Testiranje u programskom jeziku Java (JUnit) i testiranjem upravljano programiranje (TDD)

- Na održavanje programske potpore troši se i preko 50% vremena tijekom života programske potpore
- Na testiranje prije isporuke otpada > 40% vremena razvoja
- Svaki od modela razvoja programske potpore sastoji se u osnovi od sljedećih koraka (ciklus razvoja programske potpore):
    - ideja
    - razrada
    - kodiranje
    - testiranje
    - održavanje
- Validacija - radi li program ono za što je namijenjen?
- Verifikacija - kako radi ono za što je namijenjen ?
- Ciljevi testiranja:
    - Testiranje mora utvrditi (i potvrditi) kvalitetu programske potpore
    - Testiranje je jedna od aktivnosti koje se provode u cilju povećanja kvalitete programske potpore
- Osiguranje kvalitete programske potpore provodi se validacijom i verifikacijom
- Testiranje je sastavni dio osiguranja kvalitete programske potpore
- Testirati znači pronaći razuman broj testova koji utvrde da li je: očekivani rezultat ≡ dobiveni rezultat
    - ne postoji program koji nema pogrešaka
    - potpuno testiranje zahtijeva beskonačno vrijeme testiranja!
    - nepostojanje pogreške samo znaci da je još nismo otkrili tijekom testiranja ili tijekom upotrebe programske potpore!
    - pogreške treba otkriti što ranije!
- Program ≠ programska potpora
    - Kôd odnosno "instrukcije" NISU programska potpora jer osim instrukcija programska potpora sadržava:
        - izvorni kôd programa (eng. source)
        - dokumentaciju
        - testove
        - izvješća o testiranju
        - izvješća o održavanju
        - specifikaciju
- izgled tipičnog programskog sustava za testiranje
    - hijerarhija manjih cjelina
    - cjelina može biti podsustav, funkcijski blok, funkcijska jedinica ili cjelina može biti modul, komponenta
    - način testiranja bi trebao biti neovisan o programskom jeziku
    - način testiranja mora voditi računa o programskoj paradigmi
- različite razine testiranja
    - jedinično testiranje: JUnit
        - testiramo klasu po klasu zasebno
    - testiranje većih cjelina: Expect
        - testiramo cjeline unutar sustava, postepeno obuhvaćamo sve više cjelina

        - prema automatiziranom testiranju: JavaPathfinder
            - potpuno automatsko testiranje ne postoji, ali se može donekle automatizirati
- podjele prema načinu testiranja
    - statičko (strukturalno) i funkcionalno
    - bijela i crna kutija
    - prema obuhvaćenoj cjelini testiranja: komponenta, blok jedinica – regresijsko
    - prema automatiziranosti: poluautomatsko ili ručno
    - iz kôda ili prema (zadanoj) specifikaciji (conformance)
    - mutant, simbolično (concolic), kombinatorno
    - kombinacije spomenutoga - prema zadanoj odabranoj strategiji
    - testiranje kao dio razvoja: TDD
- crna kutija
    - nije potrebno poznavati unutarnju strukturu programa (pp)
    - gledamo da li "crna kutija" obavlja funkciju
    - potrebno je poznavati ŠTO radi program (pp)
- bijela kutija
    - potrebno je poznavati unutarnju strukturu programa (pp)
    - paradigma: program i pp su objektno orijentirani
    - xUnit testiranje je tipičan pristup, u našem slučaju JUnit
    - ostale mogućnosti: statička analiza–testiranja odnosno invarijante odnosno “runtime” analiza
- JUnit, XUT (device under test) i programska potpora
    - JUnit–Java Unit je program (programsko okružje) za testiranje
    - Unit predstavlja odabranu jedinicu–komponentu koju testiramo. postoji xUnit podrška za većinu (objektnih) programskih jezika
        - okružje dobivamo dodavanjem klasa u XUT
        - testiranje započinje definiranjem test–slučajeva
        - test–slučajevi se prevode (javac + JUnit.jar)
        - dobivene klase izvodimo sa JUnit Test Runner
        - očekivani rezultat mora biti jednak dobivenom rezultatu
    - test slučaj – (eng. “Test–case”) opisuje pojedinačne testove i sadržava:
        - objekt/funkcionalnost koju testiramo (dio programskog koda)
        - ulazne parametre – podatke (eng. “test–data”)
        - izlazne parametre – podatke
        - očekivane izlazne parametre – podatke
        - može sadržavati jednu ili više assert junit naredbi
    - “test suite” ili test ili test sekvenca (TESTSuite)
        - skup test slučajeva i sadržava jedan ili više pojedinačnih test-slučajeva
        - testiraju funkcionalnost prema modelu M ili specifikaciji: mora se znati što implementira pp
        - jedan ili više testova ili test-sekvenci su sastavni dio dokumentacije koja se predaje testerima ili služi za održavanje pp
    - Odnos programske potpore pp i testova može se prikazati slijedećom formulom: $M \models \varphi$
        - model M je XUT; model M je isto što i programski kod. Kasnije se uvode formalni modeli prikaza programske potpore.

