Oblikovanje programske potpore

Ispitivanje programske potpore

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

Tema

- Definicija i ciljevi ispitivanja programske podrške
- Klasifikacija ispitivanja
- Upoznavanje procesa ispitivanja
- Strategije ispitivanja
- Principi ispitivanja sustava i komponenti
- Funkcionalno i strukturno ispitivanje
- Generiranje ispitnih slučajeva
- Automatizacija procesa ispitivanja

- Cilj:
  - upoznavanje tehnika ispitivanja kao jedne od mogućih tehnika verifikacije programske podrške
  - razumijevanje terminologije, procesa i različitosti tehnika ispitivanja

Literatura

- Sommerville, I., ***Software engineering***, 8th ed, Addison Wesley, 2007.
- Glen Myers, ***The art of software testing,*** Second ed., John Wiley & Sons, New Jersey, 2004.
- S. Siegel, ***Object-Oriented Software Testing: A hierarchical Approach***, John Wiley & Sons, New Jersey, 1996.

Pripremio: Vlado Sruk

U pripremi materijala osim literature upotrijebljeni su i drugi izvori, te zahvaljujem autorima.

Software Engineering Body of Knowledge

- Knowledge Areas
  - Software Requirements
  - Software Design
  - Software Construction
  - **Software Testing**
  - Software Maintenance
  - Software Configuration Management
  - Software Engineering management
  - Software Engineering Process
  - Software Engineering Tools And Methods
  - Software Processes and Product Quality

Software Testing

1. Software Testing Fundamentals
   - 1.1. Testing-related terminology
   - 1.2. Key issues
   - 1.3. Relationships of testing to other activities
2. Test Levels
   - 2.1. The target of the test
   - 2.2. Objectives of Testing
3. Test Techniques
   - 3.1. Based on the software engineer's intuition and experience
   - 3.2. Specification-based techniques
   - 3.3. Code-based techniques
   - 3.4. Fault-based techniques
   - 3.5. Usage-based techniques
   - 3.6. Techniques based on the nature of the application
   - 3.7. Selecting and combining techniques
4. Test-related measures
   - 4.1. Evaluation of the program under test
   - 4.2. Evaluation of the tests performed
5. Test Process
   - 5.1. Practical considerations

Što je ispitivanje?

- Cilj ispitivanja je pokazati da program ispravno obavlja željene funkcije.
- Ispitivanje je proces uspostave povjerenja ispravnog rada.
- Ispitivanje je proces pokazivanja odsustva pogrešaka.

- ***Ispitivanje je proces izvođenja programa sa svrhom pronalaženja pogrešaka.***

Ispitivanje

- Aktivnost s ciljem ***otkrivanja informacija*** o ispravnosti i kvaliteti, te ***poboljšanja*** pronalaženjem kvarova i problema ispitivane programske podrške.
- Otkrivanje informacija
  - organizirana i detaljna potraga za relevantnim informacijama
  - aktivan proces istraživanja
    - postavljanje pitanja i analiza rezultata
- Za ostvarenje cilja upotrebljava:
  - eksperimentiranje;
  - logika, matematika;
  - modeli;
  - alati (programi, mjerni instrumenti, analizatori ...)

Svojstva ispitivanja

- Ispitivanje programske podrške zasniva se na dinamičkoj verifikaciji ponašanja programa u izvođenju na konačnom broju ispitnih slučajeva, pogodno odabranih iz uobičajeno beskonačne domene izvođenja, obzirom na očekivano ponašanje.
  - temelji se na provjeri rada sustava s ispitnim ulaznim podacima
    - uobičajeno se odabiru temeljem specifikacije stvarnih podataka koje sustav treba prihvatiti.
  - apsolutno detaljno ispitivanje je nemoguće
    - praktična ograničenja resursa i vremena (cijena)
- Tehnike ispitivanja se razlikuju u načinu odabira pogodnih kriterija i ispitnih slučajeva.
- Ispitivanje mora omogućiti donošenje odluke o prihvatljivosti i očekivanim rezultatima.
  - usporedba stvarnih rezultata rada s prethodno utvrđenim rezultatima

Problem ispitivanja

- **Bez specifikacije nema ispitivanja!!**
- Ispitivanje znači usporedbu stvarnih rezultata s pretpostavljenim očekivanim rezultatima na temelju specifikacija
- The Institute of Electrical and Electronics Engineers (IEEE) definira:
  - ***ispitni slučaj/test*** - jedan ili više ispitnih slučaja/scenarija (*engl. Test case*)
  - ***ispitivanje/testiranje*** - proces analize programskog koda sa svrhom pronalaska razlike između postojećeg i zahtijevanog stanja (pogreška, kvar, bug), te vrednovanja svojstava programa

    ⇒Odgovornost za verifikaciju i validaciju
    - *Verification: "Are we building the product right".*
      - *Ponašanje programa prema specifikacijama (odsustvo kvarova).*
      - *Unit testing, integrating testing,* ***formal verification***
    - *Validation: "Are we building the right product".*
      - *Program mora odgovarati zahtjevima (zadovoljstvo korisnika)*
      - *prototipovi, ispitivanje prihvatljivosti (engl. acceptance testing)*

Verifikacija i ispitivanje

- **Tehnike verifikacije programa**
  - dinamička verifikacija
    - ***Ispitivanje/Testiranje***
  - statička verifikacija
    - Ispitivanja strukture (*engl. Structure Walkthroughs*)
    - Provjere ispravnosti
  - formalna verifikacija
    - Primjena ***formalnih metoda*** matematičke logike za dokaz ispravnosti programa

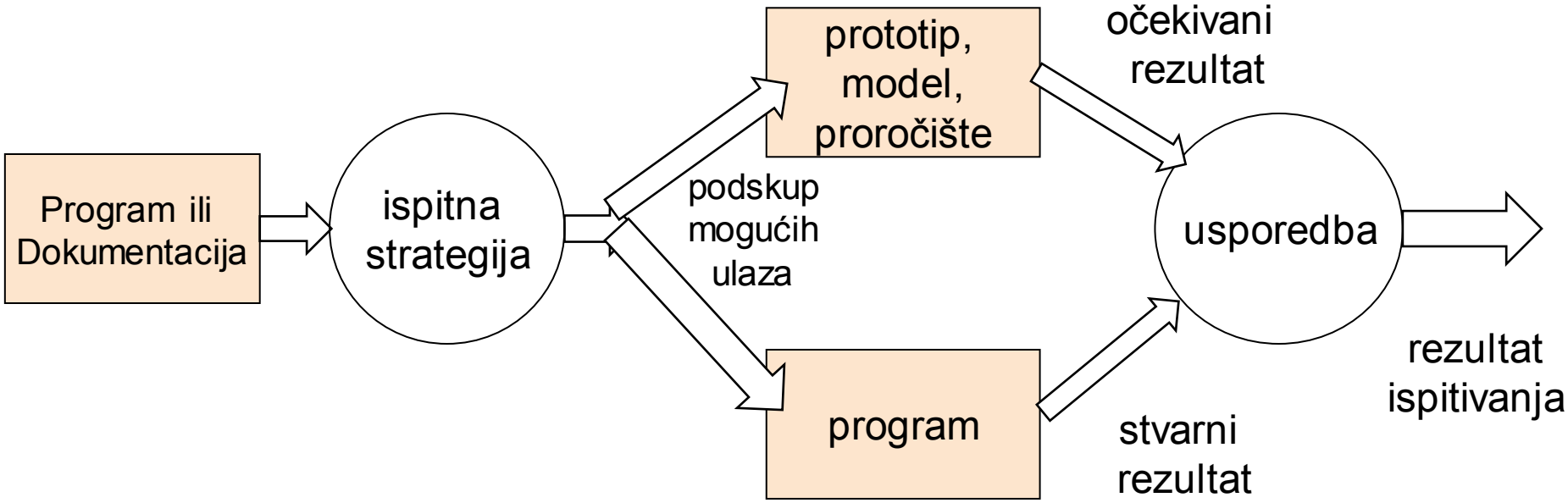
Statička verifikacija

- Statička verifikacija se provodi na specifikaciji zahtjeva, raznim nivoima oblikovanja sustava i programskom kodu
  - nadzor izvornog koda *engl. Software inspections, Walkthroughs*
  - analizatori programa *engl. Static analyzers*
    - otkrivanje nepravilnosti npr. LINT, Jtest, StyleCop
  - formalne metode
- Tehnike ispitivanja i statičke verifikacije su međusobno komplementarne i nadopunjuju se

Nadzor izvornog koda

- Analiza statičkih artefakata s ciljem otkrivanja problema
- Postupak davanja ekspertnih mišljenja, recenzija programskog produkta
  - ljudi provjeravaju izvorne artefakte
  - ne zahtijeva izvođenje
- U praksi često upotrebljavane i efikasne tehnike
  - provjera usklađenost sa specifikacijama
  - ne može provjeravati nefunkcionalna svojstva
- Prolazak, češljanje (*engl. Walkthroughs)*
  - neformalan nadzor i inspekcija izvornog koda ili dokumentacije, (često) inicirana od strane autora
- Nadzor, inspekcija (*engl. Software inspections*)
  - svrha je utvrđivanje usklađenosti sa standardima ili zahtjevima
  - usporedba dokumenata oblikovanja, koda i ostalih artefakata
  - zahtjeva planiranje i raspodjelu zadaća, formalno bilježenje i obradu rezultata

Proces ispitivanja



- uočene pogreške
- sukladnost zahtjevima
- performanse
- potvrda kvalitete

Ciljevi ispitivanja

- Ispitivanje uobičajeno obuhvaća više ciljeva, a time zahtijeva primjenu različitih strategija i ispitivanja, dokumentaciju i daju različite rezultate
- Najčešći cilj: **Pronaći i ispraviti pogreške**
- Osigurati pouzdanost, ispravnost, otkrivanje pogrešaka
- Minimizirati rizike
  - provjeriti sukladnost rada (*engl. interoperability*) s ostalim komponentama
  - pomoći u donošenju odluke o puštanju u rad/prodaju
  - zaustaviti prerano puštanje u rad/prodaju (*engl. premature product releases*)
  - minimizirati troškove tehničke podrške
  - procijeniti sukladnost specifikacijama
  - sukladnost s normama (zakonske, tehničke, …)
  - minimizirati rizik tužbi obzirom na sigurnost → vidi slijedeću sliku
- Definirati način sigurne uporabe
- Procjena kvalitete

Primjer: Microsoft EULA[1]

18. EXCLUSION OF INCIDENTAL, CONSEQUENTIAL AND CERTAIN OTHER DAMAGES. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, ***IN NO EVENT SHALL MICROSOFT OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER*** (INCLUDING, BUT NOT LIMITED TO, DAMAGES FOR LOSS OF PROFITS OR CONFIDENTIAL OR OTHER INFORMATION, FOR BUSINESS INTERRUPTION, FOR PERSONAL INJURY, FOR LOSS OF PRIVACY, FOR FAILURE TO MEET ANY DUTY INCLUDING OF GOOD FAITH OR OF REASONABLE CARE, FOR NEGLIGENCE, AND FOR ANY OTHER PECUNIARY OR OTHER LOSS WHATSOEVER) ***ARISING OUT OF OR IN ANY WAY RELATED TO THE USE OF OR INABILITY TO USE THE SOFTWARE*** THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES, OR OTHERWISE UNDER OR IN CONNECTION WITH ANY PROVISION OF THIS EULA, EVEN IN THE EVENT OF THE FAULT, TORT (INCLUDING NEGLIGENCE), STRICT LIABILITY, BREACH OF CONTRACT OR BREACH OF WARRANTY OF MICROSOFT OR ANY SUPPLIER, AND EVEN IF MICROSOFT OR ANY SUPPLIER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The guarantee - The Software is designed and offered as a general-purpose software, not for any user's particular purpose. You accept that no Software is error free and you are strongly advised to back-up your files regularly. ....

B) any support services provided by Microsoft shall be substantially as described in applicable written materials provided to you by Microsoft and Microsoft support engineers will use ***reasonable efforts, care and skill to solve*** any problem issues.

[1]EULA = End User Licence Agreement

The GPL [1]

15. Disclaimer of Warranty.

- THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. ***THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU***. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

- ***IN NO EVENT*** UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ***ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY*** WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE ***LIABLE TO YOU FOR DAMAGES***, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ***ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM*** (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

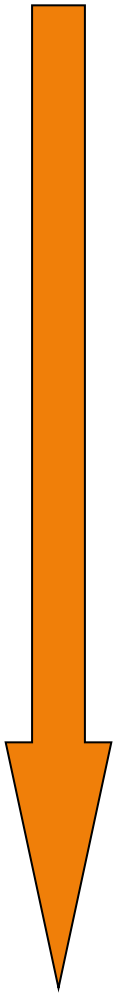
17. Interpretation of Sections 15 and 16.

- If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

[1] GPL = General Public License (GNU GPL

Kvar, pogreška, zatajenje

- ***Kvar*** (*engl. fault*)
  - proglašeni uzročnik kvara
  - može biti prikriven neko vrijeme
- ***Pogreška*** (*engl. error*)
  - dio stanja sustava odgovorno za stvaranje zastoja
  - manifestacija kvara
- ***Zatajenje*** (*engl. failure*)
  - sustav ne zadovoljava specifikacije
    - problem vidljiv izvan sustava

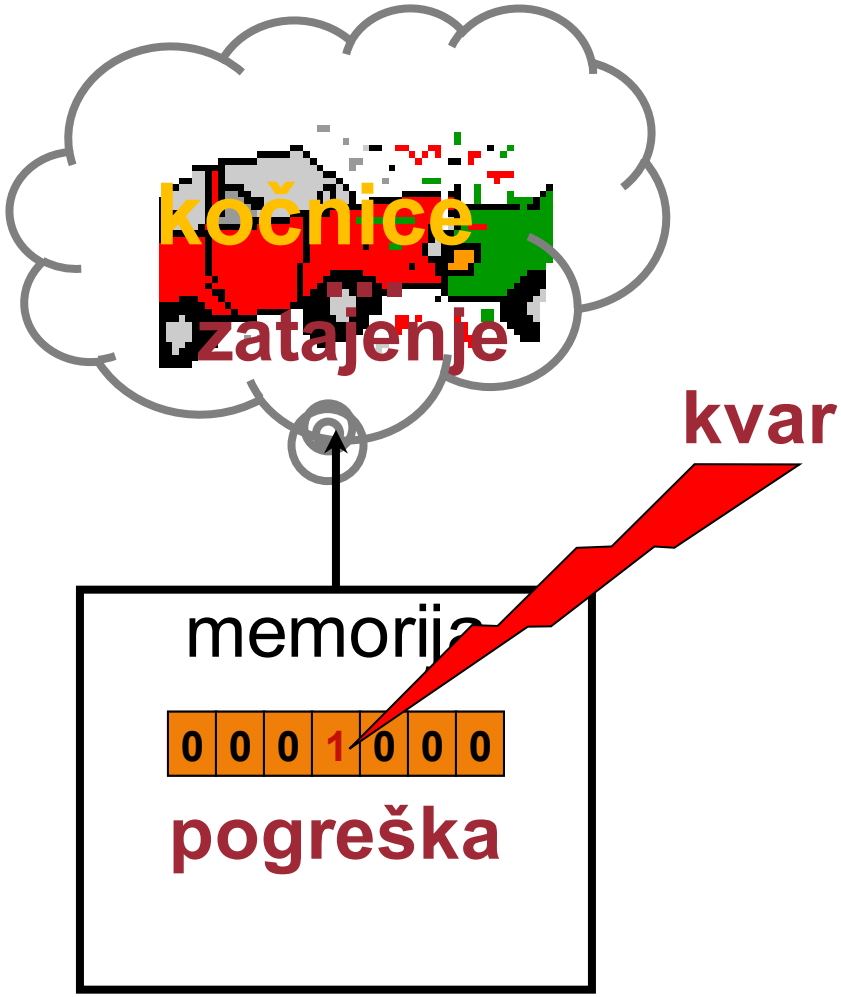


fizikalni svijet

informacija

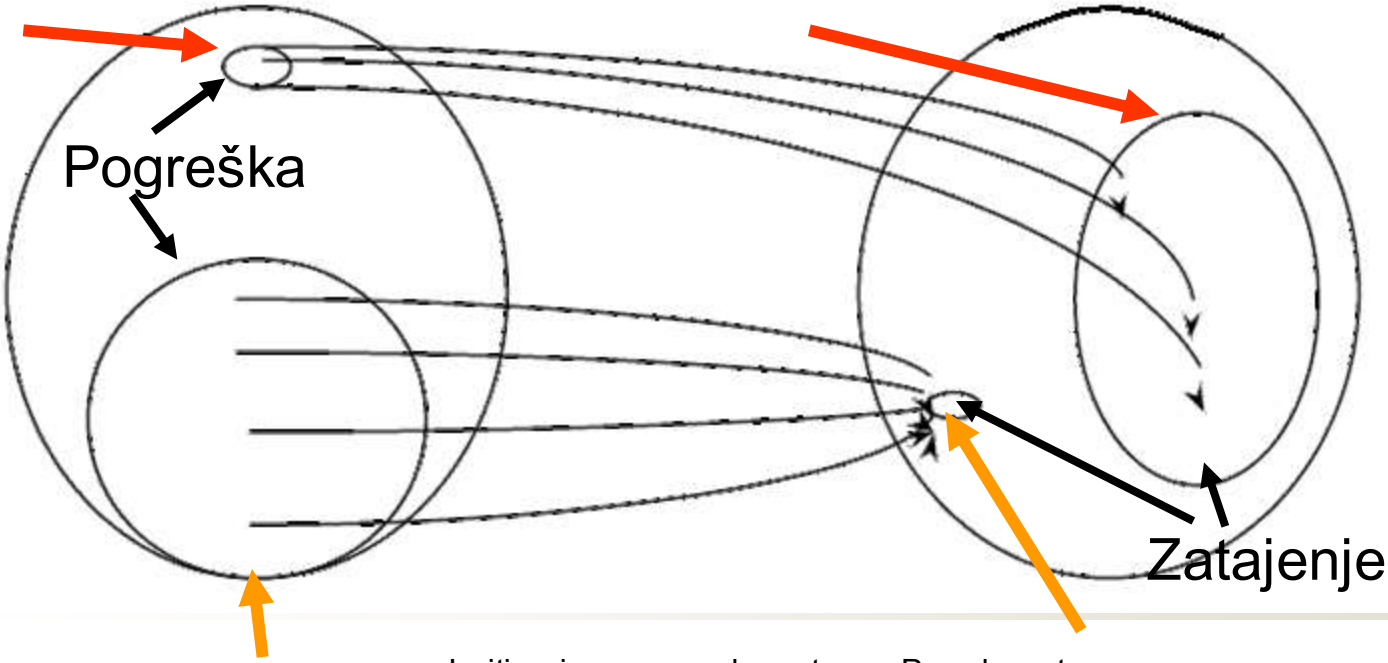
vanjska pojavnost

Primjer



Odnos pogreška i zatajenja

- **Kvar**
  - krivi rad pri oblikovanju ili programiranju
- **Pogreška** uvođenje kvara u programsku potporu (dokumentacija/program)
  - **uzrokuje** pogrešku obrade/izvođenja programa i dovodi do zatajenja
- **Paretov princip** (*engl. Pareto principle, Law of the vital few*)
  - **mali broj pogrešaka dovodi do velikog broja zatajenja (20/80)**



