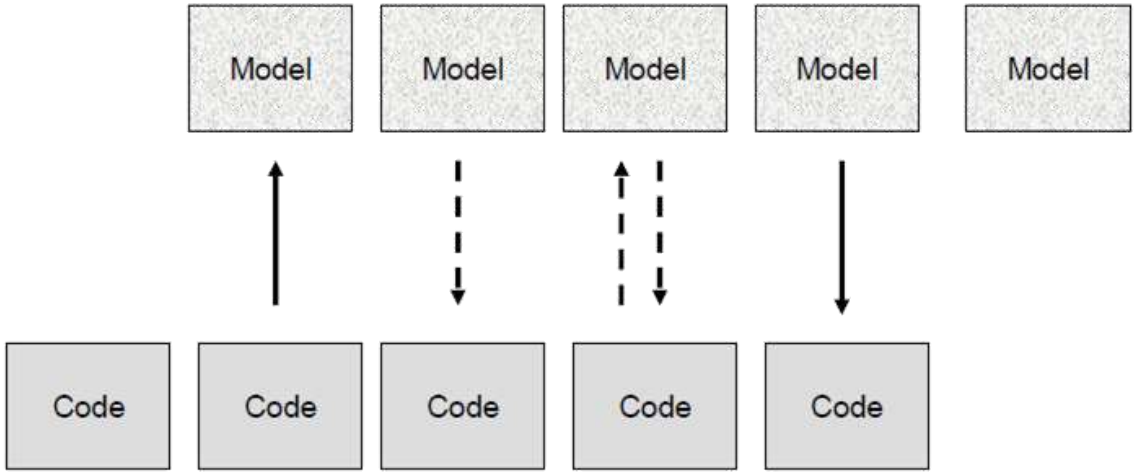
što je najznačajnije?

Posljedice

- Programska potpora je manje ovisna o okolini
  - ne potpuno neovisna
- Individualne mogućnosti dolaze do izražaja
  - produktivnost pojedinca značajno različita
  - nije nužno mjera kvalitete...
  - ...niti inteligencije
- Brz razvojni ciklus
  - uzrokuje nestrpljivost
    - ... nesistematičnost, “hack”
    - velika obećanja

engl. hack: sjeći, udarati, probiti put.

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 ....

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 pogodne za formalnu analizu.
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.
- Dokumenti se uobičajeno podvrgavaju ocjeni i reviziji (promjeni).

Rezultat oblikovanja projekta na ovom kolegiju je također dokument

Arhitektura programske potpore

- Temeljem izlučivanja zahtjeva ***analizira*** se nekoliko stilova arhitekture programske potpore kako bi se došlo do optimalnog izbora s 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*).

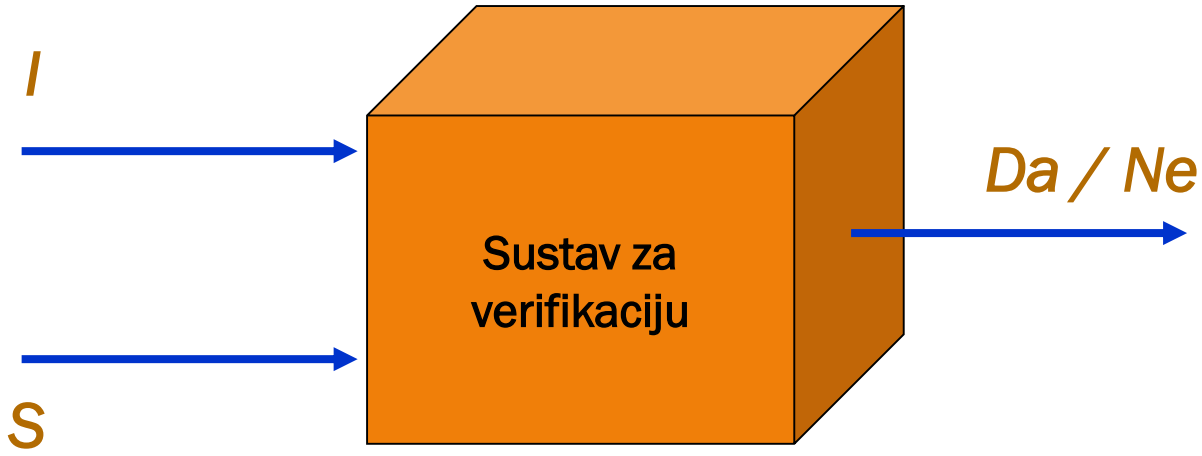
Formalne metode provjere

- 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 (FM)* 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 pomoću Emerson - Clarke notacije:

    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 namjere korisnika.
   - *S* je nekompletan ili neispravan (opisuje krivo ponašanje).