        - φ je skup logičkih varijabli (φ_i) koje opisuju istinitost testova: ako je test *i* OK tada je φ_i istinit
        - JUnit naredbe assert* nazivamo i invarijantama programskog sustava
        - Kažemo: model M zadovoljava formulu φi
    - JUnit assert* naredbe
        - assertEquals
        - assertArrayEquals
        - assertTrue i assertFalse
        - assertNull i assertNotNull
        - assertSame i assertNotSame
        - Osim @Test postoje još i sljedeće direktive za izvođenje testova: @BeforeClass, @Before, @After, @AfterClass
    - "Lažni" ili "Mock" objekti
        - "Mock" objekt - lažni, oponašajući, imitirajući objekt
        - nadomjestimo svaku neželjenu popratnu "smetnju" za testiranje "mock objektom"
- testiranjem upravljano programiranje ili eng. TDD – Test Driven Development
    - napisati dovoljno instrukcija zajedno sa (xUnit) testovima da tijekom prvog izvođenja test padne
    - ispraviti što treba pa testirati dok svi test slučajevi ne prođu
- "Klasični" pristup razvoja pp
    - specifikacija (opisna ili preko SDL ili UML jezika)
    - implementacija ("kodiranje" u ciljnom jeziku)
    - testiranje (tek sada se definiraju testovi)
    - modifikacije (temeljem rezultata testiranja)
    - (održavanje pp)
- "TDD" pristup razvoja pp
    - napisati test slučajeve
    - napisati jezgru (skeleton) koda take da se pp može pokrenuti
    - test padne (nema koda za rješenje)
    - dopuniti kod, testirati
    - ponavljati dok svi testovi ne postanu uspješni
- Paradigme razvoja pp
    - agilno programiranje
    - ekstremno programiranje
    - programiranje u parovima
    - programiranje sa automatima („switch-case" programiranje)

<u>Testiranje zasnovano na modelima</u>

- Sustav koji testiramo je programska potpora kojeg promatramo kao skup klasa u Java programu povezanih dijagramom klasa u pripadnom UML dijagramu
- Pristupi: "bottom–up" i "top–down"
    - "od jednostavnijeg–prema–složenijem" (eng. bottom-up) – prihvatljiviji tijekom razvoja programske potpore