Primjer: kvar i pogreška

- Kvar specifikacije sučelja
  - nepodudaranje formata poruka klijenta i poslužitelja
  - nepodudaranje zahtjeva i implementacije
- Kvar u algoritmima
  - *engl. Algorithmic Faults*
  - nedostatak inicijalizacija
  - pogrešna grananja
  - zanemarivanje rukovanja nul vrijednostima
- Mehanički kvar
  - *engl. Mechanical Faults*
  - dokumentacija nije sukladna stvarnom stanju
- Pogreške
  - pogreška izazvana gubitkom poruka pri opterećenu
  - pogreške izazvana ograničenjima memorije
  - vremenske pogreške
  - …

- *primjeri vrsta kvarova:*
  - *aritmetički*
  - *logički*
  - *sintaksni*
  - *memorijski*
  - *dretveni*
  - *sučelja*
  - *performansi*
  - *timski*
  - …

Zatajenje programske potpore

- *engl. software failure*
- Mogući slučajevi:
  - ne ispunjava očekivanja zahtjeva
  - zadovoljava zahtjeve no ne i očekivanja korisnika
- Razlozi zatajenja:
  - zahtjevi su nepotpuni, nekonzistentni, nemogući za implementaciju
  - pogrešna interpretacija zahtjeva
  - kvar u oblikovanju arhitekture
  - kvar u oblikovanju programa
    - upotrijebljen neodgovarajući algoritam
  - kvar u programskom kodu
    - problem implementacije
  - dokumentacija nekorektno opisuje ponašanje sustava

Posljedice pogrešaka

- *Tipovi pogrešaka*
  - *blage (engl. mild)*
  - *dosadne (engl. annoying)*
  - *uznemiravajuće (engl. disturbing)*
  - *ozbiljne (engl. serious)*
  - *granične (engl. extreme)*
  - *katastrofalne (engl. catastrophic)*
  - *zarazne (engl. infectious)*

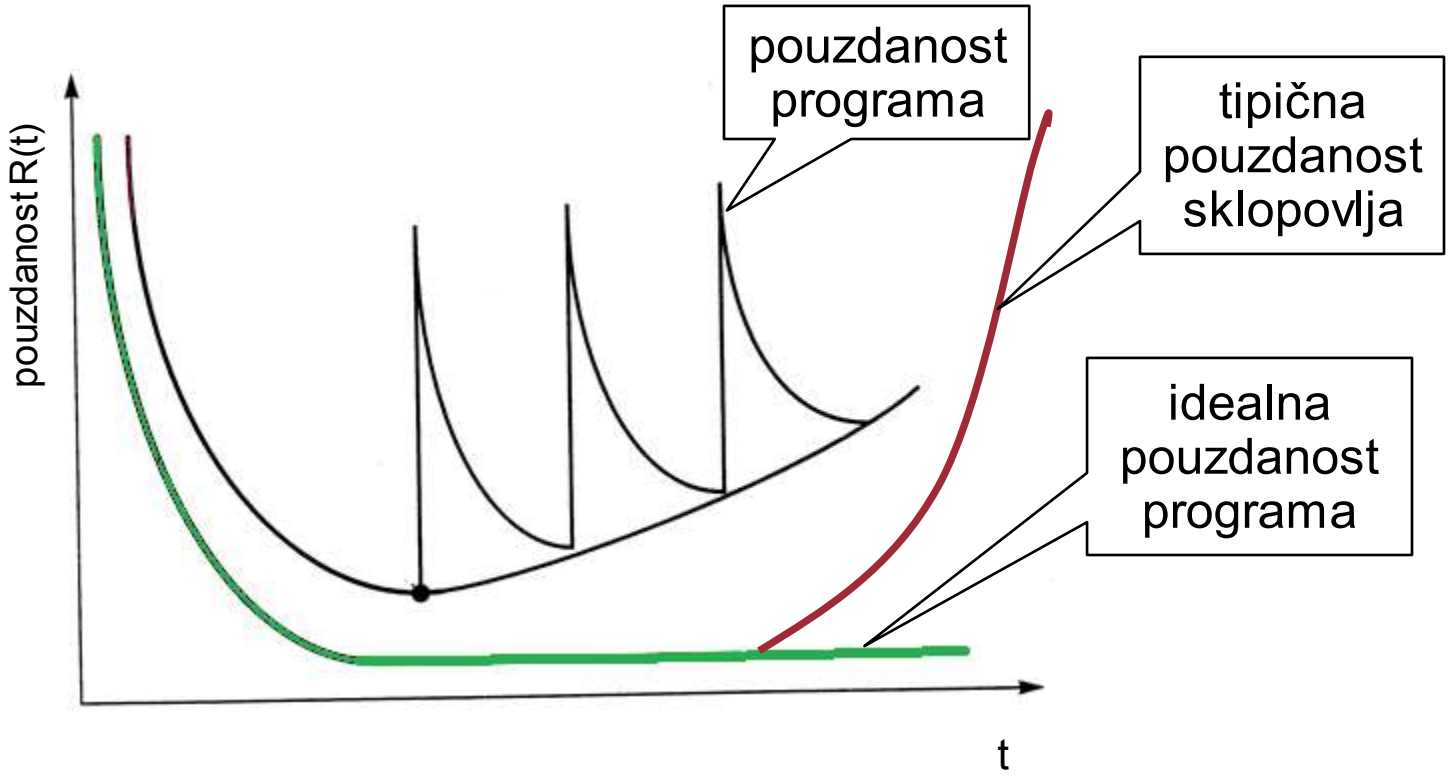
- *Kategorije pogrešaka*
  - *funkcijske, sistemske, podatkovne, pogreške kodiranja, projektiranja, dokumentacije, ...*
- *IEEE std. 1044.1 IEEE guide to classification for software anomalies*

Upravljanje pogreškama

- Pogreške u složenim sustavima su neizbježne te tražimo načine njihove minimizacije
- ***Prevencija*** – *engl. Error prevention*
  - uporaba pogodnih metoda oblikovanja za smanjenje složenosti
  - sprečavanje nekonzistentnosti - npr. CVS, ...
  - primjena verifikacije za sprječavanje kvarova u algoritmima
- ***Detekcija*** - *engl. Error detection*
  - tijekom rada
    - ***Ispitivanje***
    - Ispravljanje pogrešaka – *engl. Debugging*
    - Nadzor rada – *engl. Monitoring*
- ***Oporavak*** - *engl. Error recovery*
  - u radu programa
    - npr. dijeljenje s nulom
  - baze podataka
    - ponavljanje transakcija
  - modularna zalihost - redundancija

Pouzdanost programa

- Ispravljanje programa i uvođenje promjena programa ili sklopovlja dovodi do odstupanja od idealne krivulje pouzdanosti



Povijest ispitivanja

- Evolucija koncepta ispitivanja:

razina

- – 1956 Prva faza: ***ispravljanje pogrešaka*** (*engl. Debugging*)
  - ispitivanje = provjera rada programa i ispravljanje
  - *engl. check-out & debugging*

0

- 1957 – 1978 Druga faza: ***Orijentacija na demonstraciju*** (*engl. Demonstration*)
  - ***razdvajanje provjere rada programa i ispravljanja pogrešaka***;
  - ispitivanje pokazuje ispravan rad (tipični ulazi), sukladnost specifikaciji

1

- 1979 – 1982 Treća faza: ***Orijentacija na razaranje*** (*engl. Destruction*)
  - ispitivanje = izazivanje zatajenja sa svrhom otkrivanja pogrešaka
  - uspješno ispitivanje otkriva zatajenje

2

- 1983 – 1987 Četvrta faza: ***Orijentacija na evaluaciju*** (*engl. Evaluation*)
  - ispitivanje = dio validacije i verifikacije
  - rano otkrivanje pogrešaka u zahtjevima, oblikovanju i implementaciji
- 1988 – 2000 Peta faza: ***Orijentacija na prevenciju*** (*engl. Prevention*)
  - ispitivanje = jedan od načina pokušaja izbjegavanja pogrešaka
  - prevencija pogrešaka u zahtjevima, oblikovanju i implementaciji

3

- 2000 – Šesta faza: ***Orijentacija na razvoj prog. potpore*** (*engl. Discipline*)
  - *engl. test-driven software development*
  - rezultira programskim kodom podobnim za ispitivanje

4

Standardi ispitivanja

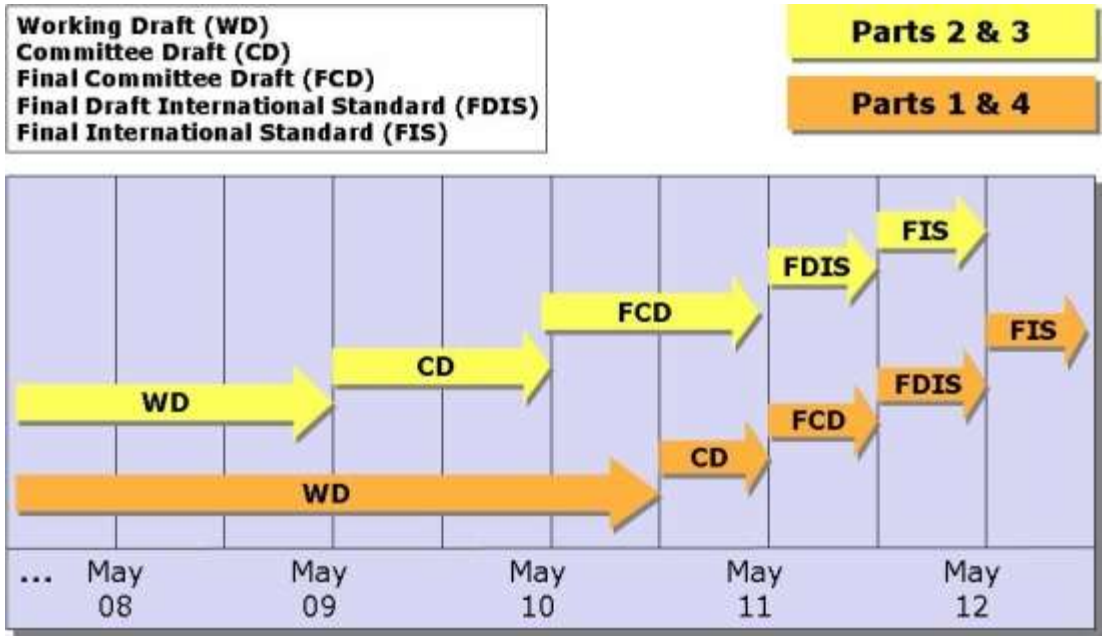
- U uporabi se nalazi velik broj definiranih standarda ispitivanja
- Standardi osiguranja kvalitete *engl. Quality Assurance standards*
  - definiraju koja ispitivanja je nužno provesti
  - npr. ISO 9003
- Industrijski standardi *engl. Industry-specific standards*
  - specificiraju razine ispitivanja
  - npr. DO178b
- Standardi ispitivanja programske potpore *engl. Testing standards*
  - specificiraju kako provesti ispitivanje

Standardi ispitivanja programske potpore

- **IEEE Standard 830-1998** - Software Requirements Specifications
- **IEEE Standard 829-1998** - Software Test Documentation
  - P829/D11, Feb 2008 Draft IEEE Standard for software and system test documentation (Revision of IEEE 829-1998)
- **IEEE Standard 1008-1987** - Software Unit Testing
- **IEEE Standard 1012-1986** - Software Verification and Validation Plans
- **ISO 9126** - Standard for the evaluation of software quality
- **ISO 25000:2005** - Software product Quality Requirements and Evaluation
- **ISO/IEC 90003:2004 - Software engineering -** Guidelines for the application of ISO 9001:2000 to computer software
  - quality management standard for computer software and related services.

Standard ISO/IEC 29119

- ISO/IEC 29119 Software Testing
  - ISO/IEC JTC1/SC7 Working Group 26
- Razvoj započeo 2007. godine
- Cilj je razviti jedan standard koji će pokrivati cijeli životni ciklus ispitivanja programske podrške



Primjer dokumenata

- IEEE Std. 829-2008 Standard for Software Test Documentation Contents

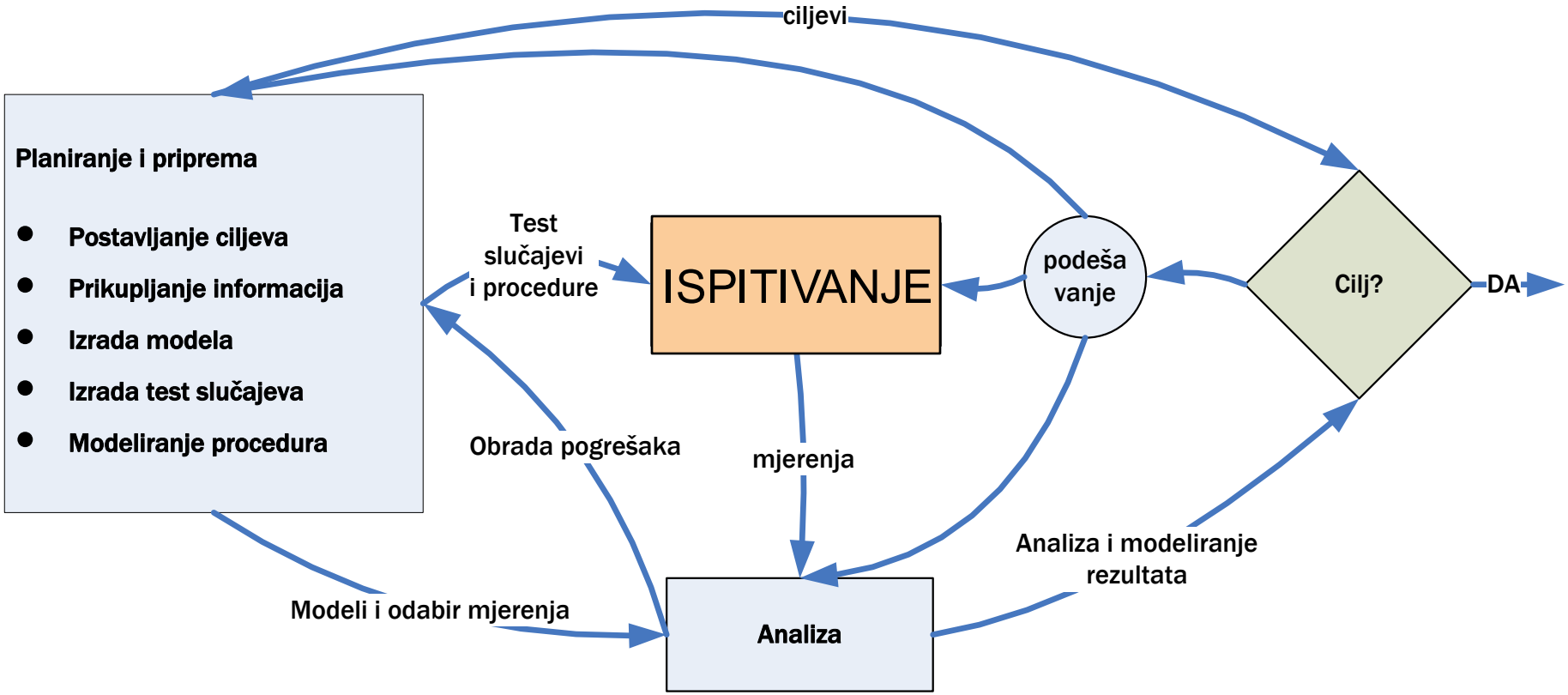
1. Plan ispitivanja – *engl. Test Plan*
   - Glavni plan ispitivanja i osnovne razine.
   - ***plan provođenja ispitiva***
2. Oblikovanje ispitivanja - *engl. Test Design Specification*
   - Svojstva, pristup i rezultata ispitivanja.
   - ***odluka što treba ispitiva***
3. Ispitni slučajevi - *engl. Test Case Specification*
   - Opis ispitnih slučajeva.
   - ***izrada ispitnih slučajeva***
4. Procedure ispitivanja- *engl. Test Procedure Specification*
   - Koraci provođenja ispitivanja.
   - ***kako se izvodi ispitivanj***
5. Bilješke ispitivanja - *engl. Test Log*
   - Zapis tijeka provođenja ispitivanja.
6. Izvješća odstupanja - *engl. Test Incident Report*
   - Bilješke anomalija u zahtjevima, oblikovanju, kodu ili provođenju ispitivanja.
7. Sažetak izvješća ispitivanja - *engl. Test Summary Report*
   - Završno izvješće

Struktura tima za ispitivanje

- Voditelj projekta (*engl. Test Manager*)
  - upravlja procesom ispitivanja i potrebnim resursima
- Analitičar
  - analiza poslovnih procesa, zahtjeva i funkcijske specifikacije
  - planiranje ispitivanja
- Voditelj upravljanja kvalitetom (*engl. Quality Assurance*)
  - definiranje standarda ispitivanja
  - praćenje i osiguranje sukladnosti procesa ispitivanja
- Voditelj ispitivanja
  - analiza zahtjeva ispitivanja
  - oblikovanje strategije i metodologije ispitivanja
  - oblikovanje ispitnih slučajeva i podatka
- Profesionalni ispitivači (*engl. Software Test Engineers*)
  - priprema ispitivanja
  - provođenje ispitivanja
  - pronalaženje pogrešaka
- Korisnici

Aktivnosti ispitivanja

- Prikaz osnovnog tijeka procesa ispitivanja



Aktivnosti ispitivanja

- Neophodna različita znanja i vještine
  - postavljanje ciljeva ispitivanja
  - oblikovanje ispitnih slučajeva
  - izrada (pisanje) ispitnih slučajeva
  - ispitivanje svih ispitnih slučajeva

