INŽENJERSTVO ZAHTJEVA

- klasifikacija prema razini detalja:
  - ⇒ korisnički zahtjevi - visoka razina apstrakcije; prirodni jezik, grafički dijagrami
  - ⇒ zahtjevi sustava - detaljna specifikacija funkcija, usluga i ograničenja
    - izražavanje zahtjeva:
      - ⇒ strukturirani prirodni jezik - izražajnost prirodnog jezika, ali uz nametnutu uniformnost; ograničena terminologija
      - ⇒ jezik za opis oblikovanja - poput programskog jezika, ali s više apstraktnih obilježja
      - ⇒ grafička notacija (UML) - grafički jezik proširen tekstom
      - ⇒ matematička specifikacija - najstrože definirana specifikacija; korisnici je ne vole jer je ne razumiju
    - tri tipa sučelja:
      - ⇒ proceduralno (API)
      - ⇒ strukture podataka koje se izmjenjuju
      - ⇒ predstavljanje podataka
    - struktura dokumenta zahtjeva:
      - ⇒ **P**redgovor
      - ⇒ **U**vod
      - ⇒ **P**ojmovnik
      - ⇒ **D**efinicija zahtjeva korisnika
      - ⇒ **A**rhitektura sustava
      - ⇒ **S**pecifikacija zahtjeva sustava
      - ⇒ **M**odel sustava
      - ⇒ **R**azvoj sustava
      - ⇒ **P**rilozi
      - ⇒ **I**ndeks
  - ⇒ specifikacija programske potpore - najdetaljniji opis, objedinjuje korisničke i zahtjeve sustava
- klasifikacija obzirom na sadržaj (korisnički i zahtjevi sustava):
  - ⇒ funkcionalni zahtjevi - usluge koje sustav mora pružati i kako se mora ponašati u određenim situacijama
  - ⇒ nefunkcionalni zahtjevi - ograničenja u uslugama i funkcijama
    - zahtjevi programskog produkta (npr. specifikacija ponašanja)
    - organizacijski (npr. standardi)
    - vanjski (npr. legislativni)
  - ⇒ zahtjevi domene primjene - novi funkcionalni zahtjevi ili ograničenja na postojeće
    - problemi: razumljivost (programeri traže detaljan opis zahtjeva) vs. implicitnost (specijalisti domene podrazumijevaju neke zahtjeve)
- procesi - dva uobičajena modela:
  - ⇒ spiralni (specifikacija - S, validacija - V, izlučivanje - I) - iteracije; kao rezultat se dobije dokumentacija zahtjeva
    - poslovni zahtjevi (S) → izlučivanje korisničkih zahtjeva (I) → korisnički zahtjevi (S) → studija izvedivosti (V) → izlučivanje zahtjeva sustava (I) → zahtjevi sustava i modeliranje (S) → izrada prototipa (V) → revizije (V)
  - ⇒ klasični
- generičke aktivnosti - krajnji rezultat je dokumentacija zahtjeva
  - ⇒ studija izvedivosti - provjerava doprinose sustava te mogućnosti ostvarenja i integracije
  - ⇒ izlučivanje zahtjeva te njihova analiza i specifikacija - razjašnjava domenu primjene, definira usluge sustava i određuje ograničenja u radu sustava
    - metode:
      - ⇒ intervju: zatvoreni (unaprijed definirana pitanja) ili otvoreni
      - ⇒ scenarij
      - ⇒ obrasci uporabe
      - ⇒ dinamičke interakcije

- spiralni model - 4 osnovne aktivnosti koje idu u iteracijama:
  - ⇒ izlučivanje / otkrivanje zahtjeva - interakcija s dionicima
  - ⇒ klasifikacija i organizacija zahtjeva - grupiranje srodnih zahtjeva u *cluster*-e
  - ⇒ ustanovljavanje prioriteta i pregovaranje - razrješavanje konflikata
  - ⇒ dokumentiranje zahtjeva
- pogledi (*viewpoints*): pogledi interakcije, indirektni pogledi, pogledi domene primjene

⇒ validacija zahtjeva

⇒ upravljanje zahtjevima

PROCESI PROGRAMSKOG INŽENJERSTVA

- model procesa - apstraktna reprezentacija procesa koja predstavlja njegov opis iz određene perspektive
- češće upotrebljavani modeli životnog ciklusa - opis faza od početka projekta do kraja životnog vijeka proizvoda
  - ⇒ ad hoc - *code and fix*
  - ⇒ prototipni
  - ⇒ spiralni - slijedi vodopadni model, ali u iteracijama
    - sektori
      - ⇒ postavljanje ciljeva
      - ⇒ procjena i smjanjivanje rizika - npr. SWOT analiza
      - ⇒ razvoj i validacija - odabir modela; u ranim spiralama koncept, a kasnije specifikacija, oblikovanje, razvoj, testiranje i uporaba
      - ⇒ planiranje - revizija i planiranje slijedeće spiralne faze
  - ⇒ agilni
  - ⇒ generički modeli:
    - ⇒ vodopadni - odvojene faze
      - faze
        - ⇒ analiza zahtjeva i definicije
        - ⇒ razvoj i oblikovanje sustava i programske potpore
        - ⇒ implementacija i testiranje modula
        - ⇒ integracija i testiranje sustava
        - ⇒ uporaba sustava i održavanje
      - pretpostavke
        - ⇒ poznati zahtjevi koji se rijetko mijenjaju
        - ⇒ korisniku nisu potrebna pojašnjenja
        - ⇒ odvojeno oblikovanje, rijetko će dovesti do pogrešaka
        - ⇒ sama tehnologija rješava neke zahtjeve
        - ⇒ sustavi nisu složeni
      - problemi
        - ⇒ teška ugradnja naknadnih promjena
        - ⇒ nefleksibilna podjela projekta
        - ⇒ prikladan ako su zahtjevi dobro razumljivi
        - ⇒ uglavnom se koristi za velike projekte
    - ⇒ inkrementalni / evolucijski - isprepletene faze
      - dva pristupa
        - ⇒ metoda odbacivanja prototipa - grubo definirani zahtjevi, razjašnjavaju se tijekom postupka
        - ⇒ istraživački razvoj i oblikovanje
          - cilj kontinuirani rad s kupcem
          - na početku dobro definirani zahtjevi
        - ⇒ problemi
          - loše strukturirani sustavi
          - potrebne posebne vještine (brz razvoj prototipa)
        - ⇒ primjena
          - mali i srednji interaktivni sustavi
          - dijelovi velikih sustava
          - sustavi s kratkim vijekom trajanja

⇒ unificirani - uporaba modela izvedenih na temelju UML-a i pridruženih aktivnosti

- svojstva
  - ⇒ priznaje utjecaj korisnika
  - ⇒ sugerira evolucijski pristup
  - ⇒ podržava objektnu orijentaciju
  - ⇒ prilagodljiv
- tri perspektive
  - ⇒ dinamička - slijed faza kroz vrijeme
  - ⇒ statička - aktivnosti procesa
  - ⇒ praktična - sugerira aktivnosti kroz iskustvo i dobru praksu
- ***Rational Unified Process*** **(RUP)**
  - ⇒ više iteracija na pojedinim dijelovima programske potpore
  - ⇒ temelji se na obrascima uporabe (scenariji)
  - ⇒ u fokusu arhitektura sustava
  - ⇒ najbolja praksa:
    - iterativan razvoj
    - upravljanje zahtjevima
    - zasnovano na komponentama
    - vizualno modeliranje (UML)
    - kontinuirana verifikacija kvalitete
    - upravljanje promjenama
  - ⇒ elementi:
    - proces - skup koraka
    - disciplina - skup aktivnosti
    - uloga (*tko*)
    - aktivnosti (*kako*)
    - artifakt (rezultat procesa) (*što*)
    - radni tijek (*kada*)
  - ⇒ horizontalna dimenzija - dinamika
    - ciklusi, faze, iteracije
      - ⇒ početak - definira doseg projekta, razvoj modela
      - ⇒ razrada - plan projekta, specifikacija značajki, temelj arhitekture sustava
      - ⇒ izgradnja / oblikovanje
      - ⇒ prijenos produkta korisnicima
    - ključne točke - definiraju pridružene dokumente ili aktivnosti
  - ⇒ vertikalna dimenzija - statički opis procesa: aktivnosti, discipline, uloge, artifakti
  - ⇒ implementacija
    - procjena trenutnog stanja
    - postavljanje / revizija ciljeva
    - identifikacija rizika
    - planiranje procesa implementacije
    - implementacija
    - evaluacija implementacije
  - ⇒ značajke
    - interativno i inkrementalno oblikovanje programske potpore
    - u središtu obrasci uporabe
    - definira i povezuje faze i aktivnosti procesa
    - opis pomoću modela (UML dijagrami)
    - arhitektura sustava - skup svih dijagrama

⇒ iteracije
- sastavni dio velikih projekata (reoblikovanje pojedinih stupnjeva)
- inkrementalan pristup - prioritetne cjeline (dijelovi višeg prioriteta se isporučuju u ranim inkrementima)
- spiralni pristup
  ⇒ postavljanje ciljeva
  ⇒ procjena i smanjivanje rizika
  ⇒ razvoj i validacija
  ⇒ planiranje sljedeće faze

⇒ komponentno usmjeren / zasnovan na komponentama (*component-based software engineering* - CBSE)
- uporaba postojećih komponenti
- stupnjevi
  ⇒ specifikacija i analiza zahtjeva
  ⇒ analiza komponenata
  ⇒ modifikacija zahtjeva
  ⇒ oblikovanje sustava s višestrukom uporabom komponenata
  ⇒ razvoj i integracija

- generičke aktivnosti

⇒ specifikacija - određivanje potrebnih usluga i ograničenja u radu i razvoju

⇒ razvoj i oblikovanje
- oblikovanje - izbor i modeliranje arhitekture
  ⇒ izbor i oblikovanje arhitekture → arhitektura sustava - dokumentirana skupom modela (grafički dijagrami)
  ⇒ apstraktna specifikacija → specifikacija programa
  ⇒ oblikovanje sučelja → specifikacija sučelja
  ⇒ oblikovanje komponenata → specifikacija komponenti
  ⇒ oblikovanje struktura podataka → specifikacija podatkovnih struktura
  ⇒ oblikovanje algoritama → specifikacija algoritama
- implementacija - preslikavanje strukture u izvršni program
  ⇒ programiranje i otklanjanje pogrešaka

⇒ validacija i verifikacija
- validacija - zadovoljava li sustav funkcijske zahtjeve
  ⇒ ispitivanje sustava
  - pretpostavka: specifikacija
  - usporedba stvarnih rezultata s postavljenim standardima
  - ispitivanje komponenti i modula - nezavisno
  - ispitivanje cjelovitog sustava
  - ispitivanje prihvatljivosti - značajke na temelju kojih kupac prihvaća i preuzima sustav
  - vrste
    ⇒ ispitivanje komponenti (*unit testing*)
    ⇒ integracijsko ispitivanje
    ⇒ ispitivanje sustava
    ⇒ test prihvatljivosti
    ⇒ test instalacije
    ⇒ *alpha* test - interno
    ⇒ *beta* test - vanjsko
- verifikacija - zadovoljava li sustav zahtjeve na ispravan način
  ⇒ provjera poželjno zasnovana na formalnim metodama

⇒ evolucija - granica između razvoja i oblikovanja te evolucije i održavanja postaje sve više irelevantna (sve manje potpuno novih sustava)

