

Oblikovanje programske potpore

O čemu je riječ u ovom predmetu?

Copyright Nikola Bogunovic, FER

1

Signifikantan događaj (1985-1987)

**Therac-25: sustav za terapiju radioaktivnim zračenjem
(elektroni ili fotoni do 25 MeV)**

6 fatalnih pogrešaka (primljene prekomjerne doze)

- Najgori akcident u 40 godina terapije zračenjem
- Program:
 - DEC PDP 11 assembler
 - konkurentni procesi
 - “test and set” nije bila nedjeljiva operacija
 - višestruki upis u zajedničku varijablu

Copyright Nikola Bogunovic, FER

2

Software "HALL OF SHAME":

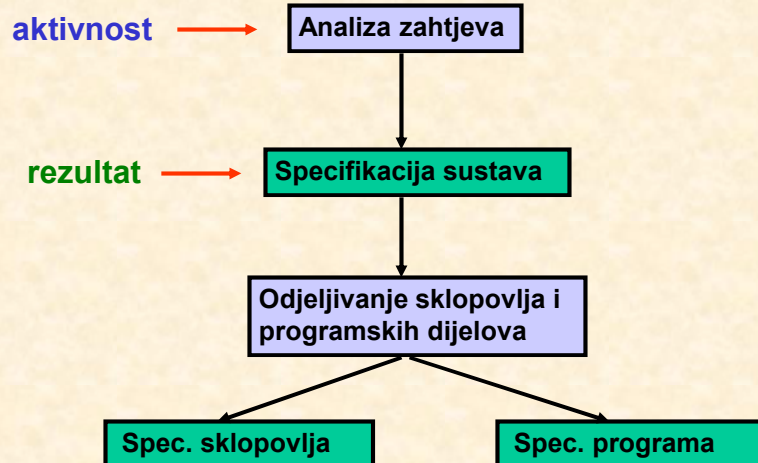
- 1993: London Stock Exchange (\$ 600 M, canceled)
- 1996: Arianespace (Ariane 5 rocket explosion, \$350 M)
- 1999: State of Mississippi (Tax system, \$185 M loss)
- 2001: Nike Inc. (Supply chain management, \$100 M loss)
- 2002: McDonald's (Purchasing system canceled, \$170 M)
- 2004: Hewlett-Packard (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)

. . .