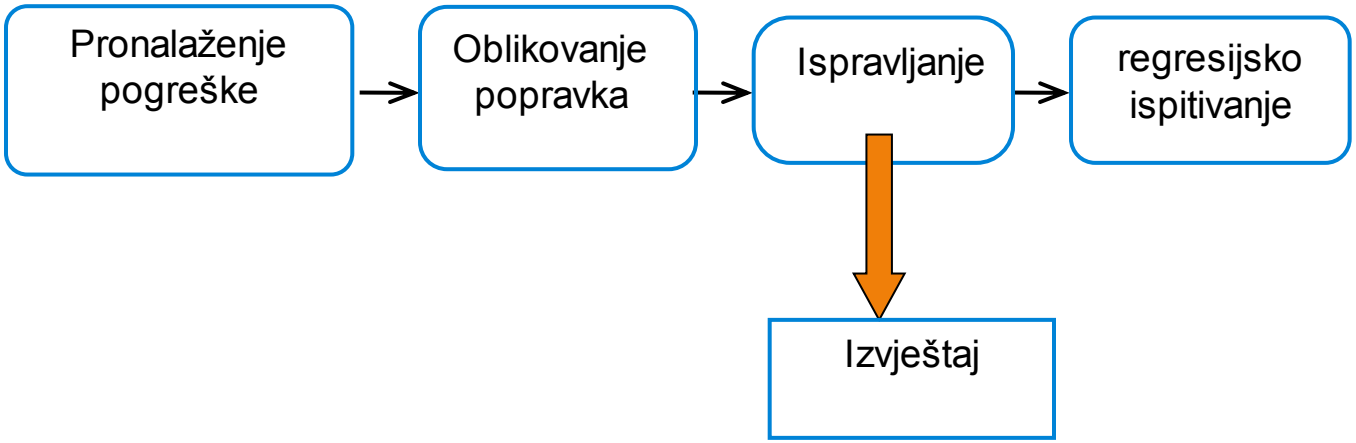
- Četiri osnovne kategorije:
- **Oblikovanje ispitivanja** (*engl. Test design*)
  - oblikovanje ispitnih vrijednosti sa svrhom zadovoljenja ciljeva ispitivanja
  - analogno oblikovanju arhitekture programske podrške
- **Automatiziranje ispitivanja** (*engl. Test automation*)
  - programiranje
- **Ispitivanje** (*engl. Test execution*)
  - provođenje ispitivanja i bilježenje rezultata
- **Valorizacija ispitivanja** (*engl. Test evaluation*)
  - poznavanje domene i postupaka ispitivanja

Aktivnosti ispitivanja

- Planiranje ispitivanja (*engl. Test Planning*)
  - planiranje procesa i aktivnosti ispitivanja, raspoređivanje resursa.
  - utvrđivanje zahtjeva, strategije i alata
- Oblikovanje ispitnih slučajeva (*engl. Test Design and Specification*)
  - upotrijebiti dobro definirane standarde i najbolje prakse
  - specificirati ispitne slučajeve tako da zadovolje postavljene zahtjeve pokrivanja
- Izrada ispitnih slučajeva (*engl. Test Set up*)
  - postavljanje ispitne okoline
- Provođenje ispitivanja
  - ručno i automatizirano
- Analiza rezultata i izvješćivanje (*engl. Test Result Analysis*)
  - sustavno izvješćivanje problemima:
- Upravljanje ispitivanjem (*engl. Test Management*)
  - upravljanje aktivnostima, kontrola plana, praćenje troškova
- Automatizacija ispitivanja (*engl. Test Automation*)
  - definiranje, razvoj i prilagodba alata
- Upravljanje ispitivanjem (*engl. Test Configuration Management*)
  - upravljanje i dokumentiranje životnim ciklusom ispitnih slučajeva, inačicama ispitnih okolina i sl.

Otkrivanje pogreške

- Pokreće proces otklanjanja pogrešaka (*engl. debugging*)
  - u okviru implementacije – programiranja
- Zahtijeva analizu i otkrivanje neispravnosti
- Odgovarajuće promjene s ciljem uklanjanja neispravnosti



Regresijsko ispitivanje

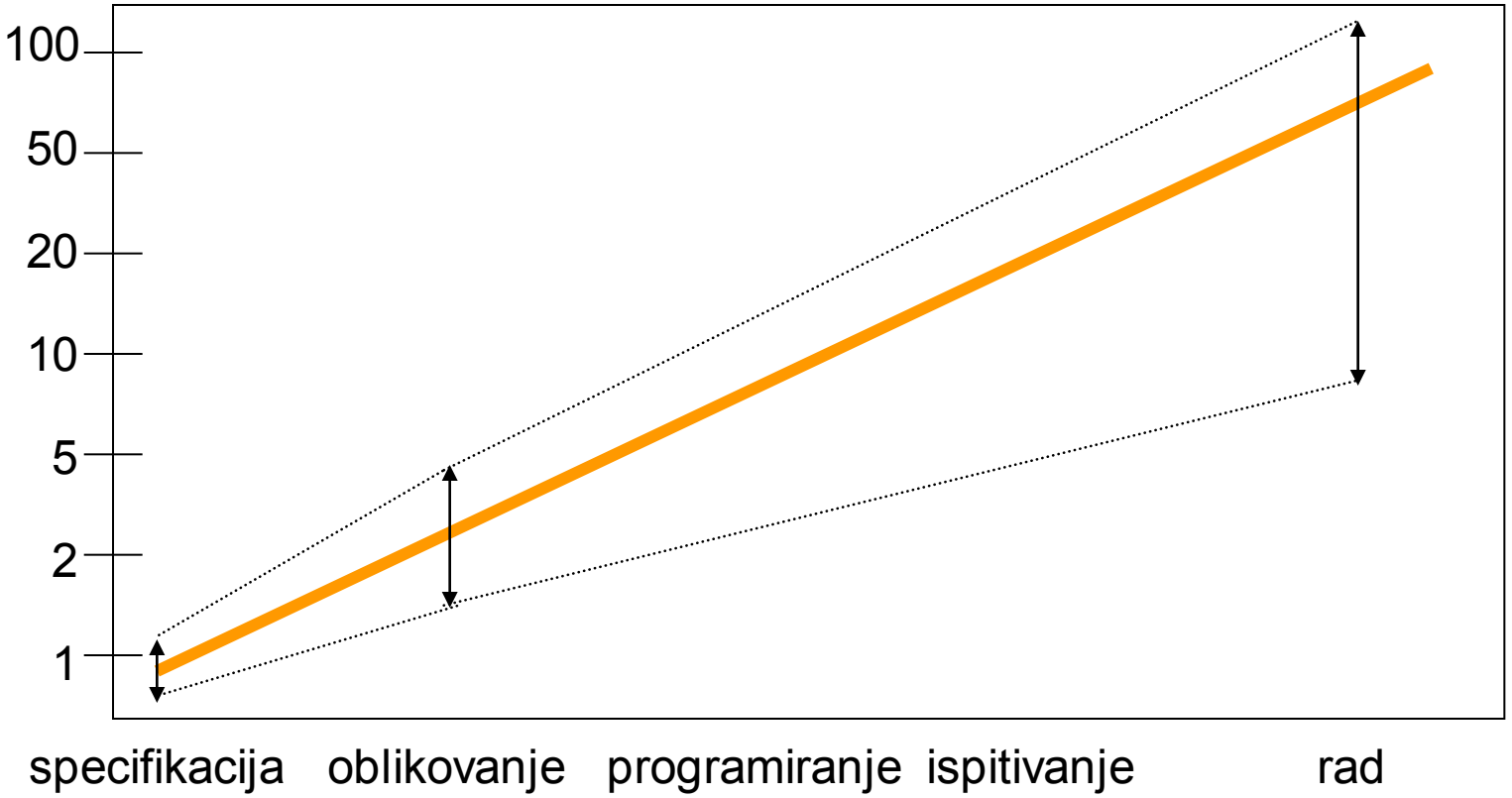
- *engl. Regression Testing*
- Ponovljeno ispitivanje nakon promjene/popravka
- Ispitivanje ispravljenih programa s ciljem potvrđivanja ispravnosti promjena i ne postojanja negativnog utjecaja na nepromijenjene dijelove programa
- Utjecaj ispravaka programa na ispitivanje:
  - neki ispitni slučajevi postaju nepotrebni
  - promjene očekivanih rezultata ispitnih slučajeva
  - izrada novih ispitnih slučajeva
- Provoditi:
  - tijekom integracije
  - nakon nadogradnji

Struktura plana ispitivanja

- Opis procesa (odabrani standard, razlozi, …)
- Sljedivost zahtjeva (*engl. Requirements traceability*)
  - omogućavanje sustavnog praćenja pojedinih zahtjeva i njegovih promjena
- Elementi ispitivanja (*engl. Tested items*)
  - što ispitujemo, što ne ispitujemo + razlozi
- Vremenik ispitivanja (*engl. Testing schedule*)
  - strategija, prioriteti, resursi …
- Procedura bilježenja rezultata (*engl. Test recording procedures*)
  - baza podataka, elementi, automatizacija…
- Zahtjevi okoline (*engl. HW & SW requirements*)
- Ograničenja
  - opis planiranih ograničenja ispitivanja

Trošak ispravljanja pogreške

- Ispravljanje uočenih pogrešaka u ranim fazama izgradnje programske potpore najjeftinije
  - raste s gotovošću programa



Ispitivanje danas

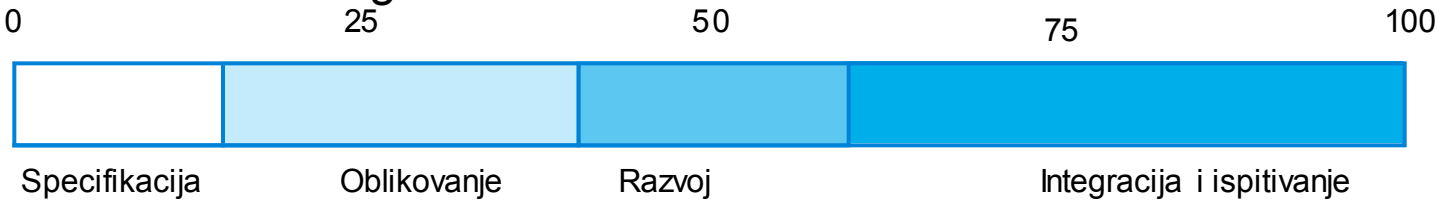
- Nije aktivnost koja započinje nakon završetka kodiranja s ograničenom svrhom otkrivanja pogrešaka
- Aktivnost objedinjena u procesu razvoja i održavanja i važan dio izgradnje produkta
- Započinje u ranim fazama analize zahtjeva
  - planovi ispitivanja i procedure moraju biti sistematično i kontinuirano razvijane, te prilagođavane sukladno razvoju
- Trošak ispitivanja predstavlja značajni udio cijene programskog produkta
  - bez obzira na primijenjenu metodologiju

Organizacija ispitivanja

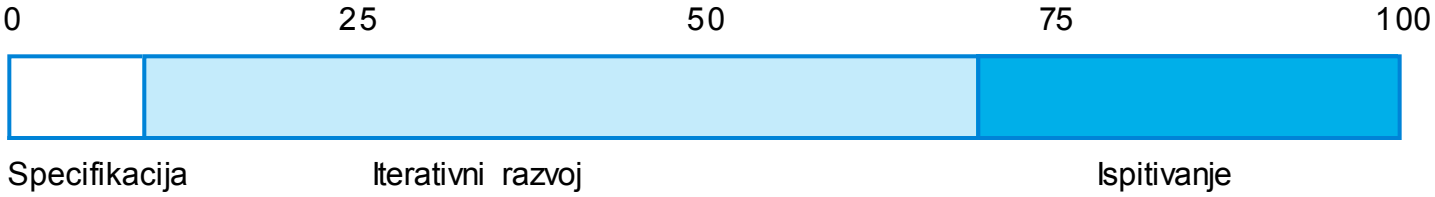
- Ovisi o svojstvima programske potpore
- Za složene programske sustave provodi se u više faza
- Različita za različite modele razvoja programske potpore
- Primjeri:
  - v-model ispitivanja
  - v-model ispitivanja s ranom pripremom
  - w-model ispitivanja

Model procesa programskog inženjerstva

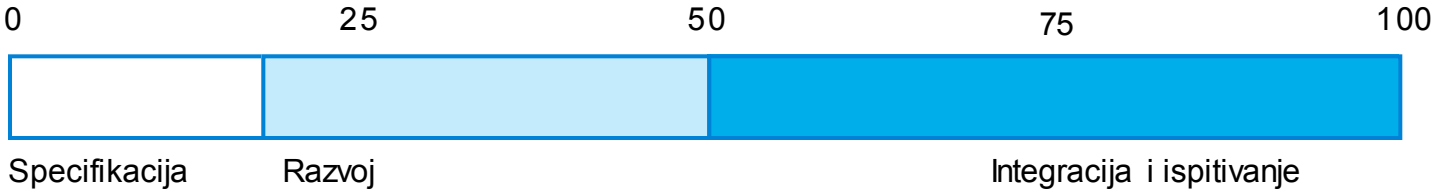
VODOPADNI engl. Waterfall model



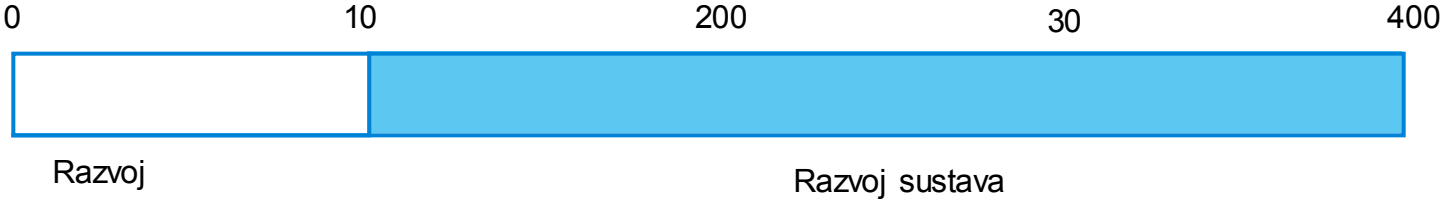
ITERATIVNI engl. Iterative development



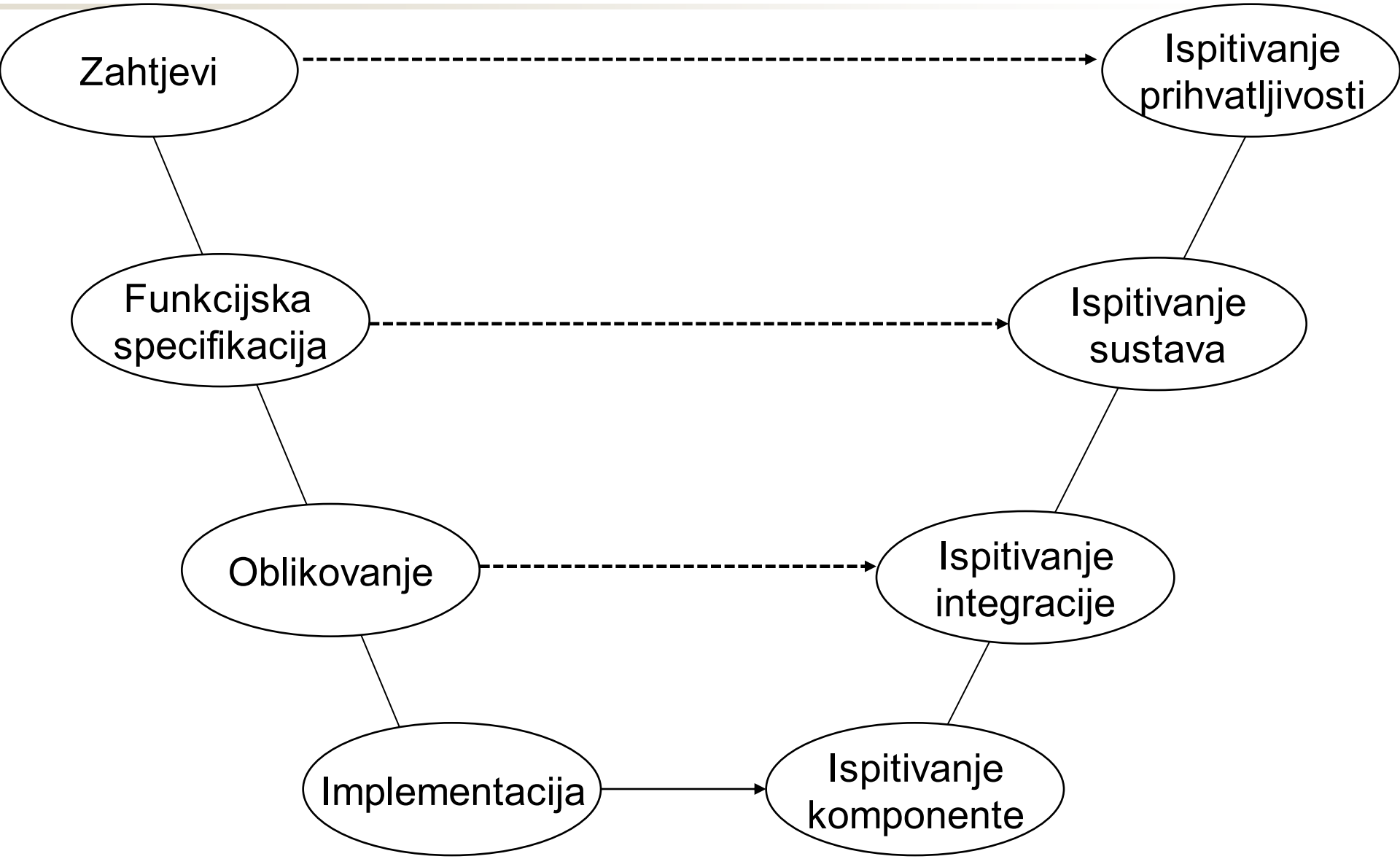
KOMPONENTNI engl. Component-based software engineering