- računalom podržano programsko inženjerstvo (*Computer Aided Software Engineering*)
  - ⇒ alati za automatizaciju oblikovanja
    - grafički editori za razvoj modela sustava
    - rječnici i zbirke za upravljanje entitetima u oblikovanju
    - grafičko okruženje za oblikovanje i konstrukciju korisničkih sučelja
    - alati za pronalaženje pogrešaka u programu
    - automatizirani prevoditelji koji generiraju nove inačice programa
  - ⇒ klasifikacija alata
    - funkcionalna perspektiva - prema specifičnoj funkciji
      - ⇒ *planning tools*, *editing tools*, *testing tools*, *debugging tools*, *documentation tools*, ...
    - procesna perspektiva - prema aktivnostima koje podupiru u procesu
      - ⇒ *specification*, *design*, *implementation*, *verification and validation*
    - integracijska perspektiva - prema organizaciji u integrirane cjeline
      - ⇒ alati - podupiru individualne zadatke (oblikovanje, uređivanje teksta, ...)
      - ⇒ radne klupe (*workbench*)
        - podupiru pojedine faze procesa (npr. specifikaciju)
        - integracija više alata
      - ⇒ razvojne okoline (*environments*)
        - skupina alata i radnih klupi
        - podupiru cijeli ili značajan dio procesa programskog inženjerstva
  - ⇒ prednosti
    - veća produktivnost
    - bolja dokumentacija
    - veća točnost
    - poboljšana kvaliteta
    - smanjeni troškovi održavanja
    - utjecaj na organizaciju rada
  - ⇒ nedostatci
    - slabo podupire timski rad
    - velika cijena
    - vrijeme učenja
    - potreba za različitim alatima

ARHITEKTURA PROGRAMSKE POTPORE

- struktura sustava koja sadrži elemente, njihova izvana vidljiva obilježja te međusobne odnose
- uloga
  - ⇒ apstrakcija sustava na visokom nivou
  - ⇒ osnovni nositelj kvalitete sustava
  - ⇒ kapitalna investicija, može se ponovno iskoristiti
  - ⇒ osnova za komunikaciju dionika
  - ⇒ prethodi detaljnoj specifikaciji
- razvoj
  - ⇒ programiranje bilo kako
  - ⇒ *programming in the small* - potprogrami, posebno prevođenje dijelova
  - ⇒ *programming in the large* - uvođenje apstrakcije i skrivanje informacija gdje je moguće
  - ⇒ razvojne okoline
  - ⇒ objektno usmjereni obrasci, integrirana razvojna okruženja (IDE)
  - ⇒ UML, *model driven architecture*
- oblikovanje - identificiranje i strukturiranje podsustava te okruženja za upravljanje i komunikaciju između njih
- prednosti
  - ⇒ smanjuje cijenu oblikovanja, razvoja i održavanja produkta
  - ⇒ ponovna uporaba rješenja
  - ⇒ poboljšava razumljivost i kvalitetu produkta
  - ⇒ razjašnjava zahtjeve
  - ⇒ donošenje temeljnih inženjerskih odluka
  - ⇒ rana analiza i uočavanje pogrešaka u oblikovanju
- model - više pogleda u okviru nekog konteksta
  - ⇒ klasifikacija
    - statičan strukturni model (moduli) - dekompozicija sustava
    - dinamički procesni model (*runtime*)
    - alocirani elementi - odnos programske potpore i razvojne / izvršne okoline
- stilovi
  - ⇒ opis pomoću rječnika ili topoloških ograničenja koja moraju zadovoljiti svi članovi stila
  - ⇒ primjeri
    - protok podataka
    - objektno usmjereni
    - repozitorij podataka
    - upravljan događajima
- klasifikacija po dosegu
  - ⇒ koncepcijski - komunikacija s netehničkim dionicima, neformalna specifikacija komponenti
  - ⇒ logički - detaljan nacrt pogodan za razvoj komponenti
  - ⇒ izvršni - za distribuirane i paralelne sustave, pridruživanje procesa fizičkom sustavu
- primjeri pogleda
  - ⇒ logički - ponašanje sustava i dekompozicija
  - ⇒ procesni - opis procesa i njihove komunikacije
  - ⇒ fizički - instalacija i izvršavanje u mrežnom okruženju
  - ⇒ razvojni - opis modula sustava
  - ⇒ podatkovni - tijek informacija u sustavu
  - ⇒ komponente i konektori
    - komponene - osnovna jedinica izračuvanvanja i pohrane podataka
    - konektori - apstrakcija interakcije

- proces izbora i evaluacije
  - ⇒ prostor oblikovanja - skup opcija na raspolaganju
    - donošenje odluka
      - ⇒ potrebna znanja:
        - zahtjevi
        - trenutno oblikovana arhitektura
        - raspoloživa tehnologija
        - *best practice*
      - ⇒ zadaće:
        - postavljanje prioriteta
        - dekompozicija i definiranje svojstava sustava
        - postavljanje sustava u kontekst i integracija
  - ⇒ *top-down design* - oblikuje najvišu strukturu, pa razrađuje detalje
  - ⇒ *bottom-up design* - donošenje odluka o komponentama, pa stvaranje komponenti više razine
  - ⇒ hibridno
- tehnika donošenja dobrih odluka oblikovanja
  - ⇒ uporaba prioriteta i ciljeva
    - pobroji i opiši alternative
    - pobroji prednost i nedostatke svake alternative u odnosu na prioritete i ciljeve
    - odredi sukobe
    - odaberi alternative koje najbolje zadovoljavaju ciljeve
    - prilagodi prioritete za daljnje donošenje odluka
  - ⇒ analiza troškova i koristi
    - procjena troškova - rad programskog inženjera (razvoj + održavanje), cijena razvojne tehnologije, cijena krajnjih korisnika i potpore
    - procjena koristi - ušteda vremena programskog inženjera, povećana prodaja ili uštede
- razvoj modela
  - ⇒ započeti izradom kostura
    - osnovni zahtjevi i oblikovni obrasci
    - neophodne komponente
    - odabrati jedan od uzoraka
  - ⇒ poboljšanje
    - utvrđivanje osnovnih interakcija komponenti i sučelja
    - odluka o raspoređivanju podataka i funkcionalnosti po komponentama
    - utvrđivanje mogućnosti ponovne uporabe postojećih ili izgradnje nove arhitekture

- principi oblikovanja
  1. podijeli i vladaj - odvojeni timovi rade na manjim problemima
     - distribuirani sustavi → podsustavi → paketi → klase
  2. povećaj koheziju
     - velika kohezija = grupiranje međusobno povezanih elemenata
     - klasifikacija
       ⇒ funkcijska - kôd koji obavlja pojedinu operaciju je grupiran
         - prednosti: olakšano razumijevanje, povećan *reusability*, lakša zamjena
       ⇒ razinska - razine formiraju hijerarhiju, viša razina pristupa nižoj, obrat ne vrijedi (API)
       ⇒ komunikacijska - moduli koji pristupaju ili mijenjaju određene podatke su grupirani
       ⇒ sekvencijska - grupiranje procedura u kojima jedna daje ulaz slijedećoj
       ⇒ proceduralna - grupiranje procedura koje se upotrebljavaju jedna nakon druge, slabija od sekvencijske
       ⇒ vremenska - grupiranje operacija koje se obavljaju tijekom iste faze rada (npr. inicijalizacija), slabija od proceduralne
       ⇒ korisnička - povezani *utilities* koji se logički ne mogu smjestiti u ostale grupe
  3. smanji međuovisnost
     - međuovisnost sadržaja - jedna komponenta prikriveno mijenja interne podatke druge
       ⇒ rješenje u OO: enkapsulacija (`private` varijable sa `get` i `set` metodama)
     - opća - pri uporabi globalne varijable
     - upravljačka - kontrola rada druge procedure uporabom zastavice ili naredbe
       ⇒ rješenje u OO: polimorfne operacije i *look-up* tablice
     - međuovisnost u objektnom oblikovanju (*stamp*) - kad je jedna klasa tip argumenta metode (jedna klasa upotrebljava drugu)
       ⇒ rješenje u OO: uporaba sučelja kao tipa argumenta i prijenos jednostavnijih varijabli
     - podatkovna - kada je tip metode argumenta primitiv ili jednostavna klasa
     - povezivanje poziva procedura - kada procedura ili metoda u OO sustavu poziva drugu
     - međuovisnost tipova - kada modul koristi podatkovni tip definiran u nekom drugom modulu
       ⇒ rješenje u OO: deklaracija tipa varijable kao najopćenitije klase ili odgovarajućeg sučelja
     - međuovisnost uključivanjem - `import` (Java) / `include` (C++)
     - vanjska - ovisnost o OS, biblioteci, HW, ...
  4. zadrži višu razinu apstrakcije - oblikovanje sakriva ili odgađa razmatranje detalja
  5. povećaj ponovnu uporabivost - poopćavanje oblikovanja
     - sustav sadrži sučelje(sučelja) za interakciju s dodatnim korisničkim kodom
  6. povećaj uporabu postojećeg - komplementaran 5.
     - aktivna ponovna uporaba komponenti smanjuje trošak i povećava stabilnost sustava
  7. oblikuj za fleksibilnost - predviđaj i pripremi se za buduće promjene (što manja međuovisnost, što veća kohezija, stvaranje apstrakcije), *no hard coding*
  8. planiraj zastaru - planiraj promjene u tehnologiji na način da program može jednostavno biti promijenjen
  9. oblikuj za prenosivost - rad na što većem broju platformi
  10. oblikuj za ispitivanje - oblikuj program za automatsko testiranje
  11. oblikuj konzervativno - obradi sve slučajeve u kojima se komponenta može neprikladno upotrijebiti
  12. oblikuj po ugovoru - sve metode imaju ugovor s pozivateljima (op. a. vjerojatno se misli na sučelje)
      - ugovaratelj ima skup zahtjeva
        ⇒ preduvjeti koje mora ispuniti pozvana metoda kad započinje izvođenje
        ⇒ uvjeti koje pozvana metoda mora osigurati kod završetka izvođenja
        ⇒ invarijante na koje pozvana metoda neće djelovati pri izvođenju

- pisanje dobre dokumentacije
  - ⇒ izričitost, obrada najvažnijih pitanja prije implementacije
  - ⇒ vrednovanje oblikovanja i poboljšanje
  - ⇒ sredstvo komunikacije između dionika
  - ⇒ pisati s gledišta čitatelja
  - ⇒ upotrebljavati standardnu organizaciju
  - ⇒ minimum
    - referentna specifikacija - skup dokumentiranih drivera, pogleda i pomoćne dokumentacije (npr. matrice odluka)
    - pregled za upravu - vizija sustava, motivi, koncepti, poveznice poslovne i tehničke strategije
    - dokumentacija komponenti - pogled na nivou sustava, specifikacija komponente, sučelja te dijagrami suradnje
  - ⇒ struktura
    - svrha - što dokument opisuje
    - opći prioriteti
    - skica sustava
    - temeljna pitanja u oblikovanju - navesti osnovne probleme, razmatrana rješenja, odabrano rješenje i razloge za njegovo odabiranje
    - detalji oblikovanja
  - ⇒ izbjegavati dokumentiranje očitih informacija
  - ⇒ ne pisati detalje koji su dio kôda ili su vidljivi u strukturi kôda
  - ⇒ svojstva dokumentacije
    - dobra - tehnički ispravna, jasno prezentirana
    - ispravna - doseže potrebe i ciljeve ključnih dionika
    - uspješna - upotrebljava se