    - "od vrha prema dnu" ili od složenijeg prema jednostavnijem (eng. top-down) – prihvatljiviji tijekom održavanja i naknadnih modifikacija
- Razine testiranja
    - jedinično testiranje: testiraj svaku metodu unutar jedne klase zasebno kroz niz ("razuman broj") testova
    - testiranje klase: iduća jedinica složenosti kod OO programa nakon metode je klasa. Kod ne–objektnih programa testiramo komponentu, blok, modul
    - integracijsko testiranje: svaka od klasa je u interakciji s nekim klasama: potrebno je par po par, skup po skup takvih interakcija testirati. Kod neobjektnih programa testiramo interakcije komponenti, blokova, modula
    - testiranje sustava: testiranje funkcionalnosti sustava, najčešće prema zahtjevima krajnjeg korisnika
    - test prihvatljivosti krajnjeg korisnika naručioca koji mora odgovoriti na pitanje: Da li je programska potpora prihvatljiva za krajnjeg korisnika? Toj grupi pripada tzv. α i β testiranje.
        - α testiranje (alfa-test) programska potpora u α statusu "najranije upotrebljive cjeline" koju se može isprobati
        - β testiranje – odabrani skup potencijalnih budućih korisnika ´ testira novu reviziju programske potpore
- Greška, pogreška i neispravnost
    - pogreška: (eng. software fault): statički defekt u programskoj potpori (u žargonu cesto se naziva "bug")
    - greška: (eng. software error) nekorektno unutarnje stanje programske potpore koje je manifestacija postojanja pogreške (tj. postojanja "buga" ili software fault)
    - neispravnost: (eng. software failure) nekorektno ponašanje programske potpore prema "vani" odnosno prema zahtjevima (requirements), odnosno prema očekivanom ponašanju
- Testiranje nije isto što i "debugging"
    - testiranje: pronalaženje ulaza koji dovode do pogreške (fault)
    - debuggiranje: proces pronalaženja pogreške ("bugg" odnosno "fault")
- Upravljivost programske potpore
    - kolika je složenost određivanja razumnog broja ulaza kao vrijednosti varijabli, operacije ili ponašanje programa
    - upravljivost programa koji isključivo primaju ulaze preko tastature je veća nego programa koji primaju ulaze direktno preko senzora, iz sklopovske potpore ili iz raspodijeljenih procesa
- Opservabilnost programske potpore
    - kolika je složenost određivanja izlaza iz programske potpore kao vrijednosti varijable, kao izlazi u sklopovsku potporu, ili prema drugim procesima, nitima
    - opservabilnost je niska kod programske potpore koja direktno komunicira s sklopovskom potporom, senzorima, bazama podataka, a visoka kod programa koji imaju direktni terminalski izlaz
- Raspodijeljenost i upravljivost: kod raspodijeljenih sustava
    - Raspodjeljeni sustavi (eng. distributed systems) i (eng. concurrent systems) konkurentni sustavi (takvi su svi sustavi programske potpore u telekomunikacijama i sustavima kritičnim po život i zdravlje ljudi) imaju lošu upravljivost i opservabilnost.

Za njihovo testiranje koriste se dodatni alate i metode za testiranje kao npr. provjera modela i dokazivanje teorema

- testiranje samo pokazuje prisutnost grešaka, testiranje NIKAKO i NIKADA ne pokazuje odsutnost grešaka
- Vrste (strategije) testiranja
    - "crna kutija" (black box) i "bijela kutija" (white box)
    - test usklađenosti eng. conformance testing: testiranje prema specifikaciji ili zahtjevima (iskazani kroz npr. UML dijagrame)
    - prema razinama testiranja: (Bottom–up i top–down)
    - regresijsko testiranje
    - α i β testiranje
    - kombinacije gornjeg
- Crna i bijela kutija kod testiranja pp sustava
    - (black-box testing): definiranje testova iz specifikacije, zahtjeva i ostalih opisa sustava
    - (white-box testing): definiranje testova iz programskog kôda, uz posebnu pažnju na grananja, petlje i ostale kontrolne strukture
- regresijsko testiranje
    - svaka klasa, komponenta, modul . . . tijekom testiranja/životnog vijeka doživljava niz modifikacija (zbog pogrešaka ili zbog dodatnih zahtijeva na funkcionalnost)
    - modifikacije mogu promijeniti (već testirano) ponašanje
    - regresijsko testiranje mora ustanoviti uzroke takvog ponašanja programske potpore
    - ispravke/modifikacije ne smiju poremetiti funkcionalnost ostatka sustava
- Modeli programske potpore
    - Grafovi
    - logički izrazi npr.: (not X or not Y) and A and B
    - karakterizacija ulazne domene: granice varijabli npr. A : 0, 1, > 1; B : 600, 700, 800
    - sintaksne strukture: if (x > y) z = x − y; else z = 2 ∗ x
- Graf može predstavljati:
    - grananja zbog kontrolnih naredbi
    - pozive metoda
    - razmjenu poruka
    - prijelaze i stanja prema UML statechart dijagramu
- Grafovi i testiranje
    - graf je G(V,E) gdje je V skup čvorova a E skup grana
    - program prikazujemo kao usmjereni graf
    - grane, putevi usmjerenom grafu predstavljaju sekvence instrukcija između kontrolnih točki, odnosno instrukcije, pozive metoda ili razmjenu poruka
    - kružići ili čvorovi predstavljaju kontrolne točke (grananja) u programu i sadržavaju vrijednosti svih varijabli
    - graf uvodimo kao reprezentaciju, model, apstrakciju programa
    - preslikavanje grafa u program nije jednoznačno i ovisi jedino o tome što želimo testirati
    - komunikacijske programe opisujemo sa 2 ili više grafova povezanih komunikacijskim kanalima
- Prekrivanja u grafu
    - prekriti sve instrukcije ili izvesti svaku instrukciju barem jednom