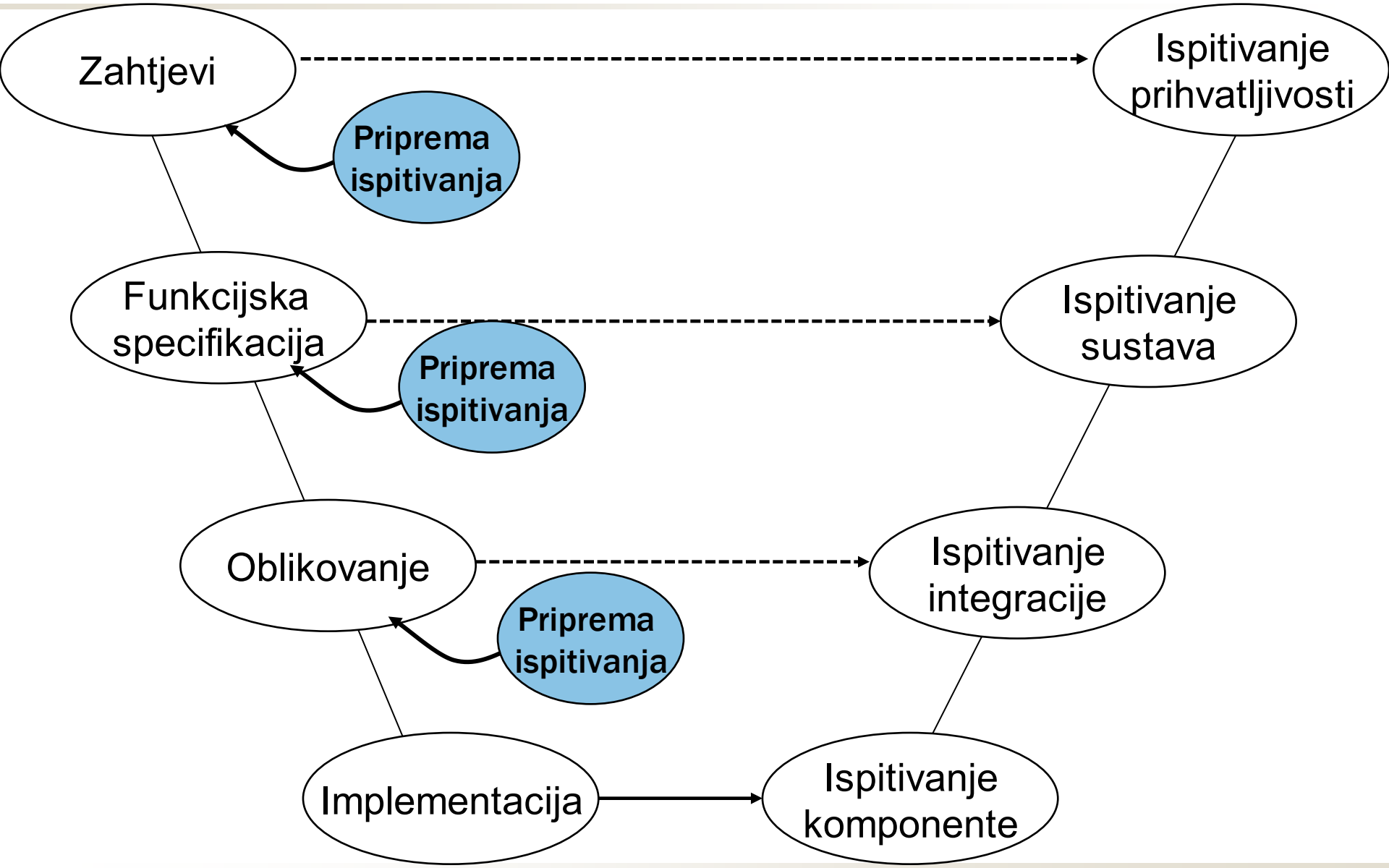
DUGOVJEČNI engl. Development and evolution costs for long-lifetime syst



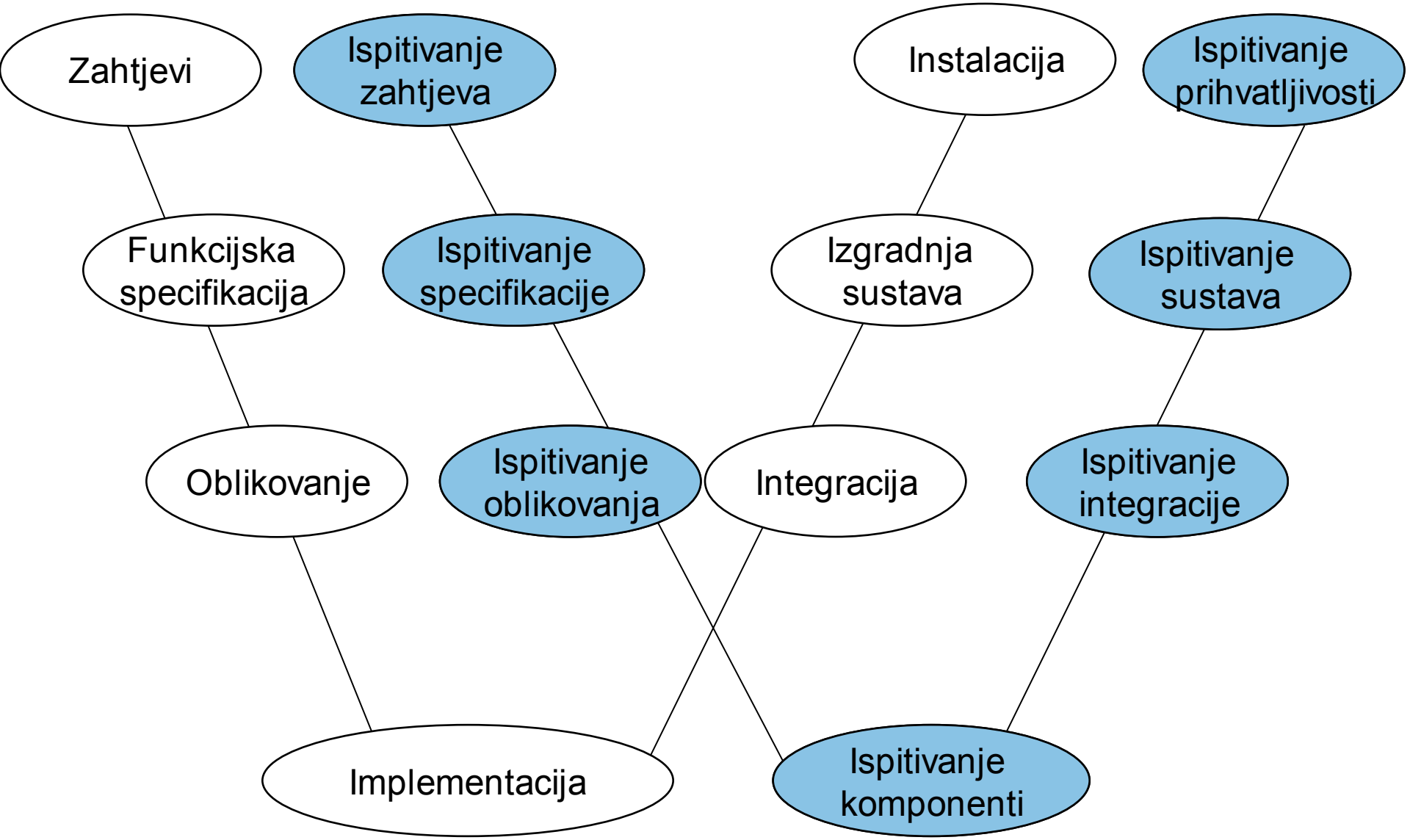
V-model ispitivanja



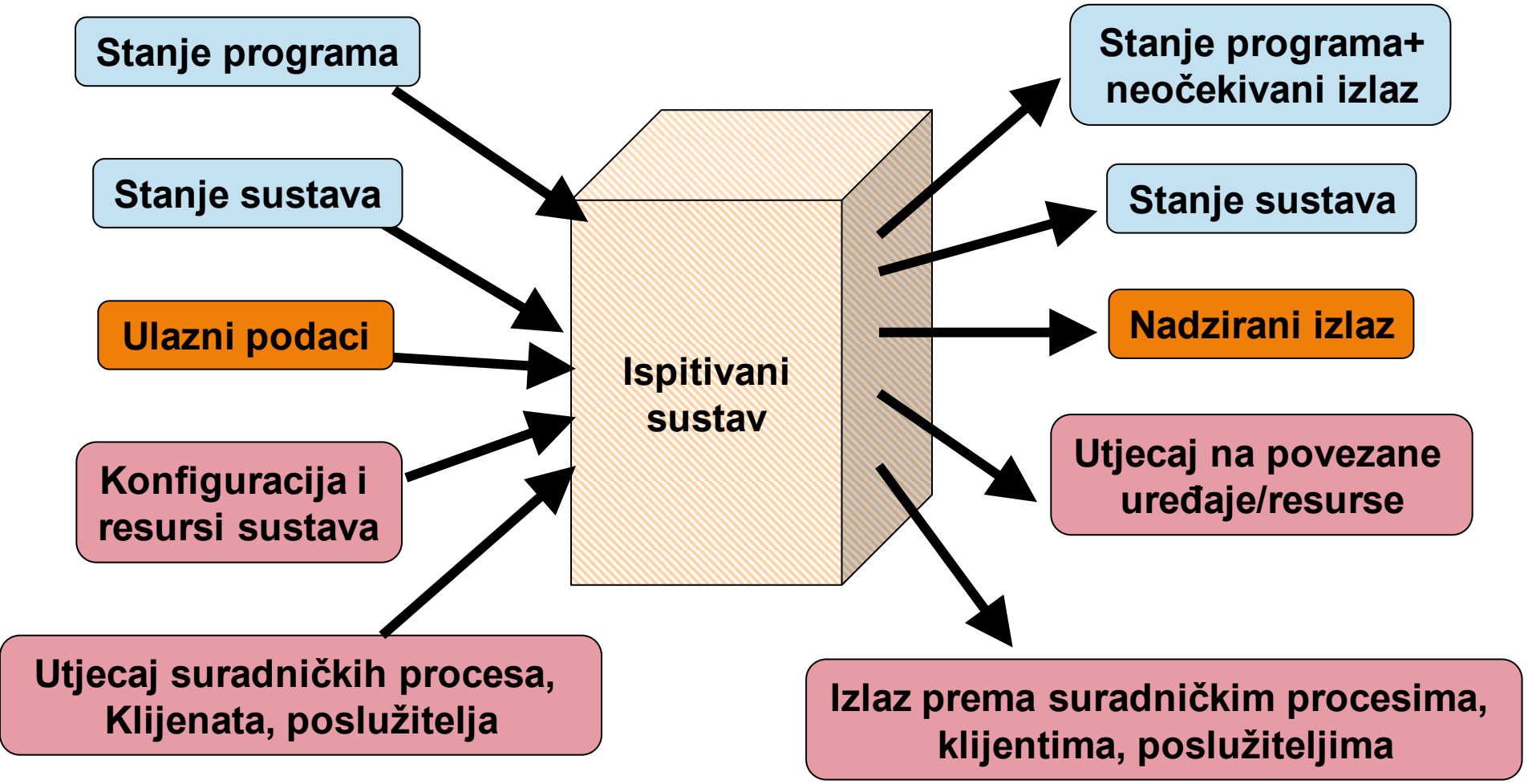
V-model ispitivanja s ranom pripremom



W-model ispitivanja

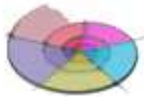


Uzročnici zatajenja programa



Svojstva ispitljivih programa

- *engl. Software Testability*
- *Ispitljivost je mjera mogućnosti jednostavnog ispitivanja programa*
  - omogućava primjenu bržeg i jednostavnijeg proces ispitivanja
- Osmotrivost (*engl. observability*)
  - jednostavna identifikacija rezultata, različiti izlazi za različite ulaze,
  - jednostavno uočavanje neispravnih rezultata
- Upravljivost (*engl. controllability*)
  - jednostavnost upravljanja tijekom provođenja ispitivanja
  - mogućnost pogodne specifikacije, automatizacije i ponovne uporabe ispitivanja
- Dekompozicija (*engl. decomposability*)
  - neovisno ispitivanje modula
- Jednostavnost (*engl. simplicity*)
  - odnosi se na složenost arhitekture, logike programa i kodiranja
- Stabilnost (*engl. stability*)
  - promjene programa tijekom ispitivanja utječu na rezultate provedenih ispitivanja
- Razumljivost (*engl. understandability*)
  - dobre informacije o strukturama, međuovisnostima i organizacija tehničke dokumentacije



Uporaba informacija u ispitivanju

- Specifikacije
  - formalne
  - neformalne
  - za odabir, generiranje, provjeru (ispitnih slučajeva)
- Informacije oblikovanja
  - za odabir, generiranje, provjeru
- Programski kod
  - za odabir, generiranje, provjeru
- Uporaba
  - povijest, model
- Iskustvo ispitivanja

Osnovni koraci ispitivanja

1. Odabir što ispitivati
   - analiza: kompletnost zahtjeva
   - oblikovanje ispitivanja
   - implementacija
2. Odluka kako ispitivati
   - statička verifikacija
     - npr. nadzor izvornog koda *engl. Software or code inspection*
   - odabir ispitivanja komponenti (*engl. Black-box, white box*)
   - odabir integracijskog ispitivanja (*engl. big bang, bottom up, top down, sandwich*)
3. Razvoj ispitnih slučajeva
4. Predikcija rezultata
   - za sve definirane ispitne slučajeve

Kada ispitivati?

- Problem ispitivanja je nemogućnost potpunog ispitivanja bilo kojeg netrivijalnog modula
- *"Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence"*
  - 1972 Dijkstra
- Tri koraka:
  - prije kodiranja
  - tijekom kodiranja
  - nakon kodiranja

Kako ispitivati?

- Ispitivanje je spoznajna aktivnost
  - zahtjeva kreativnost
- Preduvjeti efikasnog ispitivanja
  - dubinsko poznavanje sustava
  - znanje tehnika ispitivanja
  - vještine primjene tehnika na efikasan način
- Najbolje koristiti neovisne timove za ispitivanje
  - programeri za ispitivanje ***nesvjesno*** koriste podatke na kojima program radi
  - povećan rizik sukoba s razvojnim timom
- Program najčešće ne radi kada ga upotrebljava netko drugi!!!
  - najgora situacija – to otkriva korisnik.

Problemi zatajenja programa

- Ugrađena dijagnostika kao pomoć u analizi
  - umetanje provjera u kod programa
  - negativi efekti (struktura podataka, resursi, vrijeme,...)
- Zatajenje zbog efekta “nenamjernog sljepila”
  - ljudi: Ne vide ono što nije u centru pažnje
  - programi: Uvijek ne vide ono što im nije naglašeno da paze
    - Precizniji i manje prilagodljivi
- Neponovljiva zatajenja
  - razmatranje krivih uvjeta
    - nemoguće analizirati sve
- ***Ispitni slučajevi u praksi ne mogu pokriti sve mogućnosti pojavljivanja pogrešaka ☹***

Terminologija

- Ispitni podaci (I)
  - ulazi odabrani za provođenje određenog ispitnog slučaja
- Očekivani izlaz (O)
  - zabilježen ***prije*** provođenja ispitivanja
- Ispitni slučaj
  - uređeni par (I, O)
  - kako ispitivati modul, funkciju, ...
  - sadrži opis stanja prije ispitivanja, funkcije koja se ispituje
- Stvarni izlaz
  - rezultat dobiven provođenjem ispitivanja
- Kriterij prolaza ispitnog slučaja
  - kriterij usporedbe očekivanog i stvarnog izlaza određen prije provođenja ispitnog slučaja

Primjer ispitnog slučaja

- Ispitivanje funkcije "*get PIN*" za ukradene kartice
- Ulaz: 4 znamenkasti broj
- Očekivani rezultat: "Ukradena kartica"

R.br.	Ulaz	Očekivani rezultat	Stvarni rezultat	Status
1	Pročitaj karticu	Prihvati karticu i traži PIN, ako je kartica neispravna izbaci ju	Prihvaćena kartica i traži PIN	***U redu***
2	Pročitaj oštećenu karticu	Poruka "Neispravna kartica" i izbacivanje iz bankomata	Prihvaćena kartica i traži PIN	***Ne zadovoljava***
3	Unos nevaljanog PIN-a > 3	Poruka "ukradena kartica", zadržana kartica	Prihvaćena kartica i traži PIN	***Ne zadovoljava***

Primjer: Ispitni slučaj za OCSF

General Setup for Test Cases in the 2000 Series
System: SimpleChat/OCSF Phase: 2

Instructions:
1. Install Java, minimum release 1.2.0, on Windows 95, 98 or ME.
2. Install Java, minimum release 1.2.0, on Windows NT or 2000.
3. Install Java, minimum release 1.2.0, on a Solaris system.
4. Install the SimpleChat - Phase 2 on each of the above platforms.

Test Case 2001
System: SimpleChat Phase: 2
Server startup check with default arguments
Severity: 1

Instructions:
1. At the console, enter: java EchoServer.

Expected result:
1. The server reports that it is listening for clients by displaying the following message:
   Server listening for clients on port 5555
2. The server console waits for user input.

Cleanup:
1. Hit CTRL+C to kill the server.

Izvor: Lethbridge T.C., Laganière, R.: *Object-Oriented Software Engineering: Practical Software Development using UML and Java*

Izvješće o incidentu

- *engl. Test Incident Report*
- IEEE 829 IEEE Standard for Software Test
- Dobar sustav praćenja mora omogućiti jednostavan pristup svim relevantnim informacijama
- Tko je odgovoran za prijavu incidenta?
  - SVI koji ga uoče!
  - što je s korisnicima?
- Po zatvaranju mora ostati dostupno

Test incident report identifier	
Unique name or number	
Associated test plan reference (if needed)	
Summary (include references to specifications and test log)	
Incident Description	
Inputs	Opis stvarno primijenjenog ulaza
Expected results	Očekivani rezultat (iz ispitnog slučaja)
Anomalies	razlike u odnosu na stvarno + relevantni podaci
Date and time	
Procedure step	korak u kojem je došlo do incidenta
Environment	okolina ispitivanja, npr. Korisnička ispitni poslužitelj ABC
Attempts to repeat	
Testers	
Observers	
Expected impact	

Sažeto izvješće ispitivanja

- *engl. Test Summary Report*
- IEEE 829 IEEE Standard for Software Test
- Predstavlja sažetak tijeka procesa ispitivanja

Test summary report identifier		
Unique name or number		
Associated test plan reference (if needed)		
Summary		
Item tested (including version and environment)		
Summary		
Document references		
Variances		
Odstupanja od planiranog ispitivanja I stvarno provedenog		
Comprehensiveness assessment		
Summary of results		
Razriješeni problemi Nerazriješeni problemi Metrika ispitivanja – npr. pokrivenost		
Preporuke		
Evaluation		
Evaluacija pojedinog ispitivanja temeljem njihovih rezultata I ograničenja Stabilnost ispitivanja Modeliranje pouzdanosti		
Summary of activities		
Pregeld aktivmnosti ispitivanja: uključeno osoblje, utrošeni resursi		
Approvals		
Name	Job Title	Approved

Oblikovanje ispitnog slučaja

- Oblikovanje ispitnog slučaja obuhvaća određivanje ulaza i odgovarajućeg rezultata kojeg upotrebljavamo za ispitivanje sustava
- Uobičajeno predstavlja skup ispitivanja
  - težimo efikasnom otkrivanju pogrešaka

- Cilj: otkrivanje pogrešaka
- Kriterij: kompletnost
- Ograničenje: minimum resursa i vremena
- Problem:
  - kako odrediti mjesta ispitivanja?