MODULARIZACIJA I OBJEKTNO USMJERENA ARHITEKTURA

- problemi u oblikovanju
  - ⇒ ranjivost na široko dijeljene varijable
  - ⇒ nenamjerno otkrivanje interne strukture
  - ⇒ prodor odluka o oblikovanju - jedna promjena utječe na mnoge module
  - ⇒ disperzija kôda - teško utvrditi što je sve pogođeno promjenom
  - ⇒ povezane odluke o oblikovanju - povezane definicije raspršuju odluke o oblikovanju
- modularizacija omogućuje jednostavnije upravljanje sustavom, evoluciju sustava i razumijevanje
- povijest
  - ⇒ glavni program i potprogrami - dekompozicija u procesne korake s jednom niti izvođenja
    - hijerarhijska dekompozicija - temelji se na odnosu definicija-uporaba, a pozivi procedura su interakcijski mehanizam
    - jedna nit izvođenja
    - hijerarhijsko rasuđivanje - ispravno izvođenje programa ovisi o ispravnom izvođenju potprograma koji se poziva
    - implicitna struktura podsustava - potprogrami obično skupljeni u funkcijske module
  - ⇒ funkcijski moduli - agregacija procesnih koraka u module
  - ⇒ apstraktni tipovi podataka - zatvaranje podataka i operacija, skrivanje implementacije
    - skup dobro definiranih elemenata i njima pridruženih operacija
    - definirano s matematičkom preciznošću i neovisno o implementaciji
      - ⇒ skup elemenata dohvatljiv preko skupa operacija
      - ⇒ skup operacija čini sučelje
    - proceduralna paradigma
      - ⇒ program organiziran oko pojma procedure
      - ⇒ proceduralna apstrakcija - zadovoljavajuće rješenje za jednostavne podatke
      - ⇒ podatkovna apstrakcija - dodatak, grupiranje dijelova podataka koji opisuju neki entitet i manipulacija s njima kao cjelinom
      - ⇒ sustav je niz operacija nad procedurama
      - ⇒ oblikovanje od najviše funkcije pa razlaganje na više detaljnih
      - ⇒ stanje sustava centralizirano i dijeljeno između različitih funkcija
  - ⇒ objekti i objektno usmjerena arhitektura
    - objektno usmjerena paradigma - promatranje svijeta kroz objekte
      - ⇒ organiziranje proceduralnih apstrakcija u kontekstu podatkovnih
      - ⇒ sva izračunavanja se obavljaju u kontekstu objekta
        - sustav je skup objekata
        - stanje sustava je decentralizirano
        - svaki objekt ima svoje podatke koji opisuju njegovo stanje
      - ⇒ program u radu je skup objekata koji u međusobnoj kolaboraciji obavljaju dani zadatak

- koncepti
  - ⇒ identitet/objekt - svaki je jedinstven i može se referencirati
  - ⇒ razred - apstraktni tip podataka; opisuje skup objekata i sadrži proceduralne apstrakcije koje izvode operacije nad njima
  - ⇒ nasljeđivanje - mehanizam u kojem se značajke podrazreda implicitno nasljeđuju od superrazreda
    - provjera ispravnosti
      - ⇒ „*is a*“ pravilo - podskup *is a* skup; npr. *checking account is an account* - ispravno; *županija je država* - neispravno
      - ⇒ Liskov[1] princip zamjene - ako postoji varijabla čiji tip je superrazred, program se mora korektno izvoditi ako se u nju pohrani instanca tog superrazreda ili instanca bilo kojeg podrazreda
    - prednosti
      - ⇒ mehanizam apstrakcije pogodan za organizaciju
      - ⇒ mehanizam ponovne uporabe u oblikovanju i implementaciji
      - ⇒ organizacija znanja o domeni i sustavu
    - nedostatci
      - ⇒ razredi se ne mogu razumjeti bez poznavanja superrazreda
      - ⇒ nasljeđivanja uočena u fazi analize mogu dati neefikasna rješenja
  - ⇒ polimorfizam - postoji više metoda istog naziva koje različito implementiraju istu apstraktnu operaciju
    - jedna apstraktna operacija se može izvesti na različite načine u različitim razredima
    - više metoda istog naziva, izbor metode koja će se izvesti ovisi o razredu objekta koji se nalazi u varijabli
- terminologija
  - ⇒ objekt
    - bilo što iz stvarnog svijeta čemu se mogu pridružiti obilježja (opis trenutnog stanja) i ponašanje (reakcije)
    - rezultat instanciranja razreda - proces uzimanja predloška i definiranja svih pridruženih atributa i ponašanja; rezervira se memorijski prostor za smještaj atributa i pridruženih metoda
    - stvaranje: `new` + konstruktor razreda; vraća se referenca na objekt
    - svojstva
      - ⇒ stanje
      - ⇒ ponašanje
      - ⇒ jedinstvena identifikacija
    - interakcija: razmjenom poruka
    - instanca - referenca na objekt (dvije instance mogu pokazivati na isti jedinstveni objekt)
      - ⇒ objekt se briše kad su obrisane sve instance koje pokazuju na njega
  - ⇒ razred - predložak za instanciranje objekata; definiran tip podataka
    - specificira naziv, atribute stanja i pridružene metode
    - nivo detalja u specificiranju razreda ovisu o stanju razvojnog procesa
    - imenovanje: velika početna slova, imenice u singularu, ispravna razina generalizacije, paziti da je ime jednoznačno
    - apstraktni razred - sadrži barem jednu operaciju koja nema implementaciju u tom razredu, ne može se instancirati (ključne riječi: `abstract` u Javi, `virtual` u C++u)
      - ⇒ na nekoj nižoj razini hijerarhije mora postojati implementacija te operacije

[1] Barbara Liskov

⇒ varijable
- varijable instanci
  ⇒ mjesto u instanci gdje se smještaju podaci
  ⇒ atribut - jednostavni podaci
  ⇒ pridruživanje - odnosi prema drugim instancama drugih razreda
- različit koncept od objekta - jedan objekt može referencirati više varijabli istovremeno
  ⇒ u Javi dva tipa:
    - *primitive* - evaluira se u vrijednost koju sadrži, sadrži jednostavnu vrijednost
    - *reference* - evaluira se u adresu objekta, slično pokazivaču
- varijable razreda - identificiraju se pomoću ključne riječi `static`
  ⇒ dijele ju sve instance razreda, uvijek postoji samo jedna vrijednost te varijable
  ⇒ ako jedna instanca upiše vrijednost u nju, sve instance razreda vide izmjenu
  ⇒ korisne za konstante te *lookup* tablice i slično
- lokalne varijable
  ⇒ unutar metoda za privremenu pohranu vrijednosti
  ⇒ doseg unutar zagrada metode; uz metodu vezan životni vijek i vidljivost
- parametri

⇒ metoda
- proceduralna apstrakcija koja se koristi za implementaciju ponašanja razreda
- oblikovana za rad na jednom ili više atributa razreda
- poziva se razmjenom poruka
- metode razreda - analogno varijablama razreda; identificiraju se pomoću ključne riječi `static`

⇒ operacija
- proceduralna apstrakcija više razine nego metoda - jedna operacija može biti implementirana s više metoda
- specificira tip ponašanja - neovisna o kôdu koji implementira ponašanje
- vidljivost
  ⇒ `public` - svima dostupna
  ⇒ `protected` - dostupna unutar hijerarhije razreda kojom je deklarirana
  ⇒ `private` - dostupna unutar razreda u kojem je deklarirana
- *overloading* - više metoda istog naziva, ali različitog broja, tipova i mjesta parametara (nije isto što i polimorfizam)
- *overriding* / nadjačavanje - metoda definirana u superrazredu se redefinira u podrazredu
  ⇒ koristi se za restrikciju, proširenje ili optimizaciju
- klasifikacija operacija
  ⇒ osnovne - konstruktori i destruktori
  ⇒ pomoćne - ne mijenjaju stanje objekta
  ⇒ modificirajuće - utječu na promjenu stanja
  ⇒ pretvorbe - stvaraju objekt drugog tipa na osnovu trenutnog stanja
  ⇒ ponavljajuće - obrađuju skupine objekata

⇒ sučelje
- opisuje dio vidljivog ponašanja skupa objekata
- lista apstraktnih operacija

⇒ komponenta

⇒ paket

⇒ podsustav

- osnovni principi
  - ⇒ apstrakcija
    - olakšava savladavanje složenih problema
    - objekt je apstrakcija nečega u realnom svijetu
    - razred je apstrakcija objekta
    - hijerarhija - superrazred je apstrakcija podrazreda
    - operacije - skup metoda
    - atributi i pridruživanja su apstrakcije varijabli instanci
  - ⇒ enkapsulacija - skrivanje informacija unutar razreda
  - ⇒ modularnost - program se može oblikovati samo iz razreda
  - ⇒ hijerarhija - elementi istog hijerarhijskog nivoa moraju biti na istom nivou apstrakcije
- dinamičko povezivanje - u slučaju kad se odluka o izboru konkretne metode donosi za vrijeme izvođenja
  - ⇒ kad je tip varijable superrazred i postoji više polimorfnih metoda koje se mogu izvesti
- konstruktor i destruktor - programski implementacijski detalji
  - ⇒ metode koje se pozivaju pri kreiranju odnosno uništavanju objekta
  - ⇒ konstruktor inicijalizira objekt i njegove varijable; naziv isti kao i naziv razreda
- dobra praksa
  - ⇒ komentari - komentirati ono što nije očigledno; čine 25-50% kôda
  - ⇒ konzistentna organizacija elemenata razreda
  - ⇒ izbjegavaj dupliciranje kôda
  - ⇒ pridržavaj se principa objektnog usmjerenja
  - ⇒ preferiraj nedostupnot informacija - što više private
  - ⇒ ne miješaj kôd korisničkog sučelja s ostalim kôdom u programu - interakciju s korisnicima stavi u posebne razrede
- objektno usmjereni programski jezici: Simula-67; Smalltalk; C++; Java; C#

⇒ komponente i oblikovanje zasnovano na njima - višestruka sučelja; posrednici; binarna kompatibilnost

ISPITIVANJE PROGRAMSKE POTPORE

- definicija
  - ⇒ proces izvođenja programa sa svrhom pronalaženja pogrešaka
  - ⇒ proces analize programskog kôda sa svrhom pronalaska razlike između postojećeg i zahtijevanog stanja, te vrednovanja svojstava programa
- zasniva se na dinamičkoj verifikaciji ponašanja programa u izvođenju na konačnom broju pogodno izabranih ispitnih slučajeva
- ispitni slučaj (test) - jedan ili više ispitnih slučajeva (scenarija)
- tehnike verifikacije programa
  - ⇒ dinamička verifikacija - ispitivanje (testiranje)
  - ⇒ statička verifikacija - ispitivanja strukture; provjere ispravnosti
    - provodi se na specifikaciji zahtjeva, raznim nivoima oblikovanja sustava i programskom kôdu
    - nadzor izvornog kôda - analiza statičkih artefakata s ciljem otkrivanja problema
      - ⇒ recenzija programskog produkta (ne zahtijeva izvođenje)
      - ⇒ provjera usklađenosti sa specifikacijama (ne može provjeravati nefunkcionalna svojstva)
      - ⇒ prolazak (*walkthrough*; češljanje) - neformalan nadzor i inspekcija izvornog kôda ili dokumentacije
      - ⇒ nadzor (inspekcija) - utvrđivanje usklađenosti sa standardima ili zahtjevima; zahtjeva planiranje i raspodjelu zadaća, formalno bilježenje i obradu rezultata
    - analizatori programa
    - formalne metode
  - ⇒ formalna verifikacija - primjena formalnih metoda matematičke logike za dokaz ispravnosti programa
- ciljevi:
  - ⇒ najčešće - pronaći i ispraviti pogreške
  - ⇒ osigurati pouzdanost, ispravnost, otkrivanje pogrešaka
  - ⇒ minimizirati rizike
    - provjeriti sukladnost rada s ostalim komponentama
    - pomoći u donošenju odluke o puštanju u rad/prodaju (zaustaviti prerano puštanje)
    - minimizirati troškove tehničke podrške
    - procijeniti sukladnost specifikacijama i sukladnost s normama (zakonske, tehničke...)
  - ⇒ definirati način sigurne upotrebe
  - ⇒ procjena kvalitete
- kvar - uzročnik kvara (op. a. srsly -.-), može biti prikriven neko vrijeme
  - ⇒ krivi rad pri oblikovanju ili programiranju
  - ⇒ vrste
    - kvar specifikacije sučelja (npr. nepodudaranje zahtjeva i implementacije ili formata poruka klijenta i poslužitelja)
    - kvar u algoritmima (nedostatak inicijalizacija, pogrešna grananja, zanemarivanje rukovanja null vrijednostima)
    - mehanički (dokumentacija nije sukladna stvarnom stanju)
    - aritmetički
    - logički
    - sintaksni
    - memorijski
    - dretveni
    - ...