Copyright Nikola Bogunovic, FER

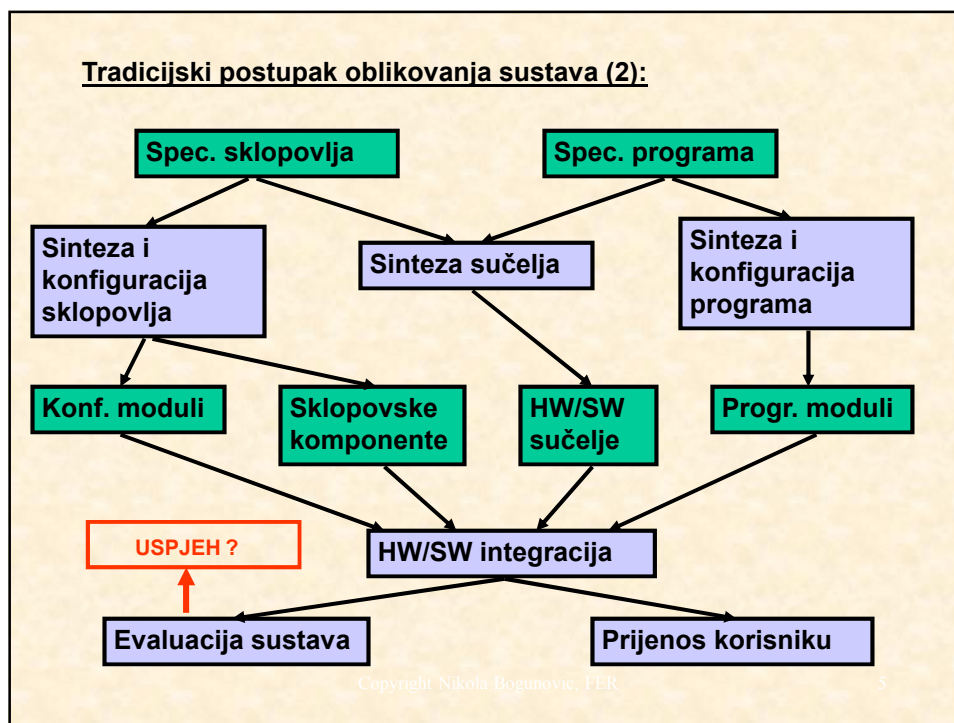
3

Tradicijski postupak oblikovanja sustava (1):



Copyright Nikola Bogunovic, FER

4



Nedostaci tradicijskog postupka oblikovanja sustava

- Zahtjevi nisu formalizirani te je nemoguće postići preciznost, a neodređenost vodi u nekompatibilnost.
- Izgradnja “ad hoc” prototipa ne omogućuju dublju analizu sustava.
- Procjena performansi (neformalni pristup) daje pogrešne vrijednosti.
- Dijeljenje na sklopovski i programski dio nije poduprto formalnim transformacijama i postupcima donošenja odluka.
- Dokumentacija se zbog neodređenosti teško interpretira.

“**Formalni**” = temeljeni na matematičkim teorijama (logika, teorija automata, teorija grafova, ...).

Nedostaci tradicijskog ispitivanja (testiranja) sustava

Ručno ispitivanje složenog sustava je nedovoljno duboko jer ljudski um nije u mogućnosti predvidjeti sve moguće interakcije.

- Cijena testiranja čini najveći dio cijene produkta.
- Pokrivenost skupa testiranih stanja se smanjuje.
- “**Corner cases**” se sve teže identificiraju.
- Vrijeme stavljanja proizvoda na tržište (“**time to market**”) smanjuje pokrivenost već ionako malog skupa testiranih stanja i podupire mentalitet “kolektivne krivnje”.

Ispravnost provjeravana **simulacijama i testiranjem može samo dokazati postojanje pogreške (“bug”) ali nikada njeno nepostojanje (Dijkstra) !!**

Copyright Nikola Bogunovic, FER

7

Što učiniti ?

Treba uporabiti principe savladavanja složenosti:

- Implementirati **višestruko korištene** dijelove (“design reuse”).
- **Odvojiti probleme:**
 - logički - fizikalni
 - logički - vremenski slijed (“timing”)
 - funkcije - komunikacije
- **Dekomponirati** (“podijeli pa vladaj”) na dijelove (**komponente**).
- **Inkrementalno** poboljšavati.
- **Apstrakcijom** eliminirati nepotrebne detalje = uvođenje modela.
- Uvesti matematički određenu semantiku = formalizacija.

Problem: Kako to uključiti u proces oblikovanja sustava ?

Copyright Nikola Bogunovic, FER

8

CrossTalk, The Journal of Defense Software Engineering

January, 2008

Computer Science Education: Where Are the Software Engineers of Tomorrow?

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.**

9

Prijedlog modernog postupka oblikovanja sustava

Slijedećih nekoliko slika preuzeto od:

Bran Selic

- IBM Distinguished Engineer at IBM Rational.
- Adjunct professor at Carleton University in Ottawa, Canada.
- Over 30 years of experience in designing and implementing large scale industrial software systems.
- Pioneered the application of model-driven development methods in real-time applications.
- He is chair of the OMG team responsible for the UML 2.0 standard.

Copyright Nikola Bogunovic, FER

10

What has Changed from a Developer's Perspective...