Pristupi ispitivanju programske podrške

- ***Ispitivanje zasnovano na pokrivenosti*** (*engl. Coverage-based testing*)
  - sve naredbe moraju biti izvršene barem jednom
  - zahtjevi ispitivanja su specificirani obzirom na pokrivenost ispitivanog programa
- ***Ispitivanje zasnovano na pogreškama*** (*engl. Fault-based testing*)
  - ispitni slučajevi koji omogućavaju otkrivanje pogrešaka
  - umjetno ubacivanje pogrešaka i otkrivanje u kojoj mjeri ih ispitivanje otkriva
- ***Ispitivanje zasnovano na kvarovima*** (*engl. Error-based testing*)
  - usmjereno na kvarove graničnih vrijednosti ili maksimalnog broja elemenata
  - ispitni slučajevi zasnovani na poznavanju tipičnih mjesta izloženih kvarovima (posebice u krivoj uporabi).
- ***Funkcijsko ispitivanje*** (*engl. Black box, function, specification*)
  - ispitni slučajevi izgrađuju se temeljem specifikacije
- ***Strukturno ispitivanje*** (*engl. White box, structure, program based*)
  - ispitivanje uzima u obzir strukturu programa
- ***Ispitivanje pod pritiskom*** (*engl. Stress Based*)
  - naglasak na robusnost u kritičnim uvjetima rada

Trajanje ispitivanja

- Potpunost ispitivanja (*engl. Complete testing)*
- Problem određivanja trajanja procesa ispitivanja
- Značenje potpunog ispitivanja?
  - ispitivanje svake linije/grane/puta?
  - ne pronalaženje novih pogrešaka?
  - kraj plana ispitivanja?
- Po završetku postupka ispitivanja znamo da nema neotkrivenih pogrešaka.
  - ako postoje možemo ih pronaći novim ispitivanjem $\Rightarrow$ Ispitivanje nije potpuno
- Pojednostavljenje problema uvođenjem pojma ***potpunog pokrivanja*** (*engl. complete coverage*).

Potpuno ispitivanje

- Za ***potpuno ispitivanje*** neophodno provesti:
  - ispitivanje svih mogućih vrijednosti varijabli (ulazne/izlazne/među)
  - ispitivanje svih mogućih kombinacija ulaza
  - ispitivanje svih mogućih sekvenci izvođenja programa
  - ispitivanje svih mogućih HW/SW konfiguracija
  - ispitivanje svih mogućih načina uporabe programa
- Primjer:
  - MASPAR (Massively Parallel Computer, 64K paralelnih procesora primjena u NASI)
  - korijen 32-bitnog cijelog broja - 4.294.967.296
    - Ispitivanje - 6 minuta, usporedba tablicama
    - pronađene 2 pogreške
  - koliko traje ispitivanje za slučaj 64-bitnog cijelog broja?
    - 18.446.744.073.709.551.616 – 49029 godina

Pokrivanje ispitivanja

- *engl. Complete coverage*
- Stupanj gotovosti ispitivanja pojedinih atributa ili dijelova programa u odnosu na broj mogućih ispitnih slučajeva.
- Metrika:
  - najčešće pokrivenost koda ispitnim slučajevima
    - Postotak programskih elemenata koji su izvedeni
    - Pokrivenost linija koda
    - Pokrivenost grana
    - Pokrivenost putova
- Nedostatak: previše pojednostavljenja
  - problem prekida, paralelnosti ..
  - provjera zanimljivih podataka i kombinacija
  - nedostatak koda (*engl. black box*)
- Dobar alat za procjenu udaljenosti od potpunog ispitivanja
- Loše za procjenu gotovosti

Problemi ispitivanja ulaznih varijabli

- Velik broj kombinacija
- Uređivani ulazi podataka (*engl. edited inputs*)
  - složeni unos podataka preko raznih uređaja
- Varijacije u vremenu unosa
  - brzo, sporo
  - prije, tijekom, nakon obrade događaja
- Obrada nevaljalih ulaza (nedopustivih ?)
- Nedokumentirani odziv - Uskršnja jaja (*engl. Easter Eggs*)
  - na neobične (neočekivane) ulaze – kako otkriti ?
- Čest problem ispitivanja:
  - razmišljanje: “niti jedan korisnik to ne bi napravio”
  - interpretacija: “niti jedan korisnik *kojeg mogu zamisliti* i *koji mi odgovara*, to ne bi *namjerno* napravio”

Kombinacijsko ispitivanje

- *engl. Combination testing*
- Međutjecaj varijabli
- Za N varijabli (V_1, V_2 …V_N) broj mogućih kombinacija
  - $v_1 \times V_2 \times \ldots \times V_N$
- Primjer:
  - broj kombinacija dvije varijable definirane kao jednoznamenkasti dekadski broj
    - 10 x 10 = 100
  - koliki je broj mogućih kombinacija prva četiri poteza u šahu?
    - 318.979.564.000

Primjer: Ispitivanje ulaznih varijabli

- Koliko kombinacija ulaznih vrijednosti trebamo ispitati u slučaju programa za zbrajanja dva dvoznamenkasta cijela broja unesena preko tipkovnice čiji se rezultat ispisuje na zaslon ekrana?
  - ulazne vrijednosti -99..99
  - 39.601 kombinacija ispravnih vrijednosti ulaza
  - što je s neispravnim vrijednostima ulaza?

Ispitivanje – sažetak

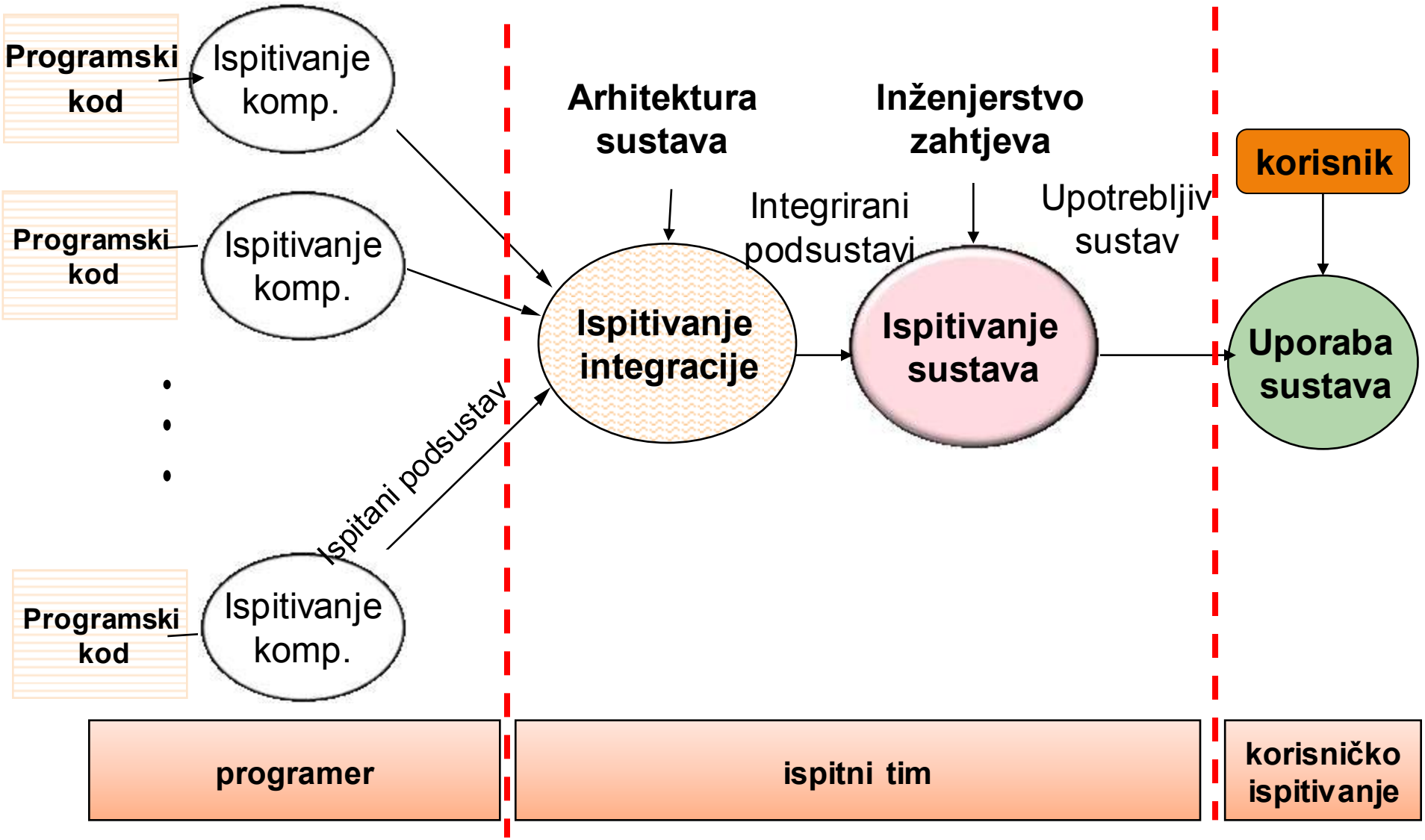
- Def.: Ispitivanje je proces izvođenja programa sa svrhom pronalaženja pogrešaka (dinamička provjera).
- IEEE standardi, dokument ispitivanja.
- Terminologija I: kvar (*engl. error*), pogreška (*engl. fault*), zatajenje (*engl. failure*).
- Terminologija II: ulazni podaci za ispitivanje, očekivani izlaz, ispitni slučaj (scenarij), stvarni izlaz, kriterij prolaza ispitivanja.
- Potrebno za ispitivanje:
  - specifikacija, informacije o oblikovanju, programski kod, uporaba programa, iskustvo.
- Ispitivanja praktično ne mogu pokriti sve mogućnosti pogrešaka.
- Potpuno ispitivanje (kriterij ?), potpuno pokrivanje.
- Kada ispitivati: V-model, V-model + priprema, W-model.
- Aktivnosti ispitivanja:
  - oblikovati, automatizirati, provoditi, evaluirati rezultate.
- Klasifikacija obzirom na odabir scenarija ispitivanja:
  - pokrivenost, pogreške, kvarovi.
- Klasifikacija obzirom na izvor informacija za scenarij ispitivanja:
  - crna kutija (kod nedostupan), bijela kutija (kod dostupan).

ORGANIZACIJA ISPITIVANJA

Proces ispitivanja sustava

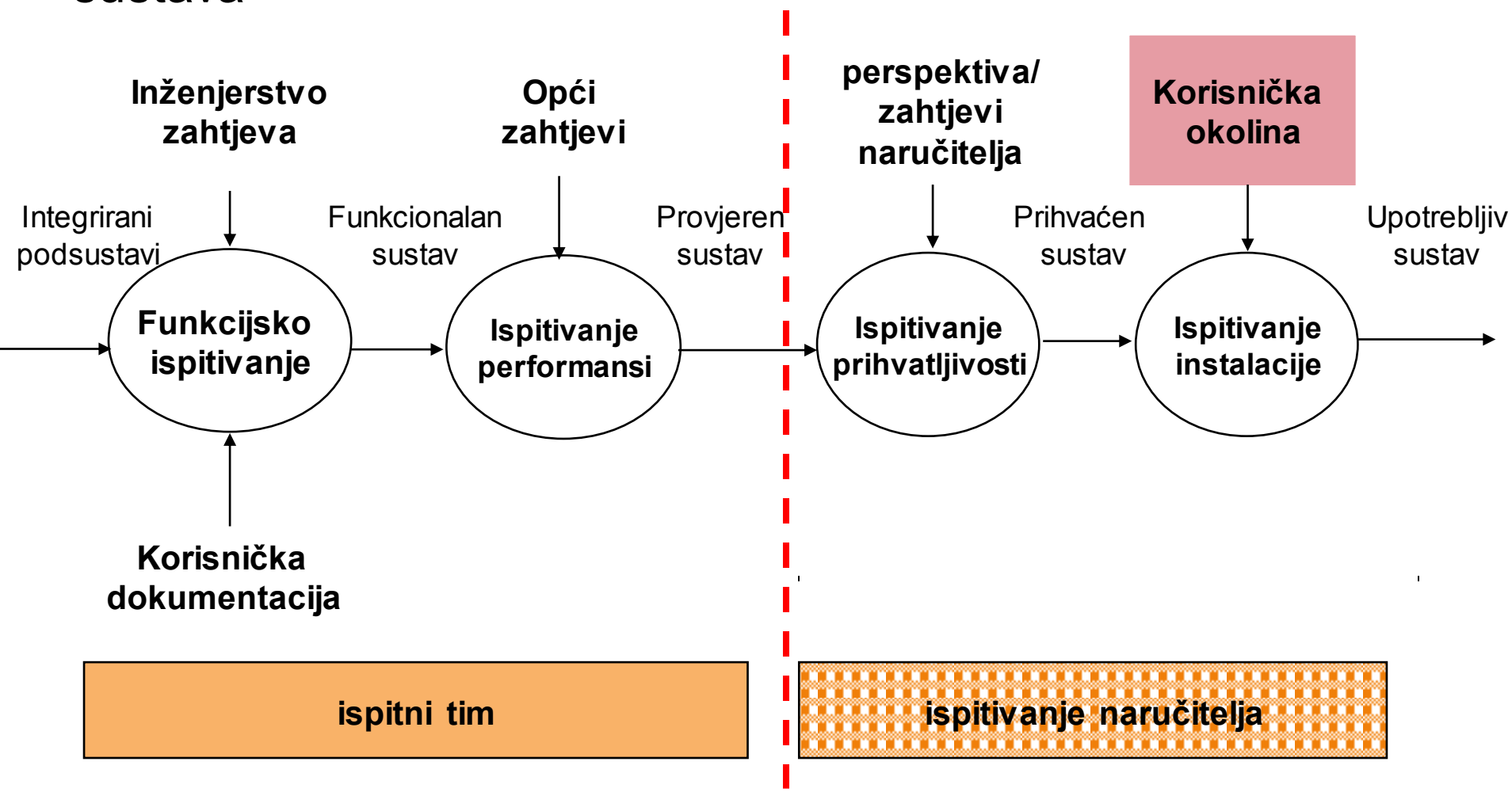
- Ispitivanje ***komponenti i modula***
  - individualne komponente se ispituju nezavisno.
  - komponente mogu biti funkcije, objekti ili koherentne skupine tih entiteta.
- Ispitivanje ***sustava***
  - ispitivanje cjelovitog sustava. Posebice je važno ispitivanje novih i nenadanih svojstava.
- Ispitivanje ***prihvatljivosti***
  - značajki na temelju kojih kupac prihvaća i preuzima sustav (*engl. acceptance*).

Organizacija ispitivanja



Ispitivanje sustava

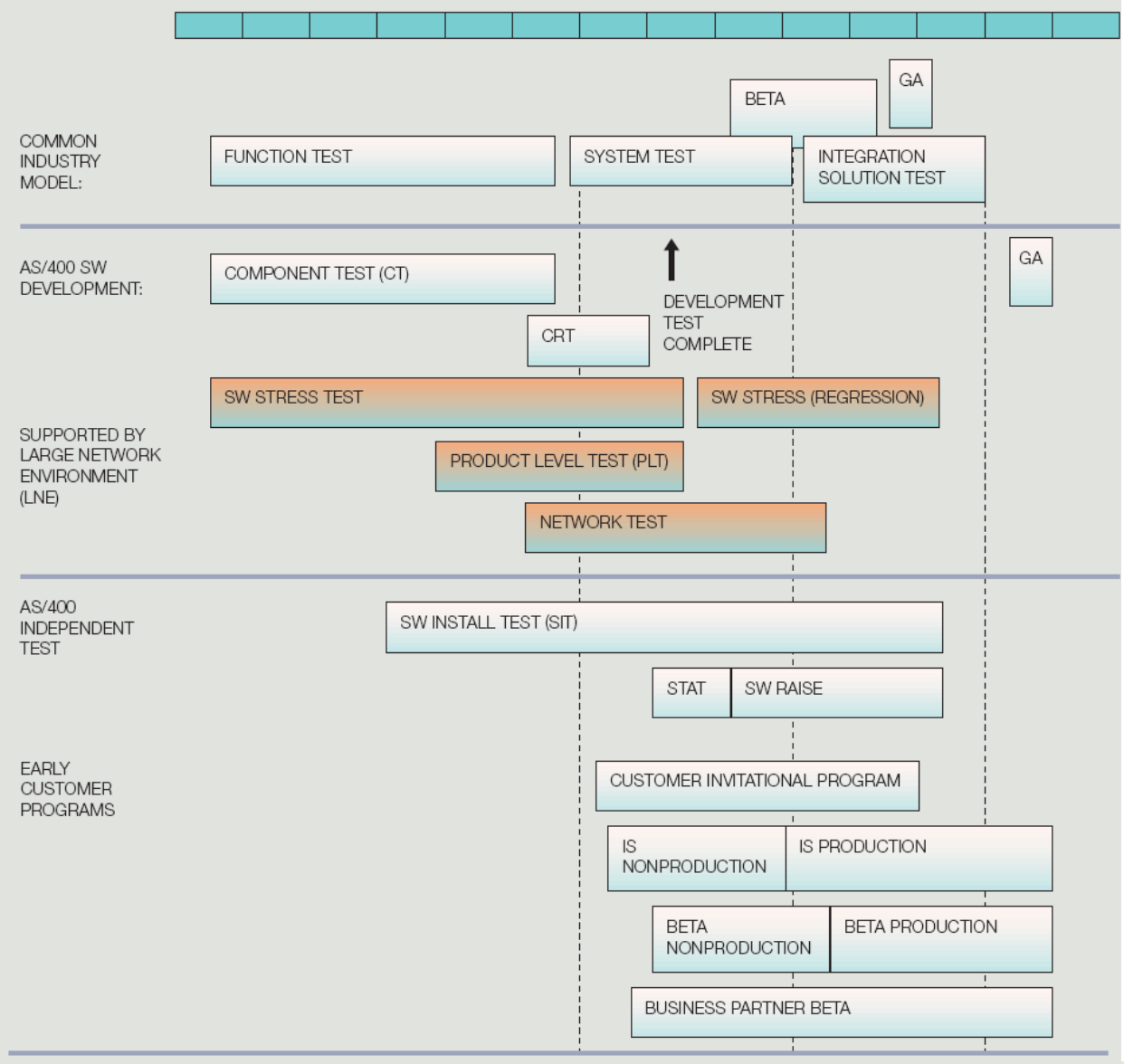
- Verificira funkcijske i nefunkcijske zahtjeve te prihvatljivost sustava



Organizacija ispitivanja

- Ispitivanje komponenti
  - *engl. Component/Unit testing*
- Integracijsko ispitivanje
  - *engl. Integration testing*
- Ispitivanje sustava
  - *engl. System/Release testing*
- Ispitivanje prihvatljivosti
  - *engl. Acceptance testing*
- Ispitivanje instalacije
  - *engl. Installation testing*
- Alfa ispitivanje
- Beta ispitivanje

Primjer procesa ispitivanja



Izvor: S. H. Kan, J. Parrish D. Manlove: In-process metrics for software testing AS/400 software testing cycle