- pogreška - dio stanja sustava odgovorno za stvaranje zastoja (manifestacija kvara)
  ⇒ uvođenje kvara u programsku potporu
  ⇒ Paretov princip - mali broj pogrešaka dovodi do velikog broja zatajenja
  ⇒ primjeri
    - izazvana gubitkom poruka pri opterećenju
    - izazvana ograničenjima memorije
    - vremenska
    - ...
  ⇒ tipovi: blage, dosadne, uznemiravajuće, ozbiljne, granične, katastrofalne, zarazne
  ⇒ kategorije: funkcijske, sistemske, podatkovne, ...
  ⇒ upravljanje:
    - prevencija - oblikovanje što manje složenosti, sprječavanje nekonzistentnosti, verifikacija za sprječavanje kvarova u algoritmima
    - detekcija - ispitivanje, *debugging*, *monitoring*
    - oporavak - u radu programa; baze podataka; modularna redundancija
- zatajenje - sustav ne zadovoljava specifikacije
  ⇒ vrste
    - ne ispunjava očekivanja zahtjeva
    - ne ispunjava očekivanja korisnika
  ⇒ razlozi
    - nepotpuni, nekonzistentni ili zahtjevi nemogući za implementaciju
    - pogrešna interpretacija zahtjeva
    - kvar u oblikovanju arhitekture ili programa
    - kvar u programskom kôdu
    - dokumentacija nekorektno opisuje ponašanje sustava
- povijest
  ⇒ ispravljanje pogrešaka (*debugging*)
  ⇒ orijentacija na demonstraciju - razdvajanje provjere rada programa i ispravljanja pogrešaka
  ⇒ orijentacija na razaranje - uspješno ispitivanje otkriva zatajenje
  ⇒ orijentacija na evaluaciju - ispitivanje je dio validacije i verifikacije, rano otkrivanje pogrešaka
  ⇒ orijentacija na prevenciju - ispitivanjem se pokušavaju izbjeći pogreške
  ⇒ orijentacija na razvoj programske potpore - *test-driven software development*
- standardi
  ⇒ osiguranje kvalitete - koja ispitivanja je nužno provesti
    - ISO/IEC 29119 - cilj razviti jedan standard koji će pokrivati cijeli životni ciklus ispitivanja programske podrške
    - primjer dokumenata
      ⇒ plan ispitivanja
      ⇒ oblikovanje ispitivanja
      ⇒ ispitni slučajevi
      ⇒ procedure ispitivanja
      ⇒ bilješke ispitivanja
      ⇒ izvješća odstupanja
      ⇒ sažetak izvješća ispitivanja
  ⇒ industrijski - specifikacija razina ispitivanja
  ⇒ ispitivanje programske potpore - specifikacija kako provesti ispitivanje

- tim za ispitivanje
  - ⇒ voditelj projekta
  - ⇒ analitičar - planiranje ispitivanja
  - ⇒ voditelj upravljanja kvalitetom - definira standarde ispitivanja; prati i osigurava sukladnost procesa ispitivanja
  - ⇒ voditelj ispitivanja - analiza zahtjeva ispitivanja i oblikovanje ispitivanja (strategija, ispitni slučajevi...)
  - ⇒ profesionalni ispitivači - priprema i provođenje ispitivanja, pronalaženje pogrešaka
  - ⇒ korisnici
- aktivnosti
  - ⇒ kategorije
    - oblikovanje ispitivanja
    - automatiziranje ispitivanja
    - ispitivanje
    - valorizacija ispitivanja
  - ⇒ planiranje ispitivanja
  - ⇒ oblikovanje ispitnih slučajeva
  - ⇒ izrada ispitnih slučajeva
  - ⇒ provođenje ispitivanja
  - ⇒ analiza rezultata i izvješćivanje
  - ⇒ upravljanje ispitivanjem
  - ⇒ automatizacija ispitivanja
  - ⇒ upravljanje ispitivanjem
- regresijsko ispitivanje - nakon promjene/popravka
  - ⇒ utjecaj: neki ispitni slučajevi postaju nepotrebni; promjene očekivanih rezultata; izrada novih ispitnih slučajeva
  - ⇒ provodi se tijekom integracije i nakon nadogradnji
- plan ispitivanja - struktura
  - ⇒ opis procesa
  - ⇒ sljedivost zahtjeva
  - ⇒ elementi ispitivanja
  - ⇒ vremenik ispitivanja
  - ⇒ procedura bilježenja rezultata
  - ⇒ zahtjevi okoline
  - ⇒ ograničenja
- modeli ispitivanja
  - ⇒ v-model
    - zahtjevi → funkcijska specifikacija → oblikovanje → implementacija → ispitivanje komponente → ispitivanje integracije → ispitivanje sustava → ispitivanje prihvatljivosti
  - ⇒ v-model s ranom pripremom
    - kod izrade zahtjeva, funkcijske specifikacije i oblikovanja se odmah pripremaju i ispitivanja
  - ⇒ w-model
    - zahtjevi → ispitivanje zahtjeva
    - funkcijska specifikacija → ispitivanje specifikacije
    - oblikovanje → ispitivanje oblikovanja
    - implementacija → ispitivanje komponenti
    - integracija → ispitivanje integracije
    - izgradnja sustava → ispitivanje sustava
    - instalacija → ispitivanje prihvatljivosti

- svojstva ispitljivih programa
  ⇒ osmotrivost - jednostavno uočavanje neispravnih rezultata
  ⇒ upravljivost - jednostavnost upravljanja tijekom provođenja ispitivanja
  ⇒ dekompozicija - neovisno ispitivanje modula
  ⇒ jednostavnost - arhitekture, logike programa i kodiranja
  ⇒ stabilnost - promjene tijekom ispitivanja utječu na rezultate provedenih ispitivanja
  ⇒ razumljivost - dobre informacije o strukturama, međuovisnostima i organizacija tehničke dokumentacije
- osnovni koraci
  1. odabir što ispitivati - analiza kompletnosti zahtjeva i oblikovanje ispitivanja
  2. odluka kako ispitivati - statička verifikacija, odabir ispitivanja komponenti, odabir integracijskog ispitivanja
  3. razvoj ispitnih slučajeva
  4. predikcija rezultata za sve ispitne slučajeve
- kada ispitivati? - prije, tijekom i nakon kodiranja
- kako ispitivati? - najbolje koristiti neovisne timove za ispitivanje (program najčešće ne radi kad ga upotrebljava netko drugi)
- terminologija
  ⇒ ispitni podaci (I) - ulazi odabrani za provođenje ispitnog slučaja
  ⇒ očekivani izlaz (O) - zabilježen prije provođenja ispitivanja
  ⇒ ispitni slučaj - uređeni par (I, O)
    - sadrži opis stanja prije ispitivanja funkcije koja se ispituje
  ⇒ stvarni izlaz - rezulatat dobiven ispitivanjem
  ⇒ kriterij prolaza - unaprijed određeni kriterij usporedbe očekivanog i stvarnog izlaza
- pristupi ispitivanju
  ⇒ zasnovano na pokrivenosti - sve naredbe izvršene bar jednom
  ⇒ zasnovano na pogreškama - ispitni slučajevi koji omogućavaju otkrivanje pogrešaka
  ⇒ zasnovano na kvarovima - usmjereno na kvarove graničnih vrijednosti ili maksimalnog broja elemenata
  ⇒ funkcijsko ispitivanje - izgradnja ispitnih slučajeva temeljem specifikacije
  ⇒ strukturno ispitivanje - uzima u obzir strukturu programa
  ⇒ pod pritiskom - naglasak na robusnost u kritičnim uvjetima rada
- potpuno ispitivanje - ispitivanje svih mogućih vrijednosti varijabli, kombinacija ulaza, sekvenci izvođenja programa, HW/SW konfiguracija i načina uporabe programa
- pokrivanje ispitivanja
  ⇒ metrika: postotak programskih elemenata koji su izvedeni, pokrivenost linija kôda, pokrivenost grana, pokrivenost putova
- organizacija ispitivanja
  ⇒ ispitivanje komponenti (*unit test*)
    - traže se pogreške u algoritmima, podacima i sintaksi
    - provodi se u kontekstu specifikacije zahtjeva
    - verificira rad programskih dijelova koje je moguće neovisno zasebno ispitati (pojedinačne funkcije ili metode, klase objekata, složene komponente s definiranim sučeljem)
    - dostupan kôd programa
    - najčešće provodi programer komponente
    - tijekom kodiranja - inkrementalno kodiranje; skraćuje se vrijeme ispitivanja
    - statičko ispitivanje - prolazno, nadzor kôda; koriste se automatizirani alati za provjeru sintaktičkih i semantičkih pogrešaka te otkrivanje odstupanja od standarda
    - dinamičko ispitivanje - funkcijsko i strukturno (crna kutija - bez uvida u kôd)
    - komponenta se izolira u svrhu ispitivanja
      ⇒ upravljački program (*driver*) - upravlja procesom izvođenja komponente (obično upotrebljava postojeće sučelje, no može zahtijevati i stvaranje novih)
      ⇒ prividna/krnja komponenta (*stub*) - simulira pozivanu komponentu (identično sučelje)

- elementi
  - ⇒ sučelje - za ispravno privhaćanje i pružanje informacija
    - parametarsko - podaci i funkcije se prenose pozivima procedure
    - dijeljena memorija - procedure dijele memorijski prostor
    - proceduralno (API)- komponente obuhvaćaju skup procedura koje pozivaju ostali podsustavi
    - sučelje zasnovano na porukama - npr. klijent-poslužitelj
  - ⇒ podaci struktura - osigura integritet podataka
  - ⇒ rubni uvjeti - provjera rada u graničnim slučajevima
  - ⇒ nezavisni putovi - svi putovi kroz kontrolne strukture
  - ⇒ iznimke - provjera ispravne obrade
- objektno orijentirani sustav
  - ⇒ komponenta je za potrebe ispitivanja najčešće razred
  - ⇒ ispitivanje obuhvaća ispitivanje svih operacija, postavljanje i ispitivanje svih atributa objekta te ispitivanje objekta u svim stanjima
  - ⇒ ispitivanje grupa razreda je oblik integracijskog ispitivanja
  - ⇒ poteškoće uzrokuju koncepti poput enkapsulacije, nasljeđivanja, polimorfizma... (ispitivane informacije nisu lokalizirane)
  - ⇒ koraci
    1. ispitni slučaj jedinstveno označiti i eksplicitno povezati s ispitivanim razredom
    2. definirati namjenu ispitnog slučaja
    3. razraditi korake ispitivanja
       i. definirati stanja objekata koja se ispituju
       ii. definirati poruke i operacije koje se ispituju, kao i njihove posljedice
       iii. definirati listu mogućih iznimaka
       iv. definirati stanje okoline pri ispitivanju
  - ⇒ slučajno ispitivanje
    1. indentificirati operacije primjenjive na razred
    2. definirati ograničenja na njihovo korištenje
    3. identificirati minimalni ispitni slučaj (slijed operacija koji definira minimalni životni vijek instanciranog objekta)
    4. generirati niz ispravnih slučajnih ispitnih sekvenci
  - ⇒ ispitivanje particija
    - smanjuje broj ispitnih slučajeva potrebnih za ispitivanje razreda
    - particija stanja - kategorizirati i ispitati operacije temeljem utjecaja na promjenu stanja objekta
    - particije atributa - kategorizirati i ispitati operacije temeljem svojstava atributa
    - particije kategorija - kategorizirati i ispitati operacije na temelju generičkih funkcija koje obavljaju
  - ⇒ ponašajno ispitivanje - ispitni slučajevi moraju pokriti sva stanja objekta
    - pogodna uporaba UML dijagrama stanja
    - automatizacija
  - ⇒ ispitivanje sučelja - objekti i komponente definirani su sučeljima, pogreške nastaju kao posljedica interakcija (cilj otkriti kvarove u sučelju ili pogrešne pretpostavke o sučelju)
    - kvarovi sučelja
      - ⇒ pogrešna uporaba
      - ⇒ nerazumijevanje sučelja - pogrešna pretpostavka specifikacije (npr. binarno pretraživanje na neuređenom nizu)
      - ⇒ vrmenske pogreške - pozvana i pozivajuća komponenta rade različitim brzinama (npr. različita brzina čitanja i pisanja po memoriji)