        - prekriti sve puteve u grafu
        - prekriti sve čvorove u grafu
        - prekriti sve grane u programu
        - izvesti svaku metodu barem jednom
        - prekriti sve uvjete (naredbe if, while, for . . .)
        - kombinacije navedenog
- Pomagala za testiranje
    - specijalizirani programski alati (poput uvedenog JUnit)
    - razni pomoćni alati (spomenut ćemo Expect)
    - specijalizirani jezici za testiranje (TTCN-3 Testing and Test Control Notation ver.3)
    - alati za provjeru modela (eng. model checkers) spomenuti će se JavaPathfinder
- Expect
    - Expect je programsko pomagalo za automatizaciju interaktivnih programa
    - Expect je skriptni jezik koji "razgovara" sa programom
    - ne podržava grafička sučelja (GUI) nego isključivo konzole (CLI)
    - osnovna primjena: skratiti dugo i zamorno ponavljanje istih sekvenca komandi
    - osim osnovne primjene može se koristiti i u testiranju
    - u izvornom obliku Expect je razvijen kao TclTk aplikacija
    - Svaki Expect program (ili skripta ili sesija) se sastoji od:
        - inicijalizacije: (set)
        - send: pošalji nalog
        - expect: "očekuje" očekivanu vrijednost
        - ponavljaj (2) – (3) ili
        - END
- TTCN-3 – jezik za testiranje
    - standardizirani jezik za opis i definiciju testova, test slucajeva i test podataka
    - uveden za testiranje protokola a podržavaju ga i ITU–T, ETSI i proizvođači pp
    - testovi su definirani kroz strukturni i funkcionalni dio
    - podatke definira preko standardizirane ASN.1 notacije (“weak”–“strong” types)
    - strukturni dio
        - TRI sučelje prema XUT odnosno i TCI (Test Control Interface) sučelje prema testeru
        - komponente – moduli koji sadržavaju testove se nalaze u (TE - TextExecutable bloku)
        - blokovi prema XUT (SA – System-Adapter i PA – Platform-Adapter) odnosno prema testeru (TM – Test-Management i TL – Test-Logs)
    - funkcijski dio
        - definira sve podatke na rigidan (formalan) način (jezik ASN.1 za definiciju podataka)
        - definira test slučajeve („test-cases“)
        - definira testove i test sekvence
        - definira specifične vremenske zahtjeve kao i zahtjeve ovisne o platformi
    - TTCN-3 je najbolje koristiti:
        - kod testiranja reaktivne pp kakvu nalazimo u upravljačkom sloju telekomunikacijskih sustava
        - kada je pp dio standarda koje srećemo kod protokola