ISPITIVANJE KOMPONENTI

Ispitivanje komponenti

- Ispitivanje koda na pogreške u algoritmima, podacima i sintaksi
- Oblikovanje ispitnih slučajeva
  - sučelje
  - strukture podataka
  - granični uvjeti
  - različiti tokovi izvođenja
  - obrada pogrešaka
  - ...
- Ispitivanje komponente provodi se u kontekstu specifikacije zahtjeva

Ispitivanje komponenti

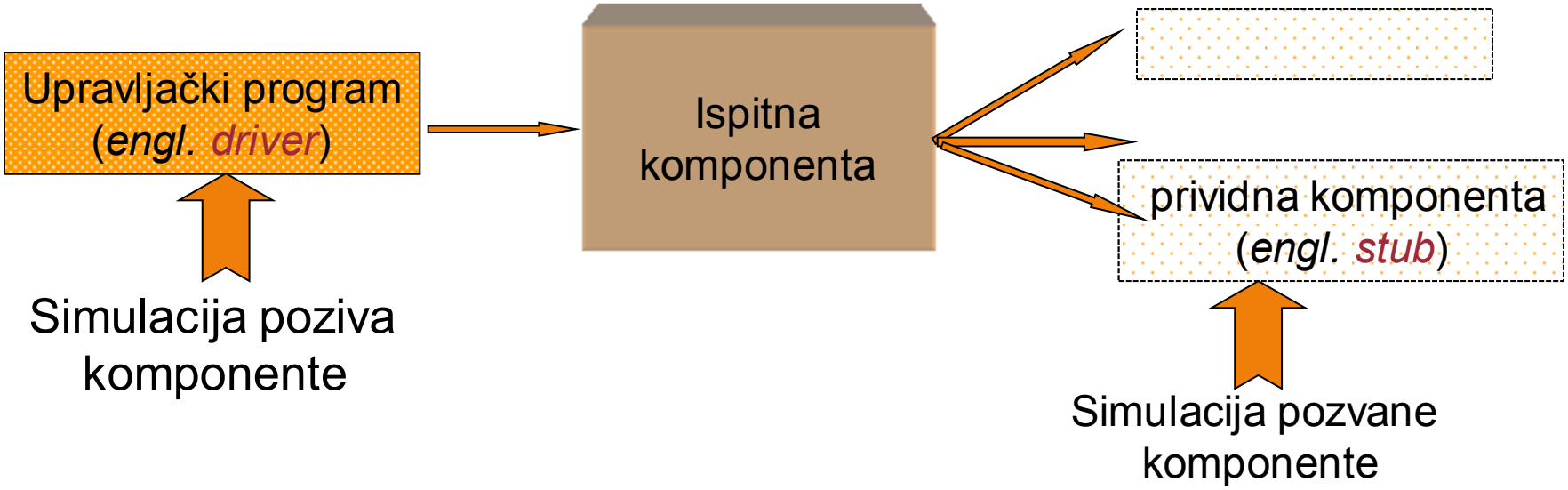
- Verificira rad programskih dijelova koje je moguće neovisno zasebno ispitati:
  - pojedinačne funkcije ili metode unutar objekta
  - klase objekata s više atributa i metoda
  - složene komponente s definiranim sučeljem za pristup njihovim funkcijama
- Postoji pristup programskom kodu i upotrebljavaju se alati za ispravljanje pogrešaka
- Tko radi?
  - najčešće programer komponente
  - ispitivanja proizlaze iz iskustva

Ispitivanje komponenti

- *engl. Unit test*
- Tijekom kodiranja
  - inkrementalno kodiranje
  - izuzetno značajno za skraćenje vremena ispitivanja
- Statičko ispitivanje
  - prolazno (neformalne prezentacije drugima)
  - nadzor koda (formalno predstavljanje drugima)
  - automatizirani alati za provjeru
    - sintaktička i semantička pogreške
    - odstupanje od standarda
- Dinamička ispitivanje:
  - funkcijsko ispitivanje, crna kutija
  - strukturno ispitivanje, crna kutija

Okolina ispitivanja komponenti

- Postupak izolacije komponente u svrhu ispitivanja
- Upravljački program (*engl. driver*) – kod koji upravlja procesom izvođenja jedne ili više komponenata
  - uobičajeno upotrebljava postojeće sučelje komponente, a može zahtijevati i stvaranje novih
- Prividna (krnja) komponenta (*engl. stub*) – kod koji simulira pozivanu komponentu
  - identično sučelje stvarnoj komponenti međutim



Elementi ispitivnja komponenti

- Sučelje
  - osigurava ispravno prihvaćanje i pružanje informacija
- Podaci struktura:
  - osigura integritet podataka
- Rubni uvjeti
  - provjera rada u graničnim slučajevima
- Nezavisni putovi
  - sve putove kroz kontrolne strukture
- Iznimke
  - provjera ispravne obrade iznimaka

Ispitivanje komponenti OO sustava

- Što je komponenta u OO sustavu?
  - za potrebe ispitivanja najčešće se uzima razred/klasa
- Ispitivanje razreda obuhvaća
  - ispitivanje svih operacija (Kako tretirati hijerarhiju?)
  - postavljanje i ispitivanje svih atributa objekta
  - ispitivanje objekta u svim stanjima (simulacija događaja koji mijenjaju stanje)
- Ispitivanje grupa razreda predstavlja oblik integracijskog ispitivanja
- Poteškoće ispitivanja razreda uzrokuju osnovni koncepti OO (enkapsulacija, nasljeđivanje, polimorfizam, ...)
  - npr. Informacije koje treba ispitati nisu lokalizirane

Koraci ispitivanja OO sustava

1. Ispitni slučaj jedinstveno označiti i eksplicitno povezati s ispitivanim razredom
2. Definirati namjenu ispitnog slučaja
3. Razraditi korake ispitivanja:
   1. definirati stanja objekata koja se ispituje
   2. definirati poruke i operacija koje se ispituju te njihove posljedice
   3. definirati listu iznimaka koje mogu proizaći tijekom ispitivanja
   4. definirati stanje okoline pri ispitivanju

Slučajno ispitivanje

- *engl. Random testing*
- Koraci
- Identificirati operacije primjenjive na razred
- Definirati ograničenja na njihovo korištenje
- Identificirati minimalni ispitni slučaj
  - slijed operacija koji definira minimalnu životni vijek instanciranog objekta
- Generirati niz ispravnih slučajnih ispitnih sekvenci

Ispitivanje particija

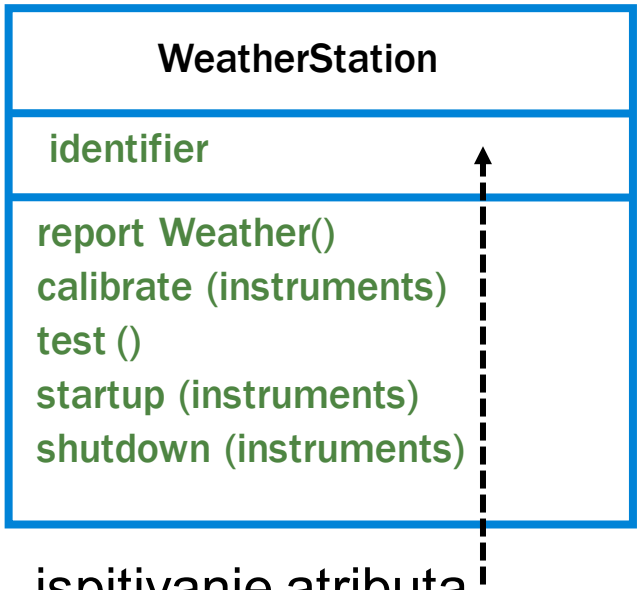
- *engl. Partition testing*
- Smanjuje broj ispitnih slučajeva potrebnih za ispitivanje razreda
  - princip ekvivalencije particija
  - particija stanja
    - *engl. state-based partitioning*
  - kategorizirati i ispitati operacije temeljem utjecaja na promjenu stanja objekta
  - particije atributa
    - *engl. attribute-based partitioning*
  - kategorizirati i ispitati operacije temeljem svojstava atributa
- Particije kategorija
  - *engl. category-based partitioning*
  - dekompozicija funkcijske specifikacije i utvrđivanje generičkih karakteristika
  - kategorizirati i ispitati operacije na temelju generičkih funkcije koje obavljaju

OO ponašajno ispitivanje

- *engl. OO behavioral testing*
- Ispitni slučajevi moraju pokriti sva stanja objekta
- Pogodna uporaba UML dijagrama stanja
- Velike mogućnosti automatizacije

Primjer

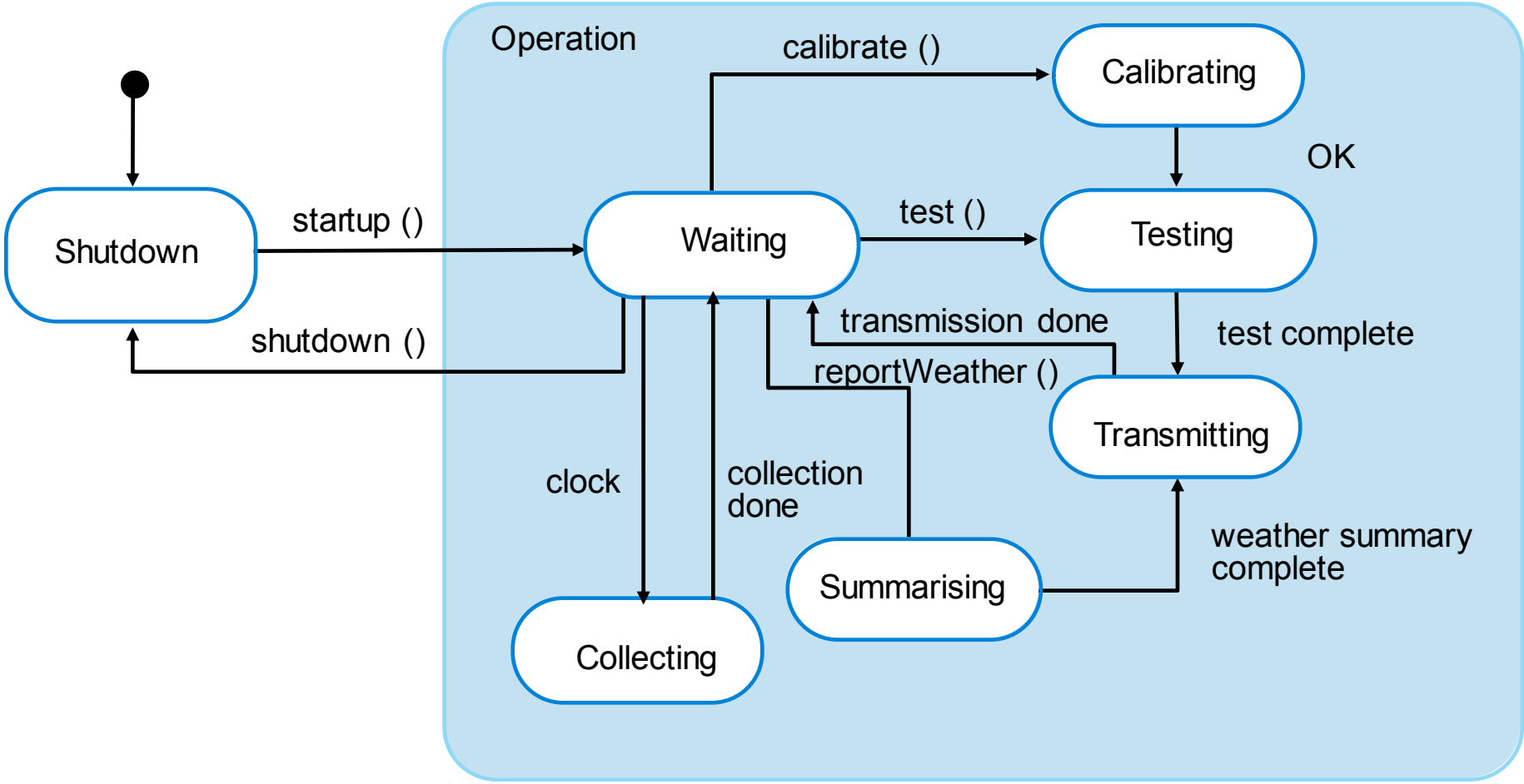
- Razred ***Weather*** *station*
- Definirati ispitne slučajeve za:
  - **reportWeather, calibrate, test, startup, shutdown**.
- Uporabom dijagrama stanja pronaći:
  - sekvence prijelaza za ispitivanje i
  - odgovarajuće sekvence događaja koje ih uzrokuju prema dijagramu



ispitivanje atributa **identifier**

•npr. provjeriti da li je postavljen

Primjer: Dijagram stanja Weather station



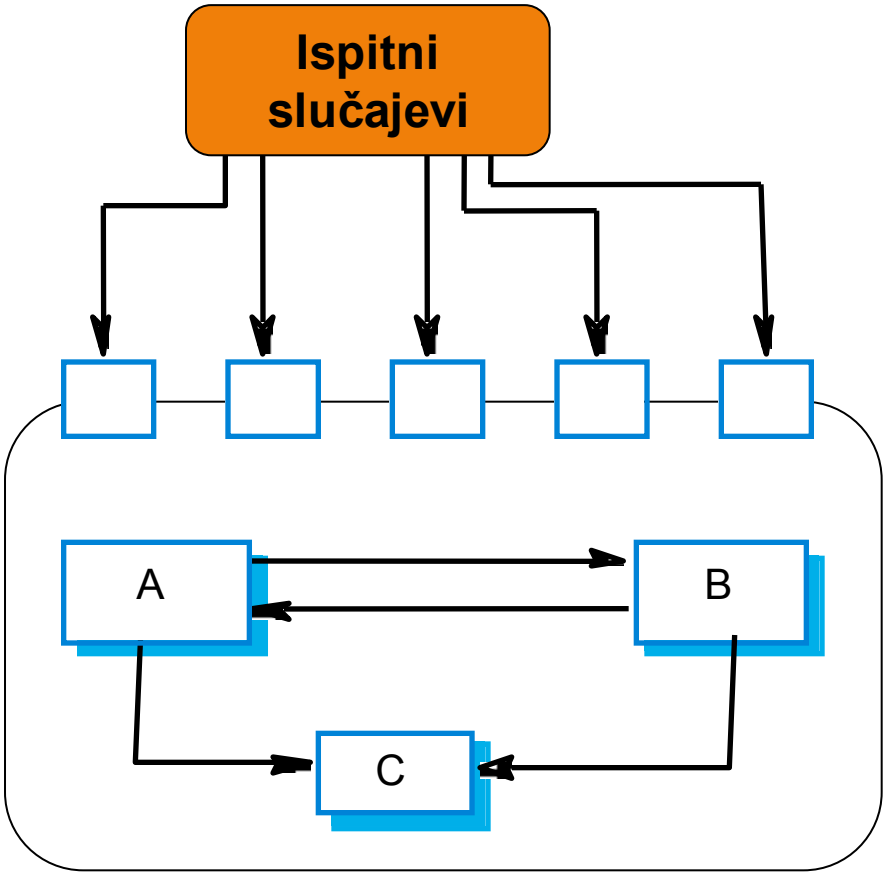
- Waiting → Calibrating → Testing → Transmitting → Waiting
- Shutdown → Waiting → Shutdown
- Waiting → Collecting → Waiting → Summarising → Transmitting → Waiting

Izvor: Sommerville, I., Software engineering

Ispitivanje sučelja

- Cilj je otkriti pogreške uzrokovane kvarovima sučelja ili pogrešnim pretpostavkama o sučelju
- Posebno značajno za OO sustave
  - objekti (i komponente) su definirani sučeljima
  - pogreške nastaju kao posljedica interakcija

- Složena komponenta



(ispitni slučaj za cijeli podsustav odnosno složenu komponentu)

Izvor: Sommerville, I., Software engineering

Tipovi sučelja programskih komponenata

- ***Parametarsko sučelje*** (*engl. Parameter interfaces*)
  - podaci i funkcije se prenose pozivima procedure
- ***Dijeljena memorija*** (*engl. Shared memory interfaces*)
  - procedure dijele zajednički memorijski prostor
- ***Proceduralno sučelje*** (*engl. Procedural interfaces*)
  - komponente obuhvaćaju skup procedura koje pozivaju ostali podsustavi (npr. objekti)
- ***Sučelje zasnovano na porukama*** (*engl. Message passing interfaces*)
  - podsustavi traže usluge od ostalih podsustava slanjem poruke (npr. klijent-uslužitelj)

Kvarovi sučelja

- ***Pogrešna uporaba*** (*engl. Interface misuse*)
  - komponenta koja poziva drugu pogrešno upotrebljava njezino sučelje
    - Npr. Pogrešan redoslijed parametara
- ***Nerazumijevanje sučelja*** (*engl. Interface misunderstanding*)
  - komponenta koja poziva drugu pogrešno pretpostavlja specifikaciju njezinog ponašanja (npr. binarno pretraživanje na neuređenom nizu)
- ***Vremenske pogreške*** (*engl. Timing errors*)
  - pozvana i pozivajuća komponenta rade ***različitom brzinom*** (npr. zajednička memorija te različita brzina čitanja i pisanja)
- Naputak za ispitivanje sučelja:
  - oblikuj ispitne slučajeve tako da parametri poprime ***ekstremne vrijednosti***
  - uvijek ispitaj pokazivače (ako se šalju sučeljem) s ***null vrijednostima***
  - oblikuj ispitni slučaj proceduralnog sučelja tako da ***zataji komponenta***
  - sustave s razmjenom poruka ispitaj na ***stres*** (generiraj više pouka nego što je normalno potrebno)
  - sustave s dijeljenom memorijom ispitaj s ***različitom redoslijedom*** aktiviranja komponenti