- naputak
  - ⇒ oblikuj ispitne slučajeve tako da parametri poprime ekstremne vrijednosti
  - ⇒ uvijek ispitaj pokazivače s null vrijednostima
  - ⇒ oblikuj ispitni slučaj proceduralnog sučelja tako da zataji komponenta
  - ⇒ sustave s razmjenom poruka ispitaj na stres
  - ⇒ sustave s dijeljenom memorijom ispitaj s različitim redoslijedom aktiviranja komponenti

⇒ integracijsko ispitivanje

- proces verifikacije interakcije programskih komponenti s dodatnim kôdom za zajednički rad
- cilj osigurati zajednički rad grupe komponenti prema specifikaciji
- složene interakcije - problem lokalizacije pogreški
- pristupi
  - ⇒ veliki prasak - integrirati sve komponente pa onda ispitivati (problem otkriti mjesto pogreške)
  - ⇒ poboljšani veliki prasak - nakon ispitivanja se integriraju komponente (i dalje je problem otkriti mjesto pogreške)
  - ⇒ inkrementalni pristup - integracija i ispitivanje sustava dio po dio (uobičajena praksa, efikasan u otkrivanju mjesta pogreške)
    - odozgo na dolje (*top down*) - razviti kostur sustava pa ga popuniti komponentama
      - ⇒ glavni program je ispitan prvi
      - ⇒ moduli su integrirani istodobno
      - ⇒ glavni naglasak je na ispitivanju sučelja
      - ⇒ nije potrebno izgrađivati upravljačke programe, ali potrebno izgrađivati prividne module
      - ⇒ brzo se dobiva radni prototip
      - ⇒ rano se otkrivaju pogreške u sučeljima, ali se kasno nalaze pogreške u kritičnim modulima na niskim razinama
      - ⇒ modularnost pospješuje ispravljanje pogrešaka
      - ⇒ sporo uključivanje većeg tima
    - odozdo na gore (*bottom up*) - integrirati komponente za najvažnije i najčešće funkcionalnosti, pa dodati ostatak
      - ⇒ omogućeno rano ispitivanje
      - ⇒ moduli mogu biti integrirani u različitim grupama
      - ⇒ glavni naglasak na funkcionalnosti i performansama modula
      - ⇒ nije potrebno izgrađivati prividne module, ali potrebna izgradnja upravljačkih programa
      - ⇒ lagana organizacija provođenja, ali dugotrajan proces u slučaju složenijih hijerarhija
      - ⇒ pogreške u kritičnim modulima nalaze se rano, ali kasno otkrivanje pogreške u sučeljima
    - funkcijska integracija - integriranje komponenti u konzistentne funkcije bez obzira na hijerarhiju (hibridno između *top down* i *bottom up*); najčešće se koristi
- objektna orijentacija
  - ⇒ pogodno je iskoristiti prepoznata grupiranja razreda (npr. međuovisne klase)
  - ⇒ kandidati: razredi grupirani u pakete, razredi u hijerarhijskoj ovisnosti, razredi povezani dijagramima interakcija pridruženim obrascima uporabe
  - ⇒ strategije
    - grupirati prema razinama apstrakcije
    - ispitivanje zasnovano na dretvama - integrira skup razreda obzirom na poticaje ulaza ili događaja
    - ispitivanje zasnovano na uporabi - integrira skup razreda koje zahtijeva obrazac uporabe
    - ispitivanje zasnovano na grupiranju - integrira skup razreda koji su neophodni za ostvarenje suradnje razreda
  - ⇒ tijek ispitivanja
    1. za svaki podrazred koristiti popis operatora za generiranje niza slučajnih ispitnih sekvenci
    2. za svaku poruku koja se generira odrediti pozvani razred i odgovarajući operator u poslužitelju
    3. za svaki operator u pozvanom objektu razreda utvrditi nove poruke koje odašilje
    4. za svaku poruku odrediti sljedeću razinu koja se poziva (izraditi odgovarajuće ispitne slučajeve)

⇒ pristupi
- *bottom up* - najčešći, integracija od krajnjih podrazreda prema hijerarhiji ovisnosti
- primjena obrazaca uporabe - opisuju interakciju razreda
  1. identificirati suradnju u hijerarhiji razreda
  2. odabrati kriterij sekvenci ispitivanja
  3. oblikovati ispitivanje

⇒ ispitivanje sustava
- ispitivanje inačice namijenjene distribuciji korisniku - provjera podudarnosti sustava s funkcijskim zahtjevima
- uobičajeno okruženje funkcijskog ispitivanja: ispitivači ne znaju detalje implementacije (veća vjerojatnost pronalaženja kvarova, nema sukoba interesa razvoja i ispitivanja)
- dvije faze - integracijsko ispitivanje (dozvoljen pristup izvornom kôdu) i ispitivanje gotovog sustava (crna kutija)
- ispitivanje performansi
  ⇒ ispitivanje svojstava sustava kao cjeline u radu
  ⇒ namjene
  - standardno ispitivanje performansi
  - pokazati zadovoljenje performansi pod radnim opterećenjima
  - pokazati proširivost sustava

  ⇒ ispitivanje pod pritiskom: određivanje stabilnosti sustava

⇒ ispitivanje prihvatljivosti
- provjera ponašanja sustava u odnosu na zahtjeve naručitelja, najčešće se provodi zajedno s timom naručitelja
- provodi se kao funkcijsko ispitivanje (crna kutija), obično prije isporuke programa
- cilj pokazati da je sustav spreman za uporabu i zadovoljava zahtjeve

⇒ ispitivanje instalacije - na instalaciji u radnoj okolini, identično ispitivanju sustava prema zahtjevima sklopovske konfiguracije

⇒ alfa ispitivanje - program pokusno upotrebljava skupina korisnika unutar tvrtke, uz prisustvo razvojnog tima

⇒ beta ispitivanje - program pokusno upotrebljavaju vanjski korisnici

- strategije ispitivanja
  ⇒ iscrpno ispitivanje - svi mogući scenariji (samo za ograničene primjere)
  ⇒ slučajno ispitivanje - odabir ispitnih slučajeva temeljem vjerojatnosti (uniformna razdioba, razdioba temeljena na prethodno prikupljenim podacima)
  ⇒ sistematsko ispitivanje - podjela ulaznih podataka u poddomene i odabir ispitnih slučajeva temeljem različitih svojstava (svojstva kôda, specifikacija, rizik, ...)
- tehnike ispitivanja
  ⇒ funkcijsko ispitivanje (crna kutija)
  - nema znanja programskog kôda ili oblikovanja sustava - koncentracija na U/I ponašanje, ispitivanje prema zahtjevima i specifikacijama
  - pretpostavka: za ulazne podatke možemo predvidjeti izlaz
  - cilj: smanjiti broj ispitnih slučajeva ekvivalentnom podjelom ulaza i analizom graničnih vrijednosti
  - oblikovanje ispitnih slučajeva
    ⇒ zasnovano na specifikaciji sustava
    ⇒ podjela vrijednosti ulaznih podataka
    ⇒ podjela ulaznih podataka u klase
    ⇒ odabir ispitnih slučajeva za svaku klasu
    ⇒ analiza graničnih vrijednosti

- ispitivanje particija
  - ⇒ podjela ulaznih podataka i rezultata u klase u kojima je ponašanje članova slično
  - ⇒ pretpostavka: particije/klase su ekvivalentne i program se za sve članove klase ponaša na isti način
  - ⇒ ispitni slučajevi moraju pokrivati sve ekvivalentne particije (posebno granične vrijednosti)
  - ⇒ odabir ekvivalentnih particija
    - podjela ulaznih particija s obzirom na valjanost vrijednosti
      - ⇒ vrijednost ispod intervala
      - ⇒ vrijednost u intervalu
      - ⇒ vrijednost iznad intervala
    - izlazne ekvivalentne podjele - zajedničke karakteristike
  - ⇒ koraci
    1. odredi particije za sve ulazne varijable
    2. za sve particije odaberi vrijednosti ispitivanja
    3. definiraj ispitne slučajeve
    4. odredi očekivane izlaze i provedi ispitivanje
- izražen problem kombinatorne eksplozije ispitnih slučajeva
- nejasno jesu li odabrani ispitni slučajevi dobro oblikovani za otkrivanje pogreške
- jednostavno za uporabu
- brz razvoj ispitnih slučajeva, rade se iz perspektive korisnika
- neovisno o jeziku implementacije

⇒ strukturno ispitivanje (bijela kutija)
- ispitivanje očekivanog ponašanja zasnovano na strukturi programa
- cilj: pokrivanje izvođenja svih mogućih naredbi i uvjeta programa najmanje jednom
- prikaz programa: graf tijeka programa (čvorovi su procesi i programske odluke, a lukovi kontrola tijeka)
- problem postojanje petlji: velik broj putova, putovi koji više puta prolaze kroz petlju su ekvivalentni, velik broj kombinacija
- ispituju se svi neovisni putovi, logički izrazi, petlje, interni podaci
- izvođenje ispitnog slučaja: prolaz kroz određeni put
- selektivno ispitivanje - jer je ispitivanje svih kombinacija najčešće nemoguće
  - ⇒ pokrivanje kôda - pokrivenost izvornog kôda provedenim ispitivanjima
    - ispitivanje temeljnih putova
      - ⇒ temeljni skup: skup putova koji minimalno jednom pokrivaju izvođenje svih naredbi i uvjeta (ne nužno jednoznačan)
      - ⇒ pomoću teorije grafova računamo broj linearno neovisnih putova u temeljnom skupu
        - CV(graf) = brojLukovaUGrafu - brojČvorovaUGrafu + 2*brojPovezanihKomponenti[2]
          - ⇒ ciklomatička složenost - preporuka je da bude manja od 10
          - ⇒ gornja granica broja ispitnih slučajeva koja garantira potpuno pokrivanje svih naredbi programa
      - ⇒ pokrivanje govori u kojoj smo fazi ispitivanja (veći postotak ne dokazuje da u programu nema kvarova)
      - ⇒ za svako pokrivanje mora postojati obrazloženje - koji kvarovi bi ostali neotkriveni ako se neki dio ne pokrije
    - ispitivanje uvjeta
      - ⇒ ispitivanje grana - svaka grana svakog uvjeta ispituje se bar jednom
      - ⇒ ispitivanje domene: za Booleov izraz s n varijabli ima 2^n ispitnih slučaja

[2] potprogrami, prekidne rutine i sl.