- TTCN-3 i Expect
    - TTCN-3 i Expect definiraju procedure – sekvence tako da “razgovaraju” odnosno međusobno izmjenjuju poruke
    - TTCN-3 ima bogatiji jezik i opisuje sve na rigidan (formalan) način
    - TTCN-3 je složen za upotrebu i treba dosta vremena za savladavanje njegovih mogućnosti
- Blok TE TestExecutable sadržava:
    - ulazne podatke za testove (“test–data”)
    - test slučajeve (“test–cases”)
    - test sekvence (“test–sequences”)
- TCI (Test Control Interface) sučelje
    - sučelje za upravljanje testovima nalazi se između testera i TE bloka
- TRI (Test Runtime Interface) sučelje
    - sučelje za izvođenje testova nalazi se između XUT i TE bloka
- TM – Test–Management i TL – Test–Logs
    - blokovi kojima tester “upravlja” testiranjem
    - TM – služi za pokretanje testova (“test suite”)
    - TL – prikuplja izvješca (“ ´ logs”) o testiranju
- SA – System–Adapter i PA – Platform–Adapter
    - blokovi preko kojih XUT dobiva “naloge” od TTCN-3 upravljačkog dijela
    - SA – pretvara testove u oblik pogodan za izvođenje na XUT
    - PA – vremenske kontrole i dodatne vanjske funkcije specifične za platformu na kojoj se nalazi XUT
- Izvođenje pojedinog “testa”, pozivaju se slijedeći blokovi:
    - TM – pokreni test
    - TE – pozovi “test” (test–suite)
    - SA – prilagodi pojedini test i pozove XUT
    - XUT – izvede test (testove) i vrati rezultate za provjeru (“očekivani” = “dobiveni” rezultat)
    - TM – pokreće slijedeći test a TL čuva izvješće o testiranju
- MBT (Model Based Testing)i priprema testiranja
    - specifikacija pp služi kao model M koji opisuje ponašanje pp.
    - ponašanje pp definiraju sekvence razmjena poruka u pp (ne zaboravimo da je pp reaktivni sustav)
    - XUT predstavlja implementaciju
    - Ako se sekvence modela (specifikacije) i implementacije (XUT) podudaraju (očekivana sekvenca = dobivenoj) – test je OK

- JavaPathFinder
    - razvijen za otkrivanje grešaka, pogrešaka i neispravnosti u Java programima
    - namijenjen za verifikaciju, odnosno testiranje raspodijeljenih sustava (eng. distributed systems)
    - konkurentne Java programe, odnosno Java programe koji intenzivno izmjenjuju informacije sa okolinom teško je efikasno testirati
    - u pozadini JavaPathFinder nalazi se provjera modela (SPIN model checker)
    - Verifikacija Java programa odvija se u sljedećim koracima:
        - dodavanje posebnih klasa i assert() naredbi

- prevodenje java datoteka iz java u class oblik
- pozivanje class fileova sa JPF (JavaPathFinder program umjesto java)
- potrebno je dodati opcije za JPF

Logika

- Logika dolazi od grčke riječi λογικοζ (logikos), značenje riječi logika: misao, ideja, logos
- Logika na osnovu argumenata donosi zaključke, bilo preko formalnih ili neformalnih sustava zaključivanja
- za nas je važan dio logike koji promatra objekte koji su "nama od interesa", ti objekti su:
    - u najširem smislu jezik i informacija
    - u malo užem smislu programska oprema tijekom cijelog životnog ciklusa
    - u najužem smislu pojedinačna instrukcija, skup instrukcija, komponenta, blok .... program
- algebra čini formalni sustav kojim opisujemo ponašanje, odnose među objektima promatranja
- vrste algebri:
    - linearna
    - procesna
    - komunikacijske algebre
    - algebre utemeljene na logici: (npr. Booleova algebra)
- Neka je zadan skup A: A = {a1 ,a2 ,...,ai ,....an}, Skup operacija Ω nad skupom A: Ω = {ω1 , ω2, ..., ωi, ... ωn}, skup aksioma, zakona sa A i Ω (komutacija, asocijacija,distribucija)
    - A i Ω zajedno čine zajedno čine algebru
- želimo primijeniti logiku tako da skup A={0,1} proširimo na objekte od interesa: tj. na instrukcije, programe i programske komponente . . .
- Floyd-Hoare logika (uređena trojka) i pravila:
    - {P} C {Q}
    - P,Q su tvrdnje (eng. assert)
    - C je komanda (instrukcija)
    - ako je tvrdnja P točna prije izvršenja C onda je tvrdnja Q točna ili je program u blokadi
- logika u C, C++, Javi, …
    - logički izrazi u naredbama imperativnih programskih jezika:
        - if (uvjet)
        - switch (uvjet)
        - while (uvjet)
        - for (uvjet)
        - operatori nad bitovima (&,|,^)
        - logički operatori (&&,||)
    - uvjeti definiraju tijek izvođenja instrukcija (eng. control-flow)
- Podjela logike
    - neformalna logika - proučava argumente iskazane običnim (govornim) jezikom
    - filozofska logika - formalni opis prirodnog jezika
    - matematička logika – logika sudova