- ◆ Changes with the greatest impact:
 - Hardware: speed, reliability, cost, capacity
 - Connectivity between computers
 - Team programming environments
 - Human-computer interfaces
 - Sophistication of development tools
 - Complexity of applications
- ◆ Most significant non-change:
 - Level of abstraction of programming

4

Q: Why is Writing Correct Software so Difficult?

A: COMPLEXITY!

Modern software is reaching levels of complexity encountered in biological systems; sometimes comprising systems of systems each of which may include tens of millions of lines of code

...any one of which may bring down the entire system at great expense

9

A Bit of Modern Software...

```
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);}};
```

Can you see the
architecture?

11

...and its Model



Can you see it now?

12

Use of Models in Engineering

- ◆ Probably as old as engineering (c.f., Vitruvius)
- ◆ Engineering model:

A reduced representation of some system that highlights the properties of interest from a given viewpoint



- We don't see everything at once
- What we do see is adjusted to human understanding

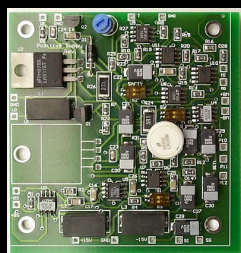
15

Engineering Models

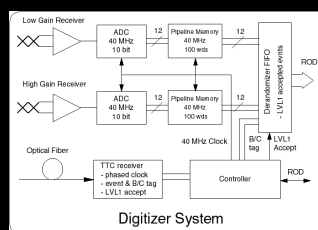


- ◆ Engineering model:

A reduced representation of some system that highlights the properties of interest from a given viewpoint



Modeled system

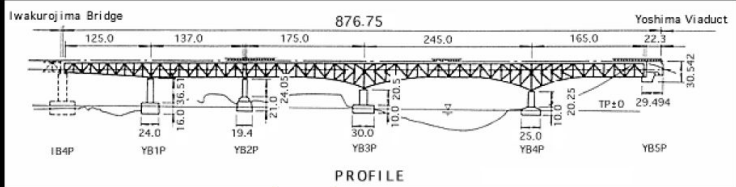


Functional Model

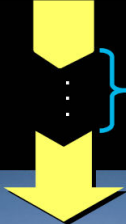
- ◆ We don't see everything at once
- ◆ We use a representation (notation) that is easily understood for the purpose on hand

16

IBM



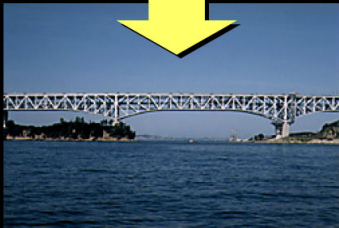
PROFILE



Differences due to:

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

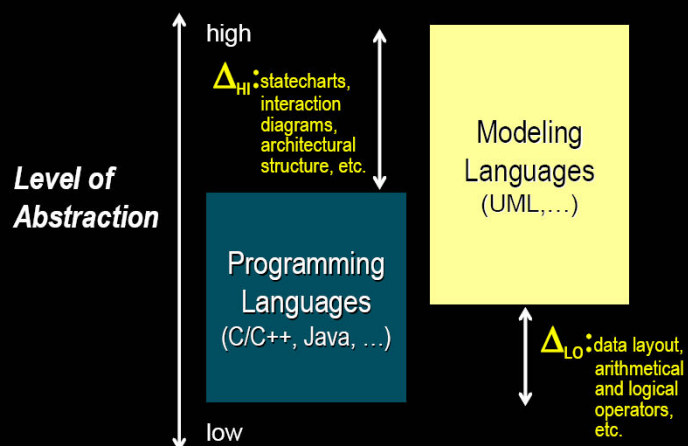
Can lead to serious errors and discrepancies in the realization