FV u praksi

- Poluvodička industrija (ASIC i VHSIC):
  - 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 koje se naziva Formalne metode (FM).

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

Formalne metode

- Formalne metode obuhvaćaju tehnike matematičkog modeliranja koje se primjenjuju u oblikovanju računalnih sustava.
  - **formalna specifikacija** sustava,
  - analiza i **verifikacija** specifikacija,
  - **verifikacija** programa.
  - **formalna sinteza** programa
- “ Formal methods are mathematical approaches to software and system development which support the rigorous specification, design and verification of computer systems.” [http://www.fmeurope.org/]

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 osnove***
  - 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.

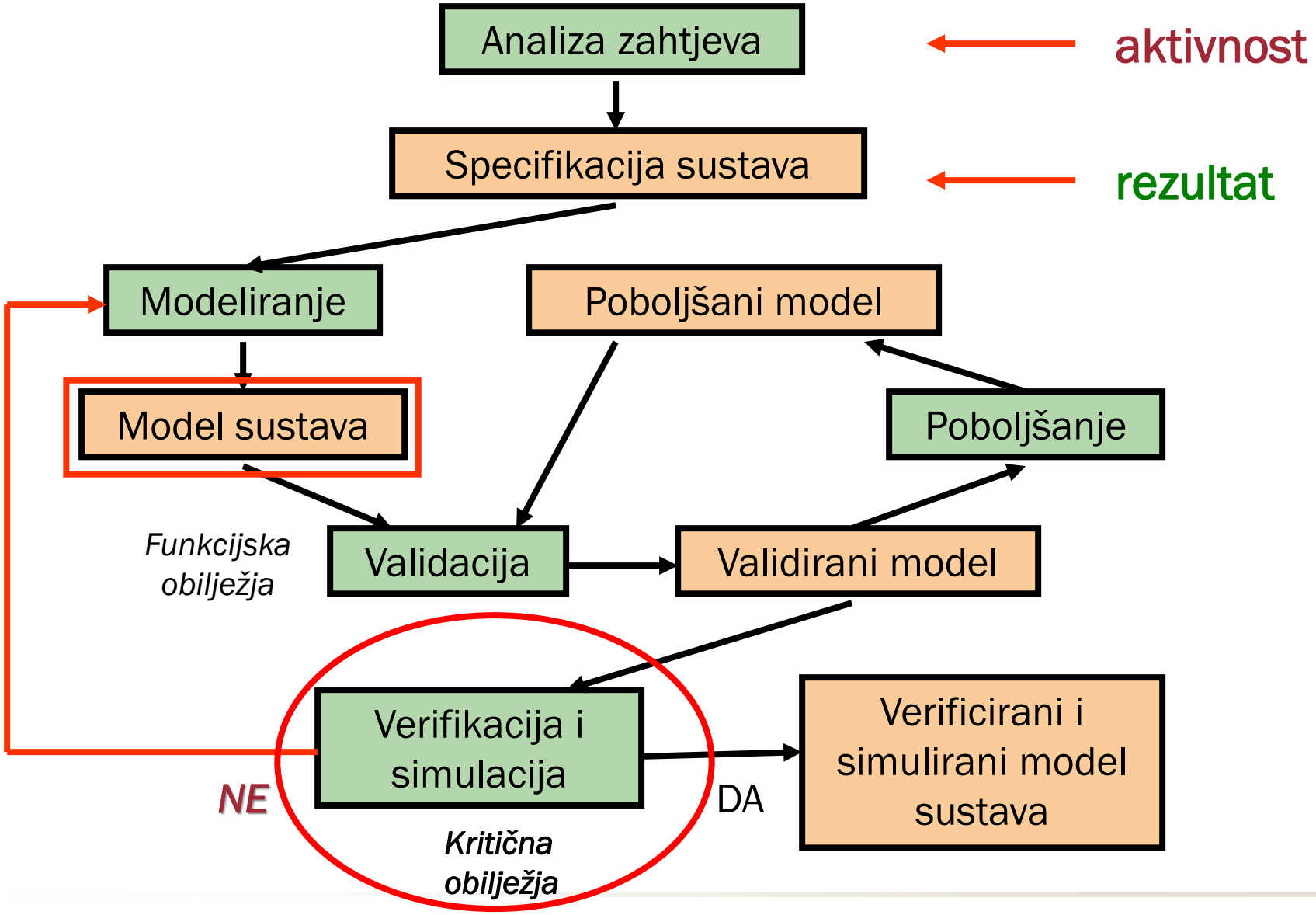
Trend i problemi

- 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

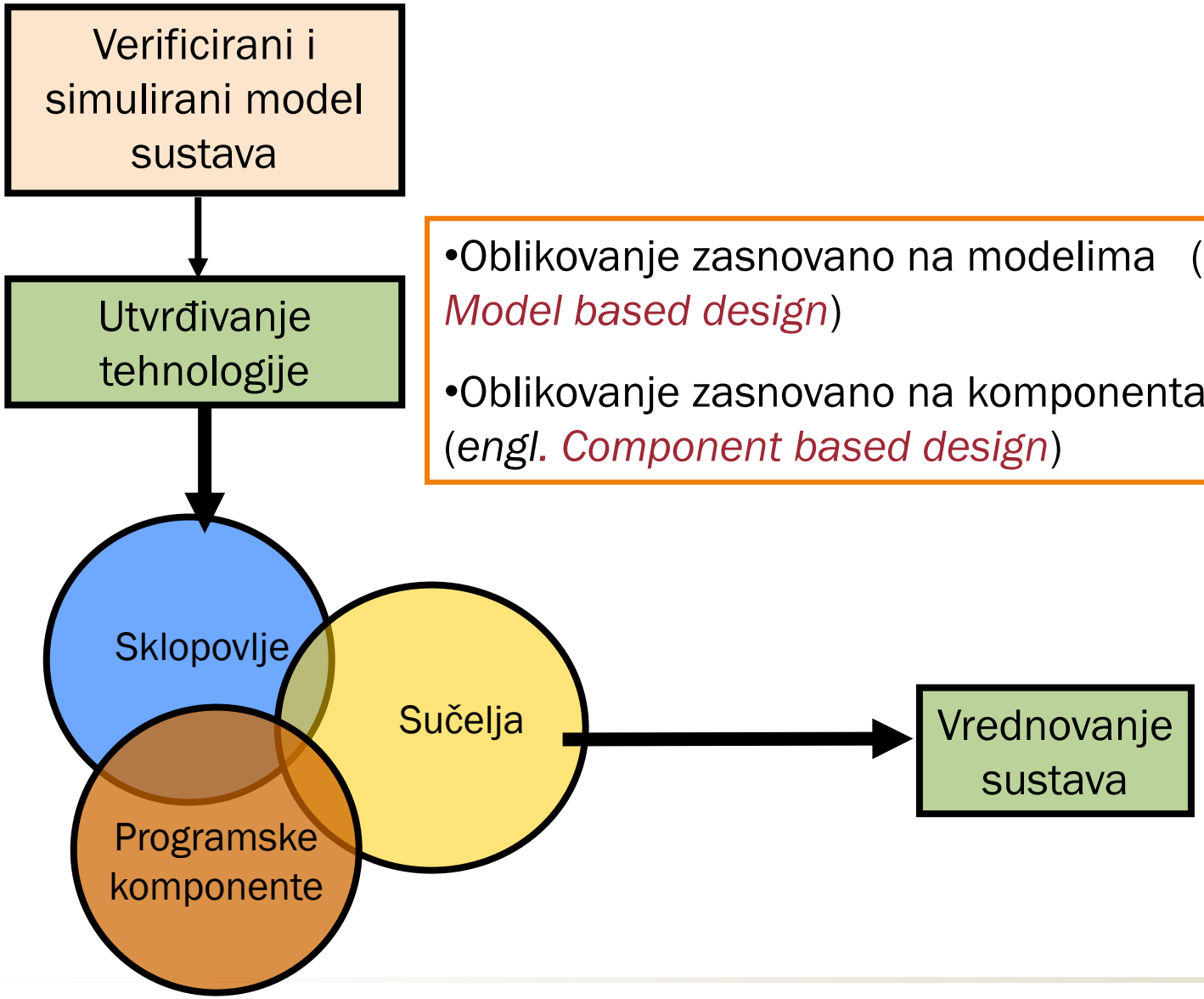
Analiza postojećeg stanja

- Temeljem analize nedostataka tradicijskog postupka oblikovanja sustava i izvedenih zaključaka slijedi predloženi moderan postupak.
- Elementi postupka:
  1. propisani strukturirani postupci
  2. dokumentiranje
  3. modeliranje (apstrakciju)
  4. višestruku uporabu komponenata
  5. formalna verifikacija modela + tradicijsko ispitivanje

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, tj. odsustvo kvarova?
  - (*engl. "Are we building the system right?"*)

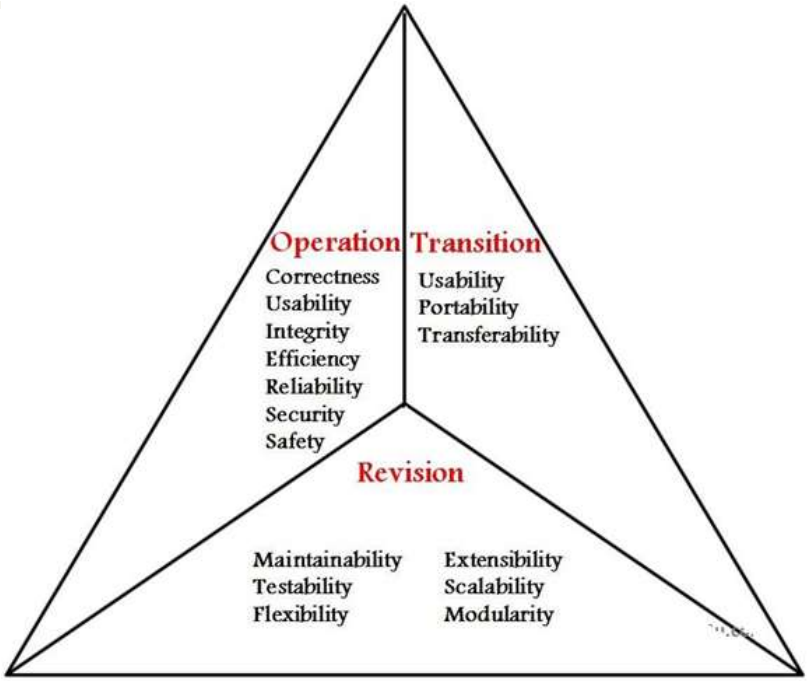
Zaključak

- Metodologija oblikovanja programske potpore kontinuirano se razvija.
- Postoji ozbiljna kriza razvoja programske potpore.
- Osnovni cilj programskog inženjerstva je oblikovanje programske potpore sa smanjenom vjerojatnošću neuspjeha.
- Moderno oblikovanje programske potpore zasnovano na modelima duboko u praksi i podržano alatima.
  - Ima mjesta za poboljšanja ☺

Diskusija

Kvaliteta programske potpore

- 1. poznavati zahtjeve
- 2. ispitivanje kvalitete
- Što je najznačajnije?
- Svojstva programske potpore:
  - Radna svojstva (engl. Operational
    - kako radi i uporabljivost?
  - Promjenjivost/Revizija
    - jedostavnostispravljanja, ispitivanja, v
  - Prenosivost/Tranzicijska
    - podrzavanje raznih platformi, ponovna uporabljivost ...



- Metrika
  - McCall's Software Quality (1977)