    - kompjuterska logika: primijenjena logika u telekomunikacijama, informatici, računarstvu
- Vrste logike
    - propozicijska, predikatna
    - logika drugog reda
    - logike višeg reda
    - modalna, temporalna (LTL i CTL)
    - ostale: {descriptivna, fuzzy, kontinuirana} logika
- Boolova algebra
    - A={0,1}
    - Ω = {I,ILI,neg,komplement,...}
    - zakoni/aksiomi:
        - asocijativnost
        - komutativnost
        - apsorcija
        - distributivnost
        - komplement
    - pruža temelj i motivaciju za proširenje na programe
    - prikazi
        - shema-simbol: nule i jedinice kao naponske vrijednosti
        - tablica: opisuje sve moguće kombinacije događaja na ulazi i izlazu
        - logička formula: kraće zapisana tablica
    - tablicu možemo prikazati i kao graf koji opisuje ponašanje odnosno definira dinamički prikaz procesa
- Automat (FSM)
    - konačni diskretni automat FSM=(S,s,A,T,L)
        - S – stanje
        - s – početno stanje
        - A – alfabet
        - L – labele (<instrukcije>*)
        - T: S × A → S
        - Striktnija definicija mora još uključiti: definiciju ulaza i izlaza i definiciju unutarnjih prijelaza
- Propozicijska logika
    - propozicijska logika je algebarski sustav: propL=(A,Ω,Z,I)
        - A – simboli ili propozicijske varijable
        - Ω – logičke poveznice ili operatori
        - Z – pravila zaključivanja (Reductio ad absurdum, eliminacija konjukcije, dvostruka negativna eliminacija, modus ponens. . .)
        - I – aksiomi (inicijalne ili početne točke)
    - propozicijsku logiku (i ostale sustave logike) sistematski izučava matematička logika
    - pravila zaključivanja se mogu implementirati u programske jezike
    - u programskoj praksi često se koristi modus ponens („rezoniranje unaprijed“) kao mehanizam zaključivanja (eng. inference)
        - modus ponens ima dvije premise:
            - if-then (p→q) znači da p implicira q

            - p je istinit
            - zaključak: q je istinit
    - predikat
        - p i q iz prethodnog primjera su sudovi
        - p ≡ Danas je srijeda, ako umjesto srijede uvedemo varijablu $dan: p ≡ Danas je $dan dobili smo predikat p za unifikaciju
        - svaki predikat nešto označava
        - postoji logički izraz za koji je predikat istinit ili lažan
        - Algebra predikata sastoji se od:
            - skupa argumenata
            - skupa varijabli
            - skupa predikata
            - skupa poveznica
            - kvantifikatori: exist (∃) i forall (∀)
- Temporalna logika
    - Modalna logika uvodi modalitete: moguće, vjerojatno i nužno kao dodatne operatore uz standardne I, ILI, negacija ...
    - Temporalna logika je poseban slučaj modalne logike koja modalne operatore interpretira u vremenu
    - Temporalna logika uvodi modalitete kao temporalne operatore: slijedeće, uvijek, sve dok, eventualno
    - Temporalna logika omogućava verifikaciju i specifikaciju programske opreme
    - Temporalna logika i verifikacija
        - Nužne pretpostavke:
            - provjera modela: postavljamo tvrdnje (predikate) koje koje su argumenti formulama temporalne logike
            - modeliranje: program (blok, modul) mora biti apstrahiran - najčešće je to struktura koja opisuje ponašanje programa (graf dostupnosti)
        - Verifikacija primjenom temporalne logike provjerava ispravnost programa za zadane tvrdnje i za sve kombinacije ulaznih podataka

<u>Logika i programiranje</u>

- Ulaz
    - specifikacija
        - specifikacijski jezik (UML, SDL i MSC)
        - dokumentacija
        - dogovori i suradnja unutar tima (tzv. neformalna ili narativna specifikacija)
    - Specifikacija mora biti razumljiva svakom članu tima, razumljivost znači jednoznačnost: uvijek ista funkcionalnost iz iste specifikacija
- Rezultat
    - Rezultat je programski proizvod (ili programska potpora) (PP)
    - Program (“file sa instrukcijama”) nije programski proizvod
- Automatizacija
    - Formalne metode – područje koje prati razvoj sustava a tako i razvoj programske potpore primjenom rigidnog, strogog matematičkog aparata.