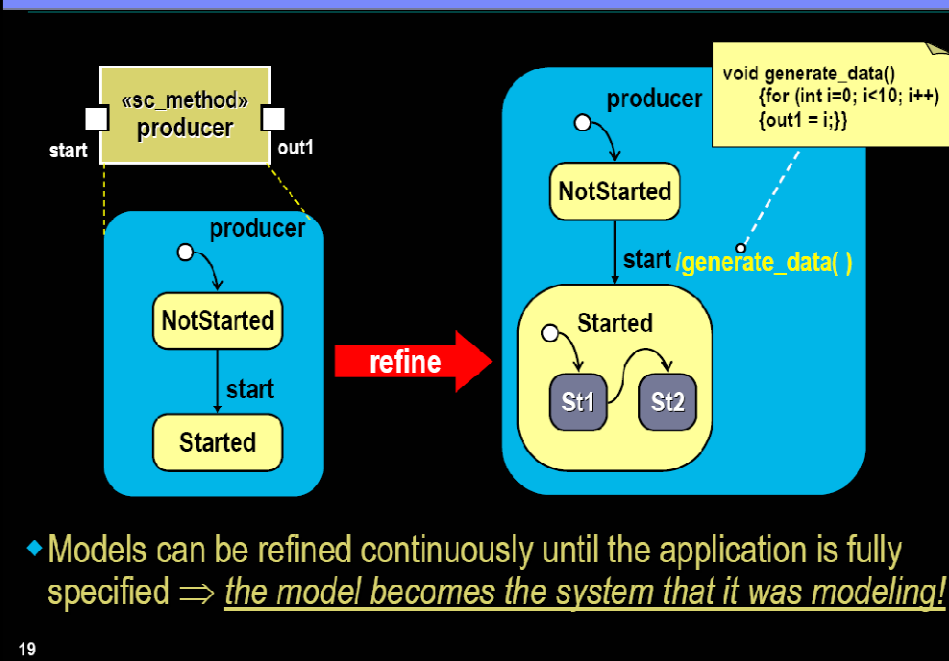
17

IBM Software Group | **Rational** software

Modeling Languages vs Programming Languages



Model Evolution: Refinement

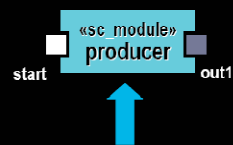


19

Model-Driven Development

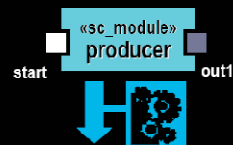
- ◆ **Model-Driven Development (MDD):** An approach to software development in which the focus and primary artifacts of development are models (as opposed to programs)
- ◆ Based on 2 time-proven approaches:

(1) ABSTRACTION



```
SC_MODULE(producer)
{
  sc_inslave<int> in1;
  int sum; //
  void accumulate () {
    sum += in1;
    cout << "Sum = " <<
    sum << endl;
  }
}
```

(2) AUTOMATION



```
SC_MODULE(producer)
{
  sc_inslave<int> in1;
  int sum; //
  void accumulate () {
    sum += in1;
    cout << "Sum = " <<
    sum << endl;
  }
}
```