INTEGRACIJSKO ISPITIVANJE

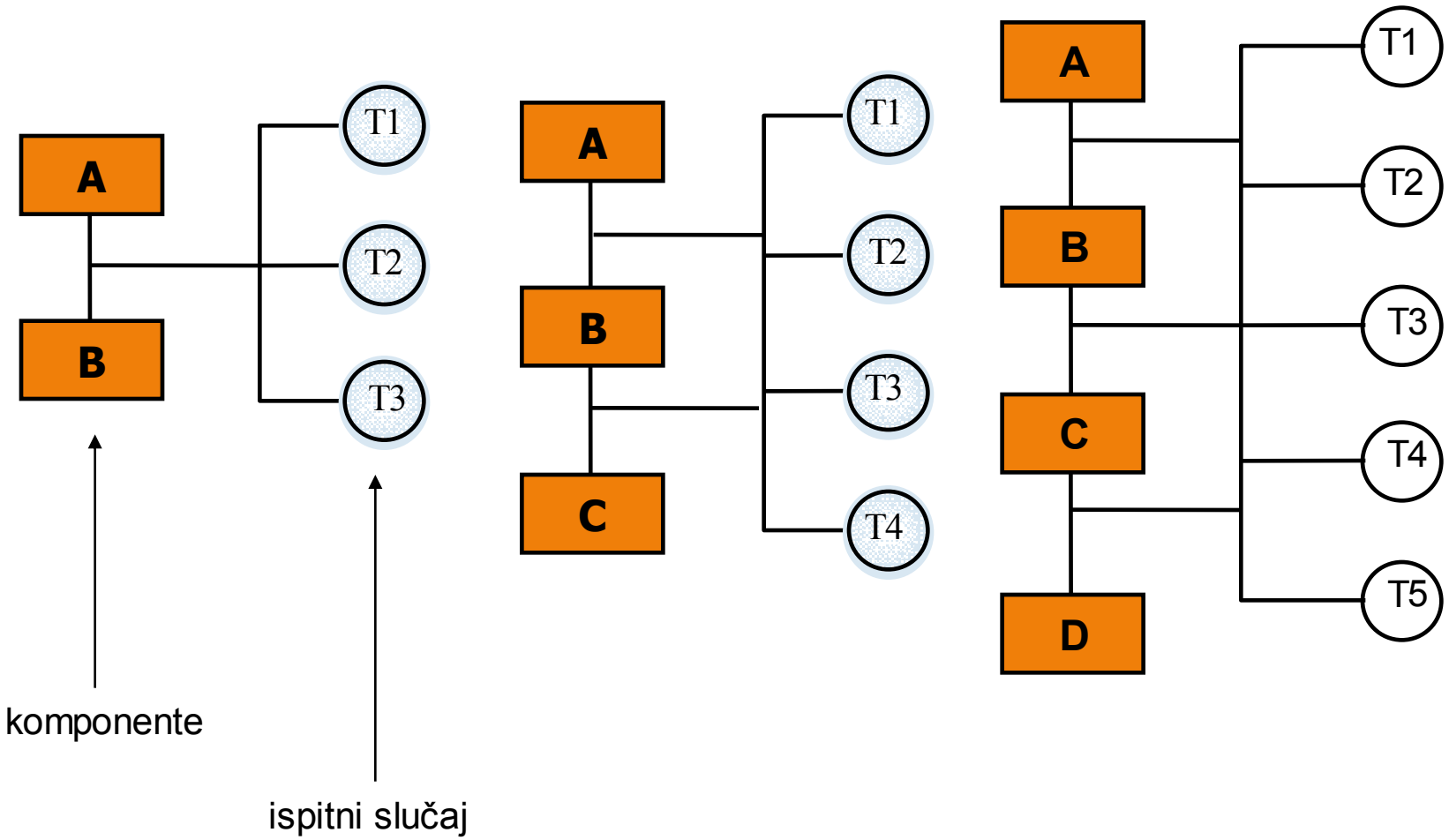
Integracijsko ispitivanje

- Proces verifikacije interakcije programskih komponenti
  - s dodatnim kodom za zajednički rad
- Sistem se ispita tijekom integracije komponenti s ciljem uočavanja problema integracije
- Cilj je osigurati zajednički rad grupe komponenti prema specifikaciji dokumentaciji zahtjeva
- Osnovni problem predstavlja lokalizacija pogrešaka
  - zbog složenih interakcija

Pristupi integracijskom ispitivanju

- ***Veliki prasak*** (*engl. Bing bang*)
  - integrirati sve komponente bez prethodnog ispitivanja
  - u slučaju pogrešaka problem predstavlja otkrivanje mjesta pogreške
- ***Poboljšani veliki prasak*** (*engl. Enhanced bing bang*)
  - integracija svih komponenti nakon provedenog ispitivanja
  - u slučaju pogrešaka i dalje prisutan problem otkrivanja mjesta
- ***Inkrementalni pristup***
  - integracija i ispitivanje sustava dio po dio
  - uobičajen pristup u praksi
  - efikasan u lokalizaciji mjesta pogreške

Inkrementalno integracijsko ispitivanje

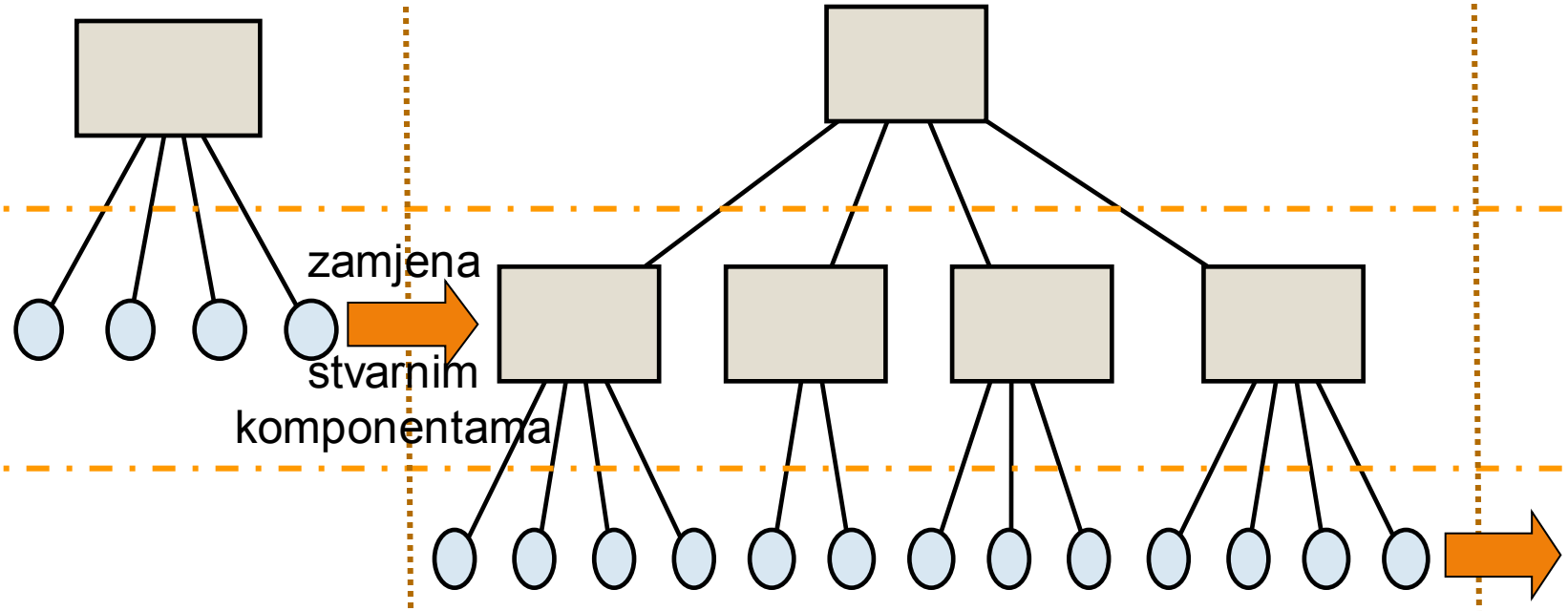


- Kako pristupiti ispitivanju?

Izvor: Sommerville, I., Software engineering

Inkrementalno integracijsko ispitivanje

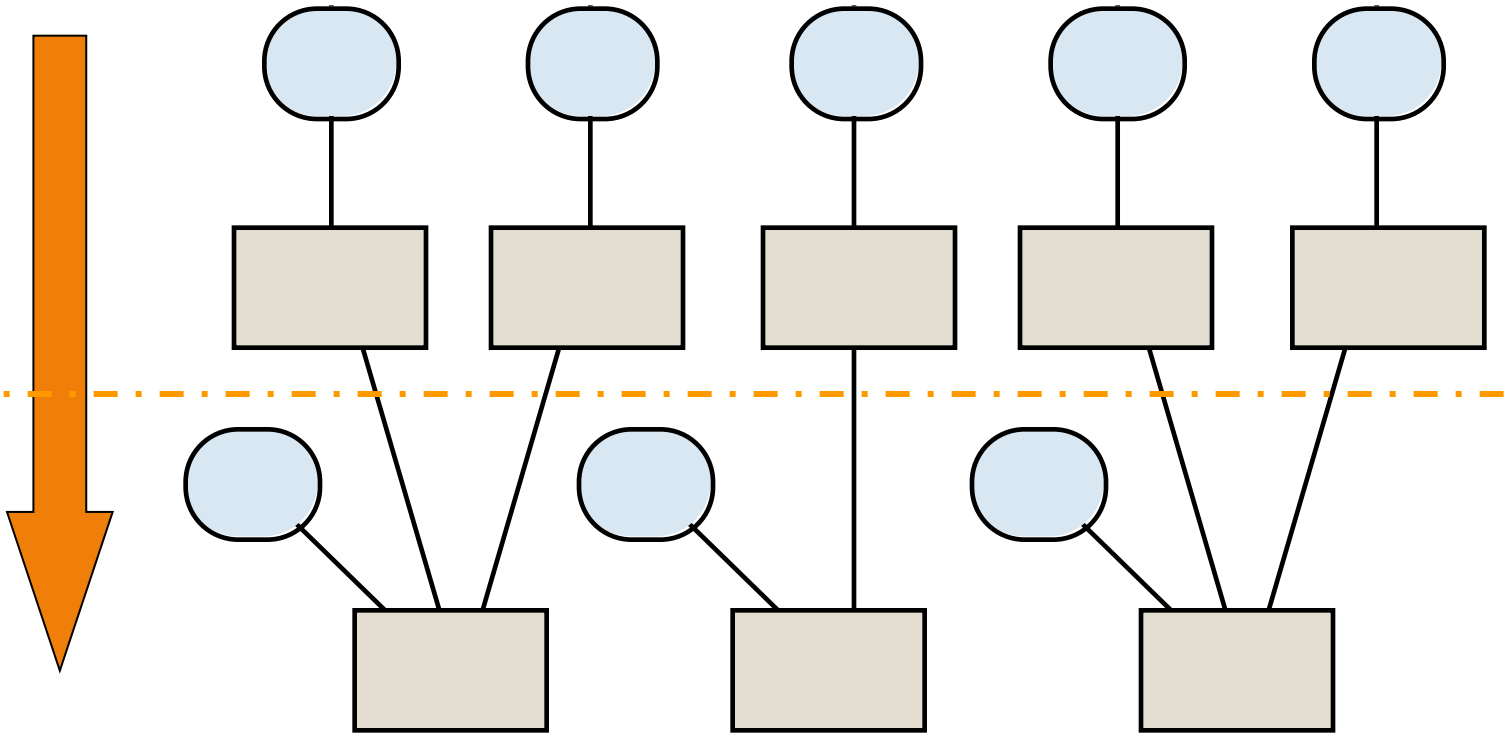
- Odozgo na dolje (*engl. Top down integration*)
  - Razviti kostur sustava i popuniti ga komponentama
  - ne treba razvijati upravljačke programe



Izvor: Sommerville, I., Software engineering

Inkrementalno integracijsko ispitivanje

- Odozdo na gore (*engl. Bottom up integration*)
  - prvo integrirati neke komponente (najvažnije i najčešće funkcionalnosti), te dodati preostale
  - ne treba razvijati prividne komponente



Izvor: Sommerville, I., Software engineering

Inkrementalno integracijsko ispitivanje

- Funkcijska integracija (*engl. Sandwich testing*)
  - integriranje komponenti u konzistentne funkcije bez obzira na hijerarhijsku strukturu
  - kombinacija odozgo-prema-dolje i odozdo-prema-gore
  - najčešći postupak u praksi

Integracijsko ispitivanje u OO

- *engl. Inter-class testing*
- U objektno orijentiranim sustavima pogodno je iskoristiti prepoznata grupiranja razreda
  - međuosvisne klase su dobri kandidati za početak inkrementalne integracije (*engl. tightly coupled*).
- Kandidati:
  - razredi grupirane u pakete
  - razredi u hijerarhijskoj ovisnosti
  - razredi povezani dijagramima interakcija pridruženim obrascima uporabe
- Najčešći pristupi
  - pristup odozdo prema gore
  - primjena obrazaca uporabe

Strategije integracijskog ispitivanja OO

- *engl. Inter-class testing*
- Integracijsko ispitivanje razreda može se grupirati prema razinama apstrakcije
  - ispitivanje zasnovano na dretvama
    - *engl. thread-based testing*
    - integrira skup razreda obzirom na poticaje ulaza ili događaja
  - ispitivanje zasnovano na uporabi
    - *engl. use-based testing*
    - integrira skup razreda koje zahtijeva obrazac uporabe
  - ispitivanje zasnovano na grupiranju
    - *engl. cluster testing*
    - integrira skup razreda koje su neophodni za ostvarenje suradnje razreda

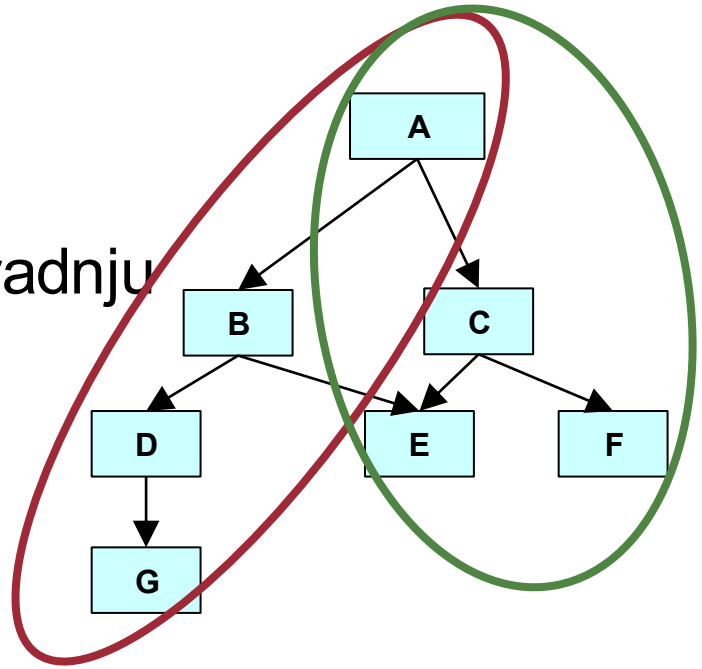
Tijek ispitivanja

1. Uobičajeni postupak je da se za svaki podrazred, koristiti popis operatora za generiranje niza slučajnih ispitnih sekvenci.
   - operatori šalju poruke drugim razredima
2. Za svaku poruku koja se generira, određuje se pozvani (suradnički) razred i odgovarajući operator u poslužitelja objekta.
3. Za svaki operator u pozvanom objektu razreda utvrditi nove poruke koje odašilje.
4. Za svaku od poruka, odrediti sljedeću razinu koja se poziva
   - izraditi odgovarajuće ispitne slučajeve

Pristupi integracijskom ispitivanju OO

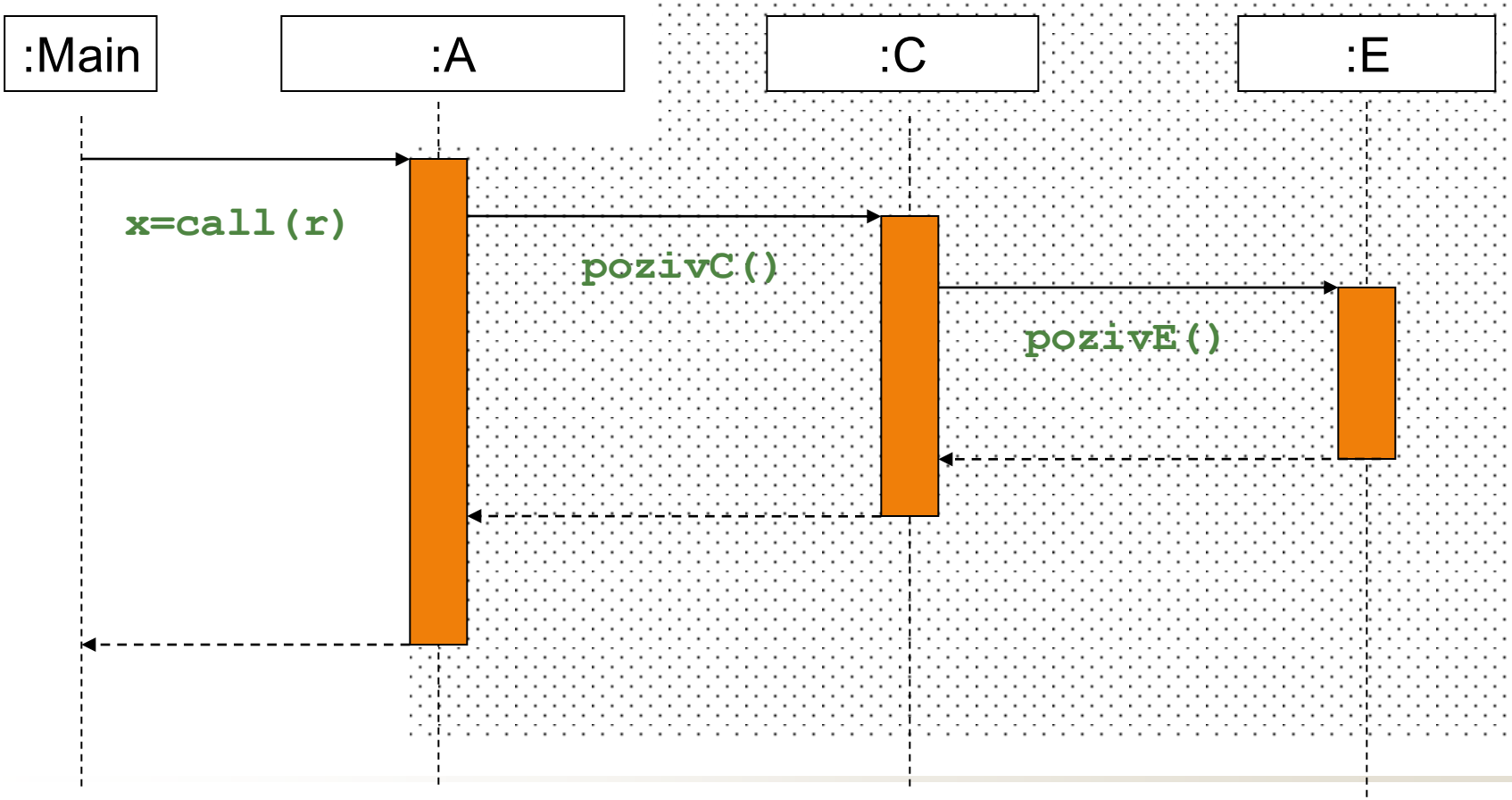
- Pristup odozdo prema gore
  - najčešća tehnika
  - integracija razreda započevši od krajnjih podrazreda prema hijerarhiji ovisnosti

- Primjena obrazaca uporabe
  - obrasci opisuju interakciju razreda
  1. u hijerarhiji razreda identificirati suradnju
  2. odabrati kriterij sekvenci ispitivanja
  3. oblikovati ispitivanje