- ispitivanje petlji
  - ⇒ fokus na valjanost primjene petlje
  - ⇒ klasifikacija petlji
    - jednostavne - maksimalno n prolaza
      - ⇒ ispitni slučaj:
        - odrediti n
        - preskočiti petlju
        - jedan prolaz
        - dva prolaza
        - m prolaza (m < n)
        - n - 1 prolaz
        - n + 1 prolaz
    - ugniježđene
    - ulančane
    - nestrukturirane - loše programiranje
- ispitivanje protoka podataka
  - ⇒ ispitivanje upravljačkog toka s obzirom na uporabu varijabli
  - ⇒ uočavanje kvarova provjerom uzorka uporabe podataka
  - ⇒ za svaku cjelinu odrediti uporabu varijabli: definiranje (D), uporaba (U), brisanje (K) i nebitno (X)
    - analiza slijeda akcija nad varijablama
- potencijalno beskonačan broj putova za ispitivanje
- često ispituje što je implementirano, a ne što bi trebalo biti
- uvid u izvor pogreške
- povećana ponovna uporaba ispitnih slučajeva
- mogućnost ciljanog ispitivanja kritičnih dijelova
- osjetljivo na promjene kôda

⇒ ispitivanje sive kutije - funkcijsko ispitivanje kombinirano s poznavanjem kôda za potvrdu očekivanih rezultata

- automatizacija ispitivanja
  - ⇒ razvijeni mnogobrojni specijalizirani alati radne okoline
  - ⇒ prednosti
    - brže i jeftinije
    - točnije
    - stabilno ispitivanje kvalitete
    - automatizirano dokumentiranje prijava pogrešaka i izvješćivanje
    - smanjenje ljudskog rada
  - ⇒ za komponente - izgradnja upravljačkih programa, analiza pokrivanja
  - ⇒ za integraciju - sučelja i protokoli
  - ⇒ za sustave - automatizirani su izvođenje ispitivanja i performanse alata
  - ⇒ za prihvatljivost - analiza zahtjeva, implementacije i upotrebljivosti
  - ⇒ pomoćne aktivnosti
    - praćenje kvarova
    - upravljanje procesom ispitivanja

⇒ automatizacija ručnog ispitivanja
- koraci
  ⇒ generiranje ulaznih podataka i očekivanih rezultata
  ⇒ izvođenje ispitivanja
  ⇒ evaluacija
- pojedinačno cijena je 3-30 puta veća od cijene ručnog ispitivanja, ali je cijena ponavljanja 0
- zahtijeva formalizirani ručni ispis ispitivanja
- cijena ovisi o postavljenom dosegu (postupak kodiranja ispitnih slučajeva)
- prednosti
  ⇒ povećana pouzdanost i kvaliteta ispitivanja
  ⇒ kraće vrijeme izvođenja ispitnih slučajeva
  ⇒ automatska analiza rezultata
- nedostatci
  ⇒ cijena
  ⇒ vrijeme pripreme

⇒ razine
- osnovna
  ⇒ metodologija snimanja i reprodukcije - skripte generirane bilježenjem korisnikovih akcija
  ⇒ podatkovno upravljana metodologija - podaci se čitaju iz datoteka, nisu ukodirani u skriptu
- napredna
  ⇒ funkcionalna dekompozicija - male skripte su ispitni moduli i funkcije
  ⇒ radni okviri (*Framework*) - *test*, *process*, *hybrid*

LOGIKA

- formalna verifikacija
- postupak provjere da formalni model izvedenog sustava (izražava se pomoću *finite-state machine*) odgovara formalnoj specifikaciji (izražava se pomoću vremenske logike) s matematičkom izvjesnošću
  ⇒ potrebna znanja iz područja
    - formalne (matematičke) logike
    - modeliranja implementacije strojevima s konačnim brojem stanja
    - izražavanja specifikacije vremenskom logikom (proširenje klasične matematičke logike)
- formalna/matematička logika - određuje postupke ispravnog rasuđivanja
  ⇒ dva temeljna pogleda (ontološki - što postoji u svijetu; epistemološki - što agent vjeruje)
  ⇒ formalni jezici koji predstavljaju informaciju na način da se zaključci mogu izvoditi automatizirano
  ⇒ sintaksa - struktura rečenice u jeziku
  ⇒ semantika - značenje rečenica (istinitost rečenice u promatranom svijetu)
  ⇒ nama bitne vrste logike: propozicijska, predikatna i vremenska
  ⇒ temelj je formalan sustav, a niti jedan formalan sustav ne može osigurati da su polazne pretpostavke istinite
  ⇒ primjena: matematika, formalna logika, zagonetke, oblikovanje računalnih sustava, automatizirano upravljanje temeljem istinitih formula, ...
- propozicijska logika - logika sudova/iskaza/tvrdnji
  ⇒ ispravna, kompletna i odrediva (op. a. definicije ovih svojstava su niže u skripti), jer operira s konačnim skupom simbola
  ⇒ sintaksa: deklarativne rečenice (neovisno o istinitosti) preslikavaju se u sustav simbola
  ⇒ sustav se sastoji od
    - prebrojivog skupa atoma/simboličkih varijabli/simbola (PS)
    - logičkih operatora
  ⇒ semantika: atomima se pridružuju obilježja istinitosti (T, F) - interpretacija
  ⇒ svojstva implikacije ($P \Rightarrow Q$)
    - materijalna implikacija: namjera modelirati uvjetnu konstrukciju, a ne uzročno-posljedičnu vezu
    - tablica istinitosti:

P	Q	$P \Rightarrow Q$
T	T	T
T	F	F
F	T	T
F	F	T

  ⇒ semantička pravila: P i R su istinite; Q i O su neistinite; A može biti istinita ili neistinita
istinite formule	neistinite formule
$\neg Q$	$\neg P$
$P \wedge R$	$Q \wedge A$
$P \vee A$	$Q \vee O$
$A \Rightarrow P$	$P \Rightarrow Q$
$Q \Rightarrow A$	$P \Leftrightarrow Q$
$P \Leftrightarrow R$	$()$[3]
$Q \Leftrightarrow O$	
    - uključuje interpretaciju i evaluaciju

[3] prazna formula

⇒ pravila ekvivalencije - dvije formule su semantički ekvivalentne/jednake ako imaju jednaku istinitosnu vrijednost za svaku interpretaciju
- kontradikcija: $(A \wedge \neg A) = ()$
- dvostruka negacija: $(\neg (\neg A)) = A$
- idempotencija (jednaka važnost): $A \wedge A = A$
- jednaka važnost: $A \vee A = A$
- komutativnost: $(B \wedge A) = (A \wedge B)$; $(A \vee B) = (B \vee A)$
- asocijativnost: $(A \vee (B \vee C)) = ((A \vee B) \vee C)$
- distributivnost: $(A \wedge (B \vee C)) = ((A \wedge B) \vee (A \wedge C))$; $(A \vee (B \wedge C)) = ((A \wedge B) \vee (A \wedge C))$
- DeMorganovi zakoni: $(\neg (A \vee B)) = (\neg A \wedge \neg B)$; $(\neg (A \wedge B)) = (\neg A \vee \neg B)$
- eliminacija uvjeta: $(A \Rightarrow B) = (\neg A \vee B)$
- eliminacija dvostrukog uvjeta: $(A \Leftrightarrow B) = (A \Rightarrow B) \wedge (B \Rightarrow A)$
- transpozicija: $(A \Rightarrow B) = (\neg B \Rightarrow \neg A)$

⇒ prioriteti operatora: najviši - negacija → konjunkcija → disjunkcija → implikacija → ekvivalencija - najniži

⇒ formule i njihova obilježja
- P - zadovoljiva, ali ne i valjana (ovisi o interpretaciji: P = T je model, a P = F)
- $P \vee \neg P$ - valjana, sve interpretacije su modeli
- $P \wedge \neg P$ - kontradiktorna (nezadovoljiva), nema modela
- () - kontradiktorna
- $P \Rightarrow (Q \Rightarrow P)$ - valjana/tautologija
- $P \wedge Q$ - zadovoljiva (samo jedan model)

⇒ terminologija: k_i je literal, a $(ki_1 \vee ... \vee ki_n)$ je klauzula (disjunkcija literala)

⇒ svaka formula može se preslikati u formulu u disjunkcijskom normalnom obliku (DNF)
- $(k1_1 \wedge ... \wedge k1_n) \vee (k2_1 \wedge ... \wedge k2_m) \vee (k3_1 \wedge ... \wedge k3_p)$
- govori je li formula **zadovoljiva** - ako su sve disjunkcije neistinite ili sve sadrže komplementarne literale (npr. $A \wedge \neg A$), nije zadovoljiva, inače je

⇒ svaka formula može se preslikati u formulu u konjunkcijskom normalnom obliku (CNF)
- $(k1_1 \vee ... \vee k1_n) \wedge (k2_1 \vee ... \vee k2_m) \wedge (k3_1 \vee ... \vee k3_p)$
- primjer: $\neg(P \Rightarrow Q) \vee (R \Rightarrow P)$
  1. eliminiraj implikaciju pomoću $(A \Rightarrow B) = (\neg A \vee B)$: $\neg(\neg \mathbf{P} \vee \mathbf{Q}) \vee (\neg \mathbf{R} \vee \mathbf{P})$
  2. DeMorgan$[\neg(A \vee B) = \neg A \wedge \neg B]$ + eliminacija dvostrukih negacija: $(\mathbf{P} \wedge \neg \mathbf{Q}) \vee (\neg R \vee P)$
  3. asocijacijska i distribucijska pravila: $(P \vee \neg R \vee P) \wedge (\neg Q \vee \neg R \vee P)$ → $(P \vee \neg R) \wedge (\neg Q \vee \neg R \vee P)$
- podrazumijevanje klauzula: ω_1 podrazumijeva klauzulu ω_2 ako su literali u ω_1 podskup literala u ω_2
- govori je li formula **tautologija** - ako sve klauzule sadrže istinitost ili sve sadrže komplementarne literale (npr. $A \vee \neg A$), formula je tautologija, inače nije

⇒ semantička ekvivalencija - definicija preko pojma logičke posljedice: dvije formule A i B su semantički ekvivalentne ako imaju jednaku istinitosnu vrijednost za svaku interpretaciju i usto vrijedi $(A \models B)$ i $(B \models A)$
- za dokaz ekvivalentnosti, treba dokazati da je $(A \Rightarrow B) \wedge (B \Rightarrow A)$ tautologija

⇒ teorem dedukcije: formula A je logička posljedica formule B $(B \models A)$ ako i samo ako je formula $(B \Rightarrow A)$ tautologija
- $(B \models A)$ ako i samo ako je $(B \wedge \neg A)$ nezadovoljiva - koristi se za dokazivanje logičke posljedice

⇒ SAT problem (problem zadovoljivosti) - temeljni NP problem
- traži se model skupa formula Γ, što je ekvivalentno traženju modela jedne složene formule koja se sastoji od konjunkcije svih formula u Γ (najčešće dan u CNF obliku)
- iscrpna procedura rješavanja CNF SAT problema sistematski pridjeljuje istinitosne vrijednosti atomičkim propozicijskim simbolima (za n atoma 2^n pridruživanja) - eksponencijalna složenost, za opći slučaj računalno neizvedivo
- za DNF polinomska složenost - postoji konačan broj literala, a dovoljno naći zadovoljivost samo u jednom disjunkcijskom članu
- mnogi stohastički algoritmi troše eksponencijalno vrijeme za najgori slučaj, a polinomsko za srednji

- formalan sustav
  - ⇒ definira se kao dvojka {Γ, L}
    - Γ - konačan skup ispravno definiranih formula
    - L - konačan skup pravila zaključivanja
  - ⇒ mehaničko generiranje dodatnih istinitih formula bez razumijevanja konteksta - pogodno za strojnu primjenu

naziv dodatne formule	ako vrijedi:	generiraj:
uvođenje konjunkcije	$P = T$ i $Q = T$	$(P \wedge Q) = T$
modus ponens	$P = T$ i $(P \Rightarrow Q) = T$	$Q = T$
modus tolens	$\neg Q = T$ i $(P \Rightarrow Q) = T$	$\neg P = T$
komutativnost ∧	$(P \wedge Q) = T$	$(Q \wedge P) = T$
∧ eliminacija	$(P \wedge Q) = T$	$P = T$ i $Q = T$
uvođenje disjunkcije	$P = T$ ili $Q = T$	$(P \vee Q) = T$
eliminacija negacije	$\neg(\neg P) = T$	$P = T$

  - ⇒ obilježja
    - sekvencija formula $\{\omega_1, ..., \omega_n\}$ ili pojedina formula ω_i je **teorem** (dokaz, dedukcija) iz skupa Γ ako je u njemu ili se može izvesti iz njega korištenjem L
    - skup je **konzistentan** ako i samo ako ne sadrži formule na temelju kojih bi ω_i i $\neg\omega_i$ (istovremeno) bili teoremi
    - sustav je **odrediv**/odlučljiv ako i samo ako postoji algoritam koji će u konačnom vremenu odrediti postoji li teorem ω_i ili ne
    - sustav je **poluodrediv**/poluodlučljiv ako i samo ako postoji algoritam koji će u konačnom vremenu odrediti teorem ako on postoji, ali ne mora u konačnom vremenu završiti s odgovorom „ne“
    - sustav je **neodrediv**/neodlučljiv ako nije ni odrediv ni poluodrediv
  - ⇒ semantika
    - interpretacija - pridruživanje istinitosti atomima
    - evaluacija - izračunavanje istinitosti složene formule
    - interpretacija je model formalnog sustava ako evaluira sve njegove formule u istinito
  - ⇒ skup formula je **zadovoljiv** ako ima barem jedan model, inače je nezadovoljiv
  - ⇒ skup formula Γ **implicira**/povlači formulu ω ako je svaki model tog skupa ujedno i model od ω
    - ω je logička posljedica skupa Γ
  - ⇒ formula je **valjana**/tautologija ako je istinita za svaku interpretaciju i evaluaciju
  - ⇒ **ispravan** je ako je svaka pravilima dokazana formula ujedno i logička posljedica skupa Γ
    - $\Gamma \vdash_L \omega_i$ implicira $\Gamma \models \omega_i$
  - ⇒ **kompletan** je ako je svaku logičku posljedicu skupa Γ moguće dokazati pravilima L
    - $\Gamma \models \omega_i$ implicira $\Gamma \vdash_L \omega_i$

- predikatna logika
  - ⇒ logika predikata prvoga reda
    - uvodi objekte, relacije, obilježja, funkcije
    - atomički predikat
      - ⇒ pred_simbol - predikat, osnovno obilježje u rečenici
      - ⇒ t_i - članovi, objekti ili odnosi u rečenici
        - konstante i varijable
        - rezervirane konstante su T i F
      - ⇒ dva načina zapisa: infiks (LISP) i prefiks (Prolog) notacija
    - funkcija - veza između objekata
    - logički operatori: $\neg$, $\wedge$, v, $\Rightarrow$, $\Leftrightarrow$
    - kvantifikacijski simboli: $\exists$ - barem jedan, $\forall$ - svi
    - pravila
      - ⇒ svaki atomički predikat je formula
      - ⇒ ako je S_i formula, tada su formule i: $\neg S$, $S_1 \wedge S_2$, S_1 v S_2, $S_1 \Rightarrow S_2$, $S_1 \Leftrightarrow S_2$
      - ⇒ ako je X varijabla, a S formula, formule su: $\exists X\ S(X)$, $\forall X\ S(X)$
  - ⇒ logika višeg reda - kvantifikacija na predikatnom (funkcijskom) simbolu
  - ⇒ poluodrediva; a čista logika je ispravna i kompletna
- formalna verifikacija računalnih sustava
  - ⇒ deduktivni pristup - implementacija dana skupom formula Γ; treba dokazati da je specifikacija sustava S logička posljedica skupa Γ
    - problem predstavljanja
    - zahtijeva stručno vođenje
    - primjena ograničena na U/I sustave (terminirajuće)
    - može se koristiti za sustave s beskonačnim brojem stanja
  - ⇒ provjera modela - provjera je li specifikacija S istinita za jednu interpretaciju
    - ograničeno na modele s konačnim brojem stanja
    - primjena u neterminirajućim (reaktivnim) sustavima
    - automatizirano izvođenje
    - zasniva se na uporabi linearne vremenske logike[4] i vremenske logike s grananjem[5]

[4] Boolean + always, until, eventually
[5] linearna vremenska logika + *„for all features“*, *„for some features“*

FORMALNA VERIFIKACIJA

- modalna logika - proširenje klasične logike „modalitetima“ istinitosti (subjektivnim konceptima)
  - ⇒ primjer
    - p - atomički propozicijski simbol, p = F u sadašnjem svijetu
      - ⇒ (moguće p) = T ako postoji bar jedan drugi svijet u kojem je p = T
      - ⇒ (nužno p) = F jer je (nužno p) = T ako i samo ako je p istinit u svim svjetovima
  - ⇒ dodavanje modalnih operatora u klasičnu propozicijsku i predikatnu logiku
  - ⇒ logike prema tipu modalnosti:
    - aletička - potrebitnost, mogućnost
    - deontička - obligatornost, dozvoljivost
    - epistemička - znanje, vjerovanje
    - vremenska - uvijek, konačno, što je bilo, što je sad, što će biti
      - ⇒ višestruki pogledi
        - propozicijska vremenska - propozicijska logika proširena vremenskim operatorima, najviša razina apstrakcije u rasuđivanju
        - vremenska predikatna logika prvog reda (varijable, funkcije, predikati, kvantifikatori) - različiti tipovi:
          - ⇒ interpretirana-neinterpretirana → pretpostavlja ili ne strukturu
          - ⇒ globalne i lokalne varijable
          - ⇒ kvantifikacija preko vremenskih operatora ili ne
        - globalna ili modularna - rasuđivanje o kompletnom sustavu ili ne
        - vremenska logika linearnog vremena - u svakom trenutku postoji samo jedna vremenska crta
        - vremenska logika s grananjem vremena - u svakom trenutku može postojati više različitih budućih vremenskih crta
        - diskretno ili kontinuirano vrijeme - u računarstvu obično diskretno
        - prošlo i buduće vrijeme - izvorno vremenska logika obuhvaća oba, ali u digitalnim sustavima uobičajeni su samo operatori budućeg vremena
      - ⇒ odabir: propozicijska, globalna, grananje, buduće vrijeme
      - ⇒ linearna vremenska struktura: Kripke struktura M - opisana uređenom trojkom (S, R, L); a.k.a. vremenska logika s grananjem (*computation tree logic* - CTL)
        - S - skup stanja
        - R - relacija dostupnosti: R $\subseteq$ S x S između svjetova - totalna binarna relacija
          - ⇒ $\forall s \in (\exists t \in S \mid (s, t) \in R)$
        - L - S → 2^{AP} - funkcija označavanja stanja: daje interpretaciju svih simbola iz skupa AP[6] za stanje s
        - tip nedeterminističkog stroja s konačnim brojem stanja
        - može se promatrati kao beskonačno stablo izvođenja sustava
        - kvantifikatori: E - postoji takav, A - vrijedi za sve
        - operatori: X - neposredno slijedi, G - za sva stanja , F - postoji takvo stanje, U - od trenutnog do nekog stanja
        - primjeri
          - ⇒ EF *a* (*exists future*) - postoji takav put da je *a* eventualno istina
          - ⇒ AF *a* (*all future*) - za sve putove vrijedi da je *a* eventualno istina
          - ⇒ EG *a* (*exists globally*) - postoji put takav da je *a* u svim stanjima istina
          - ⇒ AG *a* (*always globally*) - za sve putove vrijedi da je *a* istina
          - ⇒ E (*a* U *b*) (*exists until*) - postoji put takav da je *a* istina do ispunjenja *b*
          - ⇒ A (*a* U *b*) (*always until*) - za sve putove vrijedi da je *a* istina do *b*
          - ⇒ EX *a* (*exists next*) - postoji put takav da je *a* istina u sljedećem stanju
          - ⇒ AX *a* (*all next*) - za sve putove vrijedi da je *a* istina u sljedećem stanju

[6] skup atomičkih propozicijskih simbola

- formalna sintaksa
  - ⇒ pravila
    - svaka atomička formula je formula stanja
    - ako su f i g formule stanja, to su i ¬f i (f ∧ g), a ostale se izvode
    - ako je f formula puta, E f i A f su formule stanja
    - ako su g i h formule stanja, X g i g U h su formule puta
  - ⇒ formule stanja se evaluiraju u stanjima
  - ⇒ formule puta se evaluiraju duž puta
  - ⇒ svi drugi operatori se mogu izraziti pravilima
  - ⇒ AU i EU su binarni, a svi ostali su unarni operatori
  - ⇒ formule puta ne mogu biti ugniježđene - traže uporabu E ili A operatora da bi postale formule stanja
- semantika
  - ⇒ terminologija
    - M = (S, R, L) - model sustava
    - M, s |= f → formula f je istinita za stanje s u modelu M
    - M, s |≠ f → formula f nije istinita za stanje s u modelu M
  - ⇒ pravila

1.	M, s |= p	ako i samo ako $p^7 \in L(s)$
2.	M, s |= (a ∧ b)	ako i samo ako M, s |= a i M, s |= b
3.	M, s |= (a v b)	ako i samo ako M, s |= a ili M, s |= b
4.	M, s |= (a ⇒ b)	ako i samo ako M, s |≠ a ili M, s |= b
5.	M, s |= AX f	ako i samo ako za sve s_i takve da s → s_i vrijedi M, s_i |= f
6.	M, s |= EX f	ako i samo ako za neki s_i takav da s → s_i vrijedi M, s_i |= f
7.	M, s |= AG f	ako i samo ako za sve putove s_1 → s_2 → s_3 → ... (s = s_1) i za svaki s_i duž puta vrijedi M, s_i |= f
8.	M, s |= EG f	ako i samo postoji put s_1 → s_2 → s_3 → ... (s = s_1) takav da za svaki s_i duž puta vrijedi M, s_i |= f
9.	M, s |= AF f	ako i samo ako za sve putove s_1 → s_2 → s_3 → ... (s = s_1) za neki s_i duž puta vrijedi M, s_i |= f
10.	M, s |= EF f	ako i samo postoji put s_1 → s_2 → s_3 → ... (s = s_1) takav da za neki s_i duž puta vrijedi M, s_i |= f
11.	M, s |= A (a U b)	ako i samo ako za sve putove s_1 → s_2 → s_3 → ... (s = s_1) vrijedi da je *a* kontinuirano istinita dok se ne pojavi *b = true* u nekom stanju
12.	M, s |= E (a U b)	ako i samo ako postoji put s_1 → s_2 → s_3 → ... (s = s_1) za koji vrijedi da je *a* kontinuirano istinita dok se ne pojavi *b = true* u nekom stanju

  - ⇒ napomene
    - 11. i 12.
      - ⇒ *a* može biti istinit ili ne nakon stanja u kojem *b = true*
      - ⇒ *b* može biti istinit i prije početnog stanja *s*
    - 7.-12.
      - ⇒ konvencija je da skup budućih stanja uključuje i sadašnje stanje
      - ⇒ kao posljedica, u svakom stanju svakog modela istinite su formule:
        - p ⇒ EF p
        - (AG p) ⇒ p
        - p ⇒ A(q U p)

[7] propozicijski atomički simbol

- CTL ekvivalencije (op. a. ovo su sve dosta očite stvari, al eto, nek stoje)
  - ⇒ de Morgan
    - ¬AF f = EG ¬f
    - AF f = ¬EG ¬f
    - EG f = ¬AF ¬f
    - AG ¬f = ¬EF f
    - AG f = ¬EF ¬f
  - ⇒ X je vlastiti dual
    - ¬AX f = EX ¬f
    - AX f = ¬EX ¬f
  - ⇒ AF f = A (True U f) = ¬EG ¬F
  - ⇒ EF f = E (True U f)
  - ⇒ EG f = ¬ [A (True U ¬f)]
  - ⇒ adekvatan skup - skup postupaka dovoljan za izračun svih CTL formula (npr. postupci za izračun EX, EG, EU)
- CTL provjera modela - potrebno pronaći sva stanja koja zadovoljavaju formulu *f*, i ispitati je li željeni podskup S_0 uključen (M, S_0 |= *f*)
- razrješavanje ugniježđenih operatora - iznutra prema van

op. a. Meni zadnje dvije Srukove prezentacije nemaju nikakvog smisla, stoga sam nesposobna sastaviti ih. Prezentacije se zovu *Protok podataka* i *Ostale arhitekture*. Sretno s njima.