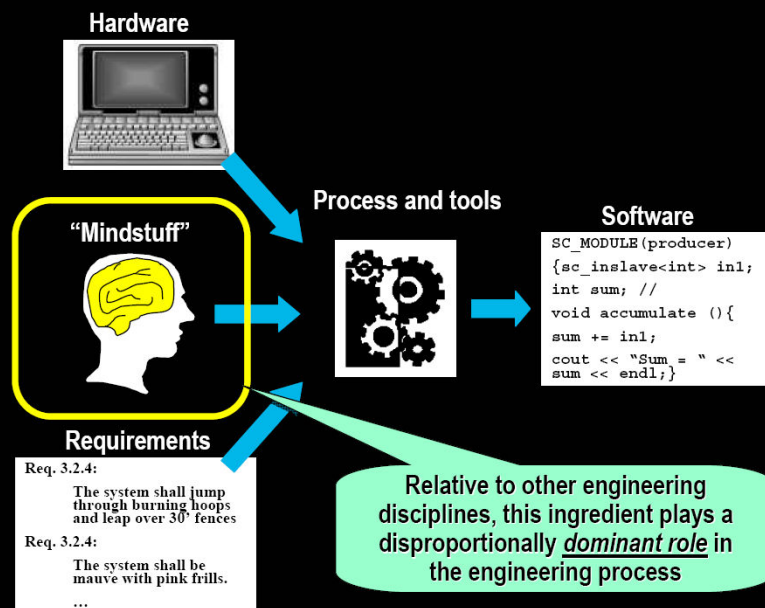
20

MDD: State of the Art and Adoption

- ◆ Systems using fully automated code generation from models written using a modeling language:
 - Size: Complete systems equivalent to ~ 5 MLoC and involving several hundred developers
 - Performance: within ± 5 -15% of equivalent manually coded system
 - There are many similar examples of successful MDD projects
- ◆ Yet, MDD is practiced by only a small percentage of software developers
 - In fact, the vast majority dislike it and many actively oppose it!
- ◆ Why?

21

The Idiosyncrasies of Software – 1



22

Some Consequences

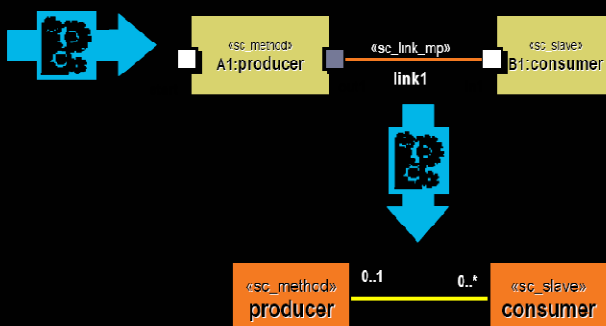
- ◆ Products are much less hampered by physical reality
 - ...but, not completely free
- ◆ The effects of aptitude differences between individuals are strongly accentuated
 - Productivity of individuals can differ by an order of magnitude
 - Not necessarily a measure of quality
 - ...or intelligence
- ◆ The path from conception to realization is exceptionally fast (edit-compile-run cycle)
 - Often leads to an impatient state of mind
 - ...which leads to unsystematic and hastily conceived solutions (hacking)
 - Also yields a highly seductive and engrossing experience
 - ...so that, often, the medium becomes the message

23

The Idiosyncrasies of Software – 2

- ◆ In all other engineering disciplines abstractions are artifacts that are necessarily distinct from the systems that they abstract
 - Results in divergence and inaccuracy of abstractions
- ◆ *Uniquely, in software, the abstraction can be integrated with its system and can be extracted automatically*

```
SC_MODULE(producer)
{
    sc_outmaster<int> out1;
    sc_in<bool> start; // kink-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)
```



24

The Remarkable Aspects of Software



- ♦ *Software has the unique property that it allows us to evolve abstract models into full-fledged implementations without changing the engineering medium, tools, or methods!*
- ♦ *It also allows us to generate abstract views directly and automatically from the implementations*

⇒ This ensures perfect accuracy of software models; since the model and the system that it models are the same thing