    - formalne metoda se mogu primijeniti tijekom svake faze razvoja u “modelu vodopada”
- Modeliranje i verifikacija
    - Nedostaci verifikacije provjerom modela:
        - eksplozija stanja – kombinatorna eksplozija
        - neprecizno modeliranje
        - nedostatak alata u inženjerskoj praksi
- Logički program (koristimo semantiku sličnu PROLOGu), PROLOG - PROgramming in LOGic
    - Logički program nastoji pronaći vrijednost varijabli takve da vrijedi gornja implikacija
    - Logički program unificira varijable među predikatima
    - Logički program ima ugrađen algoritam koji primjenjuje “bactracking ” postupak kojim nalazi rješenje
    - “backtracking” - pretpostavi rješenja pa postupkom dedukcije dolazi do konačnog rješenja
- SAT programiranje u logici
    - SAT dolazi od engleske riječi satisfiability
    - SAT znači da sustav jednadžbi iskazan kroz CNF ili DNF logičke formule mora biti istinit
    - Problem iskazujemo (opisujemo ili modeliramo) pomoću sustava logičkih formula
    - Formule interpretira odgovarajući program – interpreter
    - primjena: problemi u grafovima, planiranje

Semantika

- semantika nastoji pomoću matematičkog aparata opisati smisao programa
- semantika mora jednoznačno definirati rezultat odnosno dati „smisao“ instrukciji
- semantiku primjenjujemo na različite dijelove programske potpore tijekom cijelog života programske potpore:
    - na zahtjeve i specifikaciju
    - na programske module, blokove, podsustave
    - na instrukcije
- Vrste semantike
    - Aksiomatska
        - daje značenje frazama u jeziku iskazujući ih preko logičkih aksioma (npr. Floyd-Hoare logika)
    - Denotacijska
        - frazu u jeziku preslikava u opis ili denotaciju
        - Denotacija je najčešće pseudokod ili matematički formalizam
        - Denotacijsku semantiku možemo okvirno opisati kao prevođenje fraze iz polaznog jezika u matematički formalizam
    - Operacijska
        - definira apstraktnu mašinu (npr. konačni diskretni automat) i značenje fraze opisuje kroz prijelaze u apstraktnoj mašini
- umjesto formalnog pristupa navodimo neke od smjernica o tome što sematika je i što semantika nije (prema Harel-Rumpe)
    - Semantika je izvršna

    - semantika nema ništa zajedničkog sa mogućnošću izvođenja programa
    - semantika nije izvršna specifikacija programa (iako su ponekad vrlo slične na prvi pogled)
- Semantika opisuje ponašanje
    - semantika mora moći opisati svaki jezik i svaku paradigmu
    - ponašanje, struktura, arhitektura jezika ili sustava može ali ne mora imati veze sa ponašanjem
    - ako npr. jezik opisuje strukturu ima svoju semantiku, to nije jezik za opis ponašanja nego strukture
    - semantiku i ponašanje nije dobro izjednačiti: ponašanje nije isto što i značenje (smisao)
- Semantika opisuje ponašanje cijelog sustava
    - postoji semantika sustava
    - sustav je implementiran u jednom ili više jezika, svaki od njih ima svoju semantiku koja nije isto što i semantika sustava
    - dakle za neku cjelinu poput sustava postoji više razina semantike koja ga opisuje
- Semantika opisuje ponašanje individualnih instrukcija
    - možemo govoriti o semantici instrukcije, fraze, sustava, aplikacije…
    - semantika pruža dublji uvid u sustav i prelazi okvire semantike pojedinačne instrukcije
- Semantika znači opisati program matematičkim formalizmom
    - matematički formalizam ne predstavlja istovremeno semantiku programa
    - upotreba matematičkih simbola ne garantira formalnu definiciju semantike programa – matematički formalizam je sredstvo koje pridjeljuje jednoznačno značenje-smisao programu