PROGRAMSKO OSTVARENJE KLIJENT-POSLUŽITELJ ARHITEKTURE

- klijent-poslužitelj arhitektura - razdvajanje funkcionalnosti sustava na program korisnik/klijent i program poslužitelj
  - ⇒ poslužitelj - pruža uslugu drugim programima putem komunikacijskog kanala
    - višedretvenost:
      - ⇒ glavna dretva - čeka zahtjeve za uspostavljanjem veze
      - ⇒ sporedne dretve - stvaraju se nakon uspostavljanja veze za obradu poruka pojedinog korisnika
  - ⇒ korisnik - pristupa poslužitelju s ciljem korištenja usluge
  - ⇒ način rada
    - pokreće se poslužitelj
    - poslužitelj čeka na zahtjeve za uspostavom veze
    - pokreće se korisnik (op. a. tu po meni fali da korisnik šalje zahtjev za uspostavom veze)
    - poslužitelj prihvaća zahtjev korisnika za uspostavom veze
    - poslužitelj čeka poruku od korisnika
    - poslužitelj izvodi uslugu, vraća korisniku odgovor i nastavlja čekati poruke
    - korisnik ili poslužitelj raskida vezu
    - poslužitelj prekida s čekanjem na poruke
  - ⇒

<u>poslužitelj</u>	<u>korisnici</u>
- inicijalizacija	- inicijalizacija
- čekanje zahtjeva za uspostavom veze	- slanje zahtjeva za uspostavom veze
- uspostavljanje veze	- uspostavljanje veze
- prihvaćanje poruka	- stvaranje poruka
- obrada poruka	- slanje poruka
- slanje poruka	- prihvaćanje poruka
- raskidanje veze	- raskidanje veze
- prekid čekanja zahtjeva za uspostavom veze	

  - ⇒ komunikacijski protokol
    - prijenos poruka mrežom (najčešće Internetom)
    - mrežni protokoli (najčešće TCP veze)
- Java
  - ⇒ priključnica (*socket*) - programski entitet za uspostavu, raskidanje i slanje podataka vezom (sučelje)
  - ⇒ za jednu vezu potrebna po jedna priključnica na svakoj strani (korisnik, poslužitelj)
  - ⇒ poslužitelj
    - stvara poslužiteljsku priključnicu (*server socket*)
    - za svaki prihvaćeni zahtjev za vezom stvara priključnicu (*client socket*) za komunikaciju s korisnikom
  - ⇒ korisnik
    - stvara korisničku priključnicu (*client socket*)
  - ⇒ java.net
    - poslužitelj
      - ⇒ `ServerSocket serverSock = new ServerSocket(port);`
      - ⇒ `Socket clientSock = serverSock.accept();`
      - ⇒ `serverSock.close();`
    - korisnik
      - ⇒ `Socket clientSock = new Socket(host, port);`
      - ⇒ `clientSock.close();`

⇒ java.io - upravljanje podacima
- čitanje i pisanje na razini bajtova:
  ⇒ `output = clientSocket.getOutputStream();`
  ⇒ `output.write(msg);`
  ⇒ `input = clientSocket.getInputStream();`
  ⇒ `msg = input.read();`
- čitanje i pisanje Java objekata
  ⇒ `output = new ObjectOutputStream(clientSocket.getOutputStream());`
  ⇒ `output.writeObject(msg);`
  ⇒ `input = new ObjectInputStream(clientSocket.getInputStream());`
  ⇒ `msg = input.readObject();`

⇒ java.lang - višedretvenost
- stvaranje dretve: `Thread thread = new Thread(obj);` `obj` mora implementirati sučelje `Runnable` (metoda `run`)
- pokretanje `dretve: thread.start();`
  ⇒ pokreće se metoda `run` objekta `obj`
  ⇒ dretva se zaustavlja nakon što se izvede `run`
- grupe dretvi
  ⇒ stvaranje grupe: `ThreadGroup threadGrp = new ThreadGroup(grpNm);`
  ⇒ stvaranje dretve: `Thread thrd = new Thread(threadGrp, obj);`
  ⇒ broj aktivnih dretvi: `int count = threadGrp.activeCount();`
  ⇒ lista aktivnih dretvi:
  ```
  Thread[] threadList = new Thread[count];
  threadGrp.enumerate(threadList);
  ```

⇒ *Object Client-Server Framework* (OCSF)
- korisnička strana
  ⇒ `public abstract class AbstractClient implements Runnable`
    - uspostavljanje i raskidanje veze s poslužiteljem
    - slanje i primanje podataka
    - mora se izvesti podrazred i implementirati apstraktne metode
    - konstruktor: `public AbstractClient(String host, int port)` - inicijalizira IP/DNS adresu i vrata veze, odnosno poslužitelja
    - `public void run()` - sadrži petlju koja prima poruke i obrađuje ih pozivom metode `handleMessageFromServer` (implementacija u podrazredima)
    - upravljačke metode (<<control>>)
      ⇒ `public void openConnection()`
        - stvara *client socket* za vezu s poslužiteljem (koristi varijable `host` i `port`)
        - stvara i pokreće dretvu koja izvodi metodu `run`
      ⇒ `public void closeConnection()` - raskida vezu s poslužiteljem i zaustavlja rad dretve
      ⇒ `public void sendToServer(Object msg)` - slanje poruke (može biti bilo koji objekt)
    - pristupne metode (<<accessor>>)
      ⇒ `public boolean isConnected()`
      ⇒ `public String getHost()`
      ⇒ `public void setHost(String host)` - promjena IP/DNS adrese
      ⇒ `public int getPort()`
      ⇒ `public void setPort(int port)` - promjena vrata dok korisnik nije spojen
      ⇒ `public InetAddress getInetAddress()` - dohvaća objekt s detaljnim informacijama o vezi prema poslužitelju
    - metode koje se mogu redefinirati u podrazredima (<<hook>>)
      ⇒ `protected void connectionEstablished()`
      ⇒ `protected void connectionClosed()`
      ⇒ `protected void connectionException(Exception e)` - u slučaju iznimke, npr. ako poslužitelj prekida vezu

- metoda koja se mora ostvariti u podrazredima (<<slot>>)
  ⇒ `protected abstract void handleMessageFromServer(Object msg)` - nakon što se primi poruka od poslužitelja
- korištenje
  ⇒ ostvariti podrazred i u njemu potrebne metode
  ⇒ ostvariti glavni program koji:
    - instancira podrazred
    - poziva `openConnection`
    - stvara poruku i šalje ju pozivom `sendToServer`
    - raskida vezu pozivom `closeConnection`

- poslužiteljska strana
  ⇒ `public abstract class AbstractServer implements Runnable`
  - čekanje na zahtjeve za spajanjem
  - stvaranje novih veza prema korisnicima
  - konstruktor: `public AbstractServer(int port)` - vrata na kojima će poslužitelj čekati
  - `public void run()` - sadrži petlju koja prihvaća zahtjeve za spajanjem korisnika i stvara nove objekte razreda `ConnectionToClient`
  - upravljačke metode (<<control>>)
    ⇒ `public void listen()`
      - stvara *server socket*
      - stvara i pokreće posebnu dretvu koja izvodi `run`
    ⇒ `public void stopListening()` - zaustavlja rad dretve, ne raskida postojeće veze
    ⇒ `public void close()` - zaustavlja rad dretve i raskida postojeće veze
    ⇒ `public void sendToAllClients(Object msg)` - šalje poruku svim spojenim korisnicima
  - pristupne metode (<<accessor>>)
    ⇒ `public boolean isListening()`
    ⇒ `public Thread[] getClientConnections()` - vraća sve uspostavljene veze kao polje `ConnectionToClient` objekata
    ⇒ `public int getPort()` - dohvat vrata na kojima poslužitelj čeka zahtjeve
    ⇒ `public void setPort(int port)` - postavlja mrežna vrata za idući poziv metode listen
    ⇒ `public void setBacklog(int backlog)` - veličina repa čekanja za uspostavljanje veze sa *server socket*-om
  - metode koje se mogu redefinirati u podrazredima (<<hook>>)
    ⇒ `void serverStarted()` - nakon što započne prihvaćanje spajanja
    ⇒ `void clientConnected(ConnectionToClient client)` - nakon što se uspostavi veza s korisnikom
    ⇒ `void clientDisconnected(ConnectionToClient client)` - nakon što poslužitelj odspoji korisnika
    ⇒ `void clientException(ConnectionToClient client, Throwable exception)` - kada se korisnik sam odspoji ili u slučaju mrežne pogreške
    ⇒ `void serverStopped()` - nakon što poslužitelj prestane prihvaćati zahtjeve za spajanjem
    ⇒ `void listeningException(Throwable exception)` - kada poslužitelj prestane slušati zbog kvara
    ⇒ `void serverClosed()` - nakon završetka rada poslužitelja
  - metoda koja se mora implementirati u podrazredima (<<slot>>)
    ⇒ `abstract void handleMessageFromClient(Object msg, ConnectionToClient client)` - poziva se iz `ConnectionToClient` nakon što se primi poruka od korisnika

⇒ `public abstract class ConnectionToClient extends Thread`
- slanje podataka spojenom korisniku
- ne izvode se podrazredi
- `Thread`
  ⇒ `public void run()` - prihvaća poruke korisnika i prosljeđuje ih u `handleMessageFromClient` u `AbstractServer`
- konstruktor: `ConnectionToClient(ThreadGroup grp, Socket clientSocket, AbstractServer server)` - dodaje dretvu u `grp` i pokreće ju pozivom metode `run`
- upravljačke metode (<<control>>)
  ⇒ `public void sendToClient(Object msg)` - za slanje poruke određenom korisniku
  ⇒ `public void close()` - raskidanje veze s određenim korisnikom
- pristupne metode(<<accessor>>)
  ⇒ `public InetAddress getInetAddress()` - IP adresa određenog korisnika
  ⇒ `public void setInfo(String infoType, Object info)` - sprema po ključu proizvoljne informacije za određenog korisnika
  ⇒ `public Object getInfo(String infoType)` - dohvaća informacije određenog korisnika po ključu

⇒ korištenje:
- ostvariti podrazred od `AbstractServer` i potrebne metode
- napisati glavni program koji:
  ⇒ stvara instancu podrazreda
  ⇒ poziva `listen`
  ⇒ zaustavlja rad pozivom `close` ili `stopListening`