25

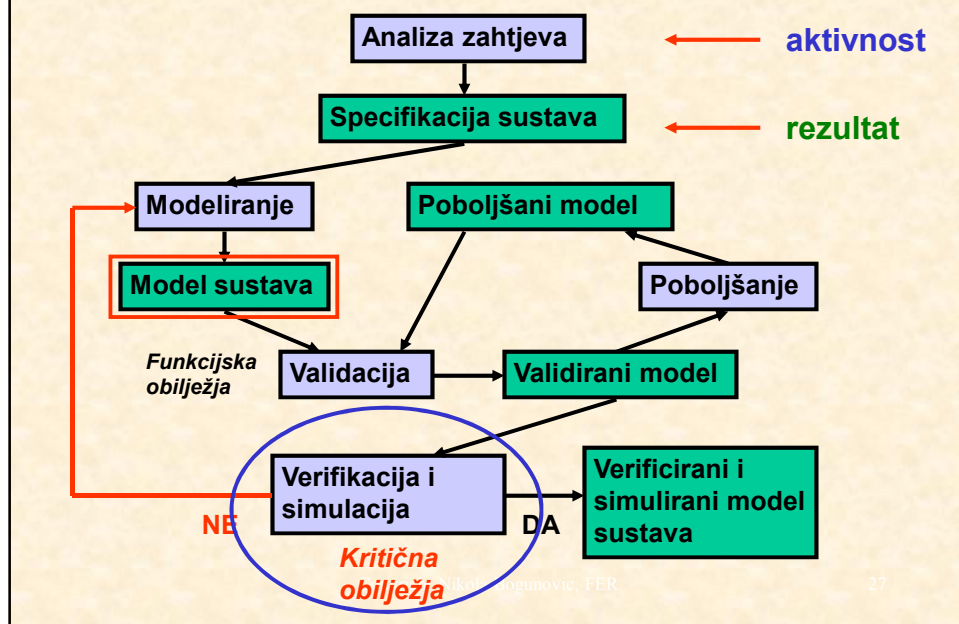
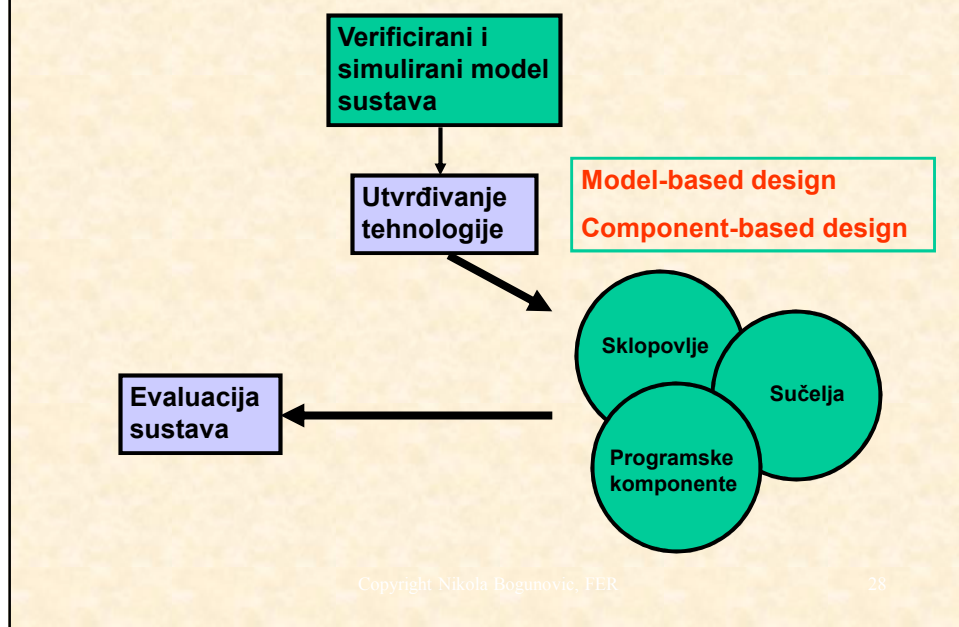
IBM Software Group | Rational software

Zaključak:

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. Programski produkt oblikovati u manjim cjelinama (komponentama).
5. Uz tradicijsko ispitivanje (testiranje) uvesti formalne metode provjere (verifikacije) modela programske potpore.

Copyright Nikola Bogunovic, FER

26

Moderan postupak oblikovanja sustava (1):**Moderan postupak oblikovanja sustava (2):**

Validacija i verifikacija

Validacija: da li smo napravili ispravan sustav ?
 (da li sustav zadovoljava funkcijske zahtjeve)
 (engl. "Are we building the right system?")