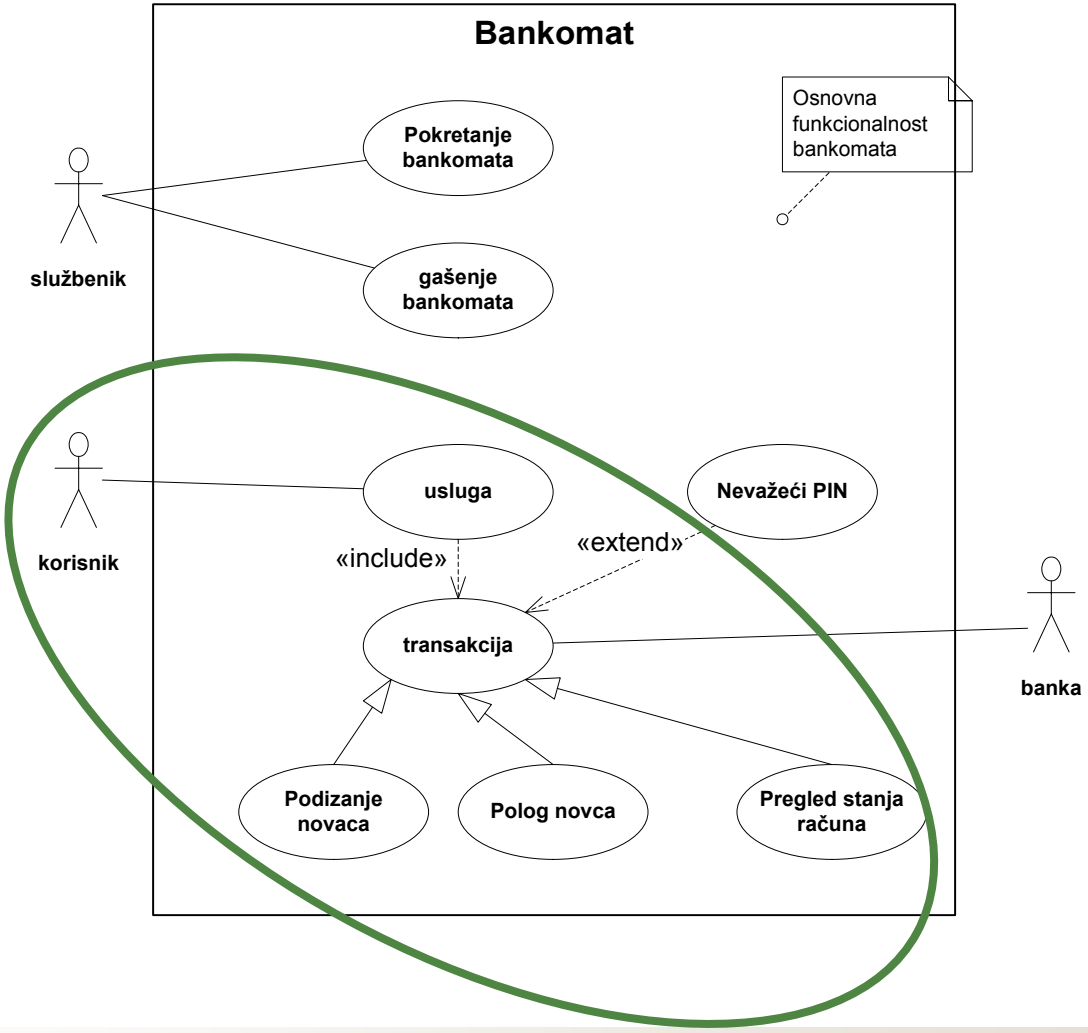
Primjer: Uporaba dijagrama interakcija

- Ispitivanje metode **call** s ulazom **r** i očekivanim rezultatom **x**



Primjer: Obrasci uporabe

- Bankomat:



Prednosti i nedostaci

- **Ispitivanje odozdo prema gore**
- Omogućuje rano ispitivanje
- Moduli mogu biti integrirana u različitim grupama
- Glavni naglasak je na funkcionalnosti i performansama modula
- Nije potrebno izgrađivati prividne module
- Lagana organizacija provođenja
- Pogreške u kritičnim modula nalaze rano

- Nedostaci
  - potrebna izgradnja upravljačkih programa
  - dugotrajan proces u slučaju složenih hijerarhija
  - kasno otkrivanje pogreške u sučeljima (kritično)

- **Ispitivanje odozgo prema dolje**
- Upravljački (glavni) program je ispitan prvi
- Moduli su integrirani istodobno
- Glavni naglasak je postavljen na ispitivanje sučelja
- Nije potrebno izgrađivati upravljačke programe
- brzo se dobiva radni prototip
  - rano otkrivanje pogrešaka u sučeljima (kritično)
- Sučelje pogreške otkrio rano
- Modularnost pospješuje ispravljanje pogrešaka
- Nedostaci
  - potrebno je izgrađivati prividne module
  - sporo uključivanje većeg tima
  - pogreške u kritičnim modula na niskim razinama nalaze se kasno

ISPITIVANJE SUSTAVA

Ispitivanje sustava

- *engl. System (Release) testing*
- Proces ispitivanja inačice namijenjene distribuciji korisniku
- Osnovni cilj provjera podudarnosti sustava s ***funkcijskim*** zahtjevima
- Uobičajeno okruženje funkcijskog ispitivanja
  - ispitivači nemaju znanje o detaljima implementacije
  - veća vjerojatnost pronalaženja kvarova koje nisu uočili programeri
  - neovisni ispitivači uklanjaju sukob interesa razvoja i ispitivanja
- Dvije faze
  - integracijsko ispitivanje – ispitivači imaju pristup izvornom kodu
  - ispitivanje gotovog sustava - ispitivači vide sustav kao crnu kutiju

Ispitivanje performansi

- Prije isporuke potrebno je ispitati ***svojstva sustava kao cjeline u radu*** (*engl. emergent properties – svojstvo sustava koje proizlazi od interakcije sastavnih dijelova*)
- Različite namjene:
  - standardno ispitivanje performansi (*engl. benchmarks*)
  - pokazati zadovoljenje performansi pod različitim radnim opterećenjima
  - pokazati proširivost sustava
  - ispitivanje se planira s povećanjem opterećenja sve dok performanse ne postanu nezadovoljavajuće
- ***Ispitivanje pod pritiskom*** (*engl. Stress testing*)
  - namjena određivanje stabilnosti sustava
    - Robusnost, raspoloživost i obrada pogrešaka na graničnim vrijednostima opterećenja npr. broja podataka, učestalost zahtjeva, veličina i sl.
    - npr. transakcija u sekundi, u raspodijeljenim sustavima broj korisnika, DoS napadi,...
  - istjerivanje pogrešaka

Ispitivanje prihvatljivosti

- *engl. Acceptance testing*
- Provjerava ponašanje sustava u odnosu na zahtjeve naručitelja (verifikacija je značajnija)
- Najčešće se provodi zajednički s timom naručitelja
- Provodi se kao funkcijsko ispitivanje (ispitivanje crne kutije)
- Uobičajeno se provodi prije isporuke programa
  - moguće i nakon isporuke prema dogovoru s kupcem
- Cilj je pokazati da sustav zadovoljava zahtjeve i spreman je za uporabu

Instalacijsko, α i β ispitivanje

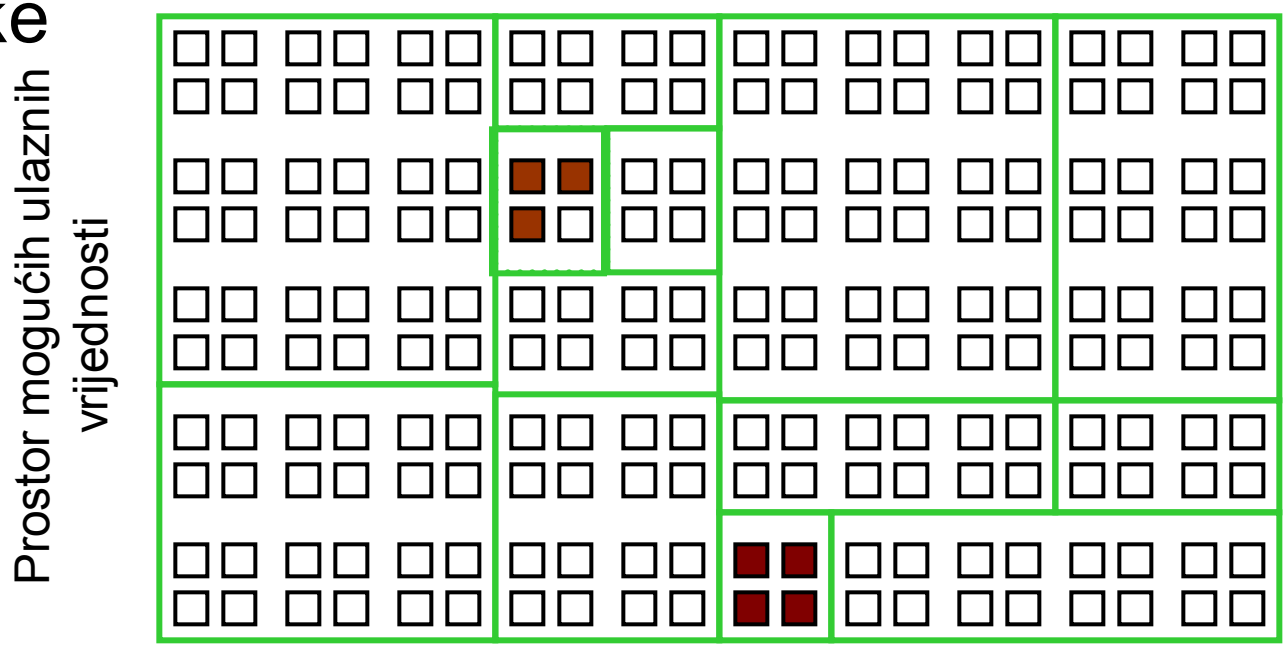
- ***Instalacijsko ispitivanje***
  - provodi se nakon ispitivanja prihvatljivosti na instalaciji u radnoj okolini
  - identično ispitivanju sustava prema zahtjevima sklopovske konfiguracije
- ***Alfa / Beta ispitivanje***
  - prije završne isporuke, program pokusno upotrebljava ciljana skupina korisnika
    - Alfa – unutar tvrtke, konzorcija – interno uz prisustvo razvojnog tima
    - Beta – vanjski korisnici

Strategije ispitivanja

- ***Iscrpno ispitivanje*** (*engl. Exhaustive testing*)
  - ispitivanje svih mogućih scenarija
  - moguće samo za ograničene primjere
- ***Slučajno ispitivanje*** (*engl. Random testing*)
  - nije isto kao i “ad hoc” ispitivanje
  - odabir ispitnih slučajeva temeljem vjerojatnosti
    - Uniformna razdioba
    - Razdioba temeljena na prethodno prikupljenim podacima
- ***Sistematsko ispitivanje***
  - podjela ulaznih podataka u poddomene (particije) i odabir ispitnih slučajeva temeljem različitih svojstava
    - svojstva koda, specifikacija, rizik, ...

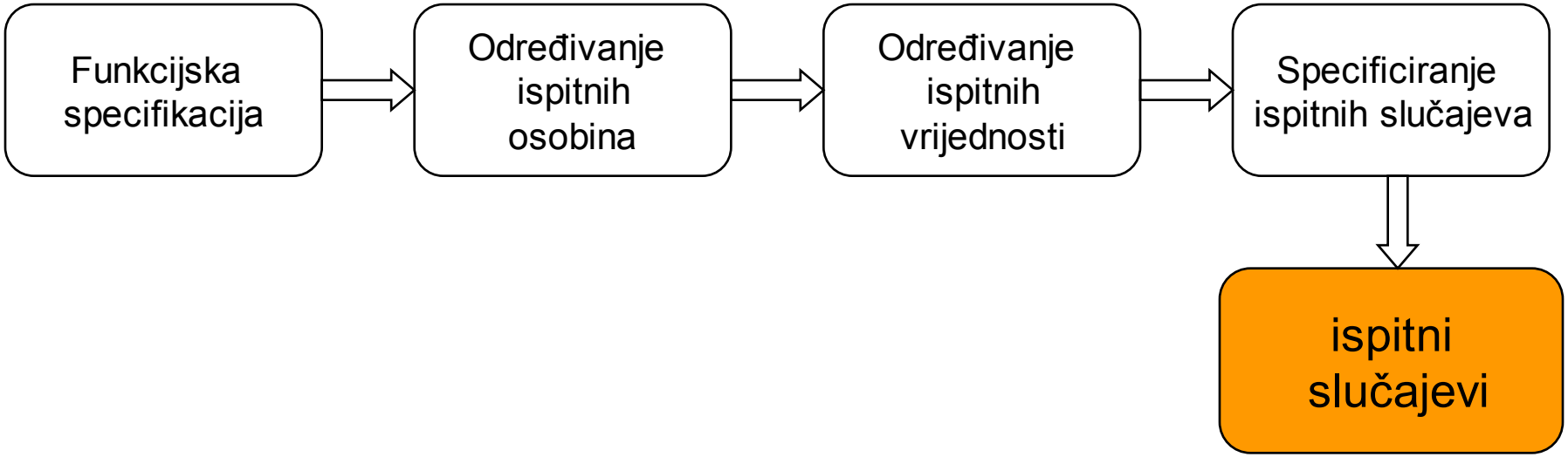
Sistematsko ispitivanje particija

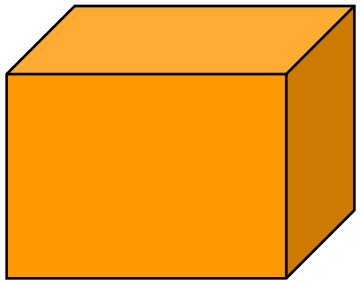
- Uobičajeno 30-85 pogrešaka u 1000 LOC
- Dobro ispitani programi 0.5 – 3 pogreške
- Pogreške su često grupirane
- Cilj je izolirati područja s vjerojatnom pojavom pogreške



Sistematsko ispitivanje

- Osnovni koraci:





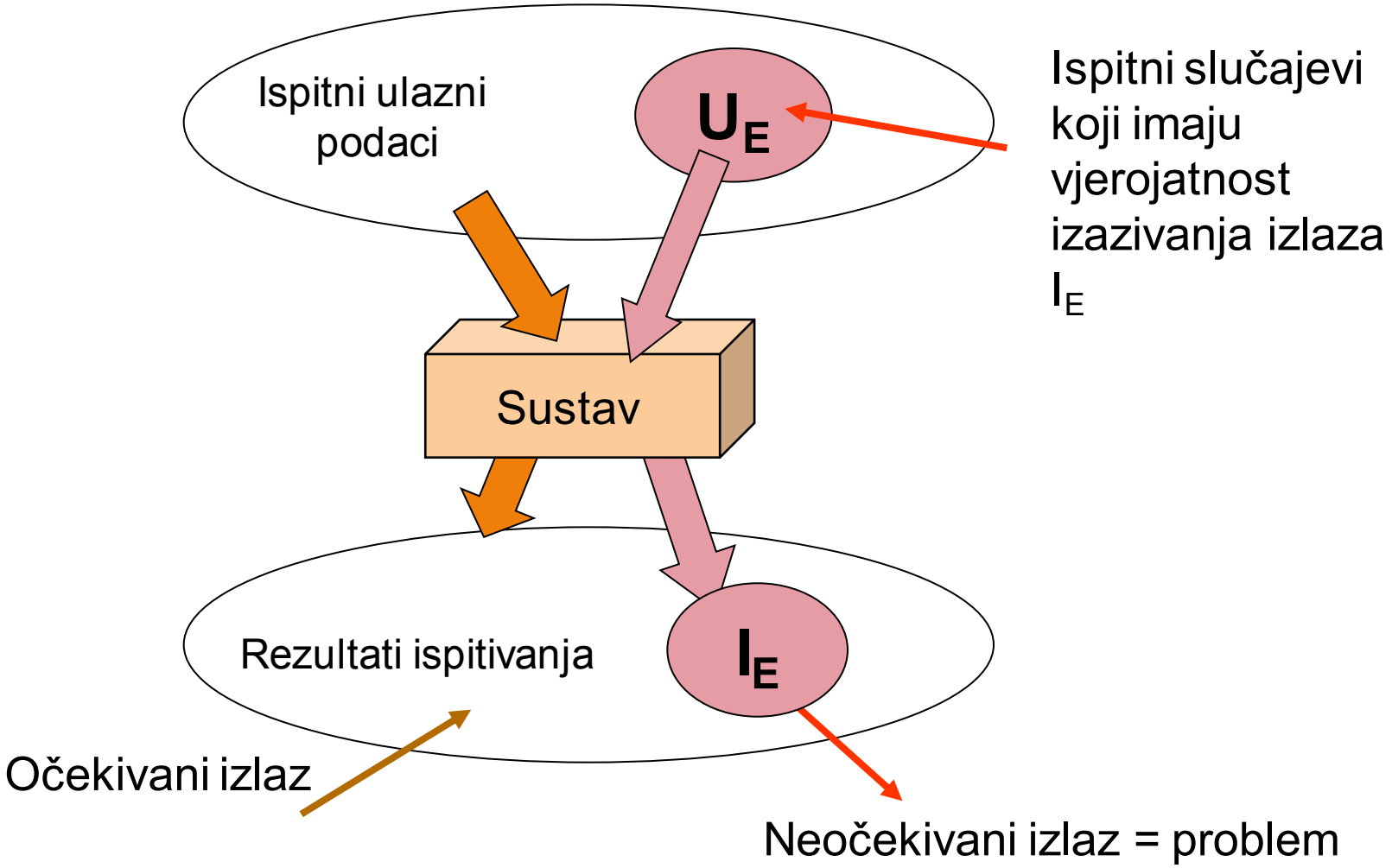
Tehnike ispitivanja

FUNKCIJSKO ISPITIVANJE

Funkcijsko ispitivanje

- *engl. Black box testing*
- Pretpostavlja da *nema znanja programskog koda* ili oblikovanja sustava
  - koncentracija na U/I ponašanje
  - ispitivanje samo prema zahtjevima i specifikacijama
- Pretpostavka je da za ulazne podatke možemo predvidjeti izlaz
  - gotovo nemoguće generirati sve moguće ulaze!
- Cilj: Smanjiti broj ispitnih slučajeva ekvivalentnom podjelom ulaza i analizom graničnih vrijednosti
- Oblikovanje ispitnih slučajeva (*engl. test case design*)
  - zasnovano na specifikaciji sustava
  - podjela vrijednosti ulaznih podataka
  - podjela ulaznih podataka u klase
  - odabir ispitnih slučajeva za svaku klasu
  - analiza graničnih vrijednosti

Funkcijsko ispitivanje



Ispitivanje particija