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

Copyright Nikola Bogunovic, FER

29

Temelji formalne provjere - verifikacije (FV) **modela** sustava

Dva objekta:

S - specifikacija (što sustav treba raditi) - **formal. spec.**

I - implementacija (kako to sustav radi) - **formal. model**

Odredi da li: **I odgovara S.**

"Odgovara":

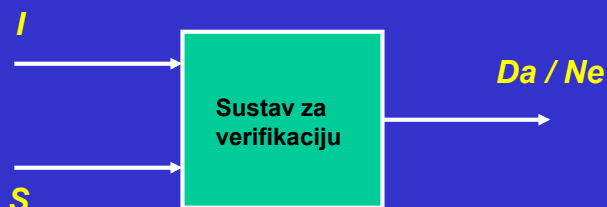
ekvivalencija, simulacijske relacije, **logička zadovoljivost**, logička implikacija, implementacijske relacije, ...

Copyright Nikola Bogunovic, FER

30

Formalna verifikacija

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



Copyright Nikola Bogunovic, FER

31

Primjeri:

Ulazno/izlazni programi:

$S:$ $y = x^2$
 $I:$ $y = 1 + 3 + \dots (2x - 1)$

Reaktivni programi:

trajna interakcija s okolinom (operacijski sustavi, upravljanje avio letovima, upravljanje nuklearnim reaktorom, ...)

$S, I:$ potreban je formalni opis njihovog ponašanja **tijekom vremena** !?

Copyright Nikola Bogunovic, FER

32

Kako **specificirati** “što želimo” (formalan opis) za reaktivne sustave:

Logika predikata prvoga reda (X je var., b,c su konstante):

$$\forall X [c \Leftrightarrow (X \wedge b)]$$

Nedostatak: nema vremenskih obilježja.

Proširenje: **Vremenska logika** za specifikaciju **S**

“Signal REQ mora biti istinit i ostati istinit sve dok signal GRANT ne postane istinit”

Modeli **implementacije** (/), opisi su ekvivalentni:

Regularni jezici

Automati s konačnim brojem stanja

Copyright Nikola Bogunovic, FER

33

Vremenska logika

Specifikacija (ponašanje u budućnosti), npr:

“Nikada se ne smije dogoditi

da su vrata otvorena dok se lift kreće”,

t.j (**otvorena_vrata** \wedge **lift_se_kreće**) mora uvijek biti neistinito.

Primjer specifikacije:

AG \neg (**otvorena_vrata** \wedge **lift_se_krece**)

Emerson - Clarke notacija (gornji primjer)

Pnueli - Manna notacija: \Diamond \square - ne u ovom kolegiju

Copyright Nikola Bogunovic, FER

34

Pažnja:

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 **S**.

Temeljno: radi se s **modelima** (apstrakcijama) stvarne implementacije

Copyright Nikola Bogunovic, FER

35

Mogući razlozi neispravnosti postupka i rezultata formalne verifikacije

1. Razlika između matematičkih objekata i realnosti

- Model / nije obuhvatio bitna obilježja realne implementacije (npr. ograničena duljina riječi, precizna informacija o vremenskim odnosima i sl.).
- **S** pretpostavlja suviše jaka ograničenja okoliša.

2. Neadekvatna specifikacija željenih obilježja

- S** je nekompletan ili neispravan (opisuje krivo ponašanje a ne intencije korisnika).

Copyright Nikola Bogunovic, FER

36

Okosnice za FV

Algoritamska verifikacija

(provjera modela, *engl. model checking*)

Istraživačka tehnika za strojeve s **konačnim** brojem stanja.

Pobrojavanje (enumeriranje i **simboličke varijante**).

Pokazat će se u ovom kolegiju.

Postupci zasnovana na dokazivanju teorema (ATP)

Primjenjivi na sustave s **beskonačnim** brojem stanja.

Zahtijevaju interakciju korisnika (ekspertno znanje).

Neće se razmatrati u ovom kolegiju.

Copyright Nikola Bogunovic, FER

37

Algoritamska verifikacija

Treba dokazati da vremenski specificirano obilježje vrijedi tijekom svih ponašanja sustava (*engl. computations*).

Dokaz da obilježje **p** vrijedi tijekom svih ponašanja sustava:

Konstruiraj sva stanja dosegljiva iz početnih.

Provjeri da svako stanje zadovoljava obilježje **p**.

Temeljni problem: izračun svih dosegljivih stanja
("reachability analysis")

Danas postignuto: sustavi sa $\sim 10^{500}$ stanja

Copyright Nikola Bogunovic, FER

38

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.

Copyright Nikola Bogunovic, FER

39

FV u praksi:

Poluvodička industrija (ASICs i drugi sklopovi VHICS):

formiranje istraživačkih i razvojnih odjela, uvođenje FV

Komunikacijski protokoli

često su formalno verificirani

Sigurnosno-kritični sustavi

uskoro će se zahtijevati FV

Programsko inženjerstvo

slabo prihvaćanje FV tehnika

Copyright Nikola Bogunovic, FER

40

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).

Copyright Nikola Bogunovic, FER

41

Spas u formalnim metodama (FM) ?

(US Computer Science and Technology Board)

FM trebaju biti ključna inženjerska vještina u računarstvu

koja povećava kakvoću produkta i istovremeno

smanjuje vrijeme stavljanja produkta na tržište

(“time to market”),

te se mora uključiti u obrazovanje CS i EE.

Copyright Nikola Bogunovic, FER

42

FM džungla

ACL2, ACP, ASR, Action Semantics, Argos, ASM, ADLT, BDDs, B, Boyer-Moore, Caesar/Aldebaran, CCS, Circal, COLD, Coq, COSPAN, CSP, FDR2, CWB, DisCo, DC, Estelle, EVES, GIL, HOL, HyTech, IMPS, I/O Automata, ITL, Isabelle, JAPE, KIV, Kronos, LAMBDA, LARCH, LeanTaP, LEGO, LOTOS, Lustre, MALPAS, Meije, Mizar, μ CRL, Murphi, NP-tools, Nqthm, Nuprl, Obj, Otter, Petri Nets, PI-calculus, Pobl, ProofPower, PVS, RAISE, Rapide, Refinement Calculus, SDL, SGM, Signal, **SMV**, SPARK, SPIN, Step, TAM, TAM97, Temporal-Rover, TLA, TPS, TRIO, TTM/RTTL, Unity, Uppaal, VeriSoft, VDM, VIS, Z, ...

Copyright Nikola Bogunovic, FER

43

Gdje smo u evoluciji formalnih postupaka oblikovanja

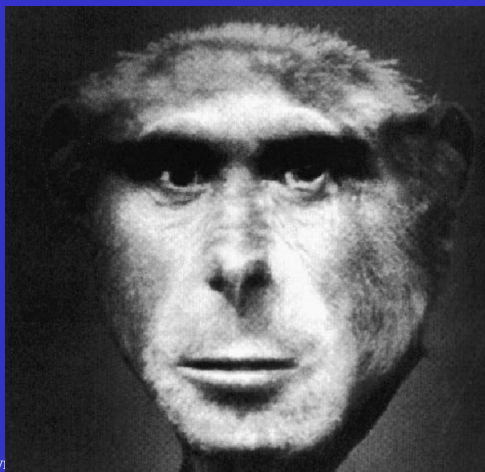
Semantika modela sustava ima dobru teorijsku osnovicu ali slabu primjenu.

Postupci oblikovanja sustava imaju intenzivnu primjenu ali slabu teorijsku osnovicu.

Budući ključni rad:

razumjeti,
razvijati i
uspoređivati
različite
**računalne
modele**

("models of
computation")



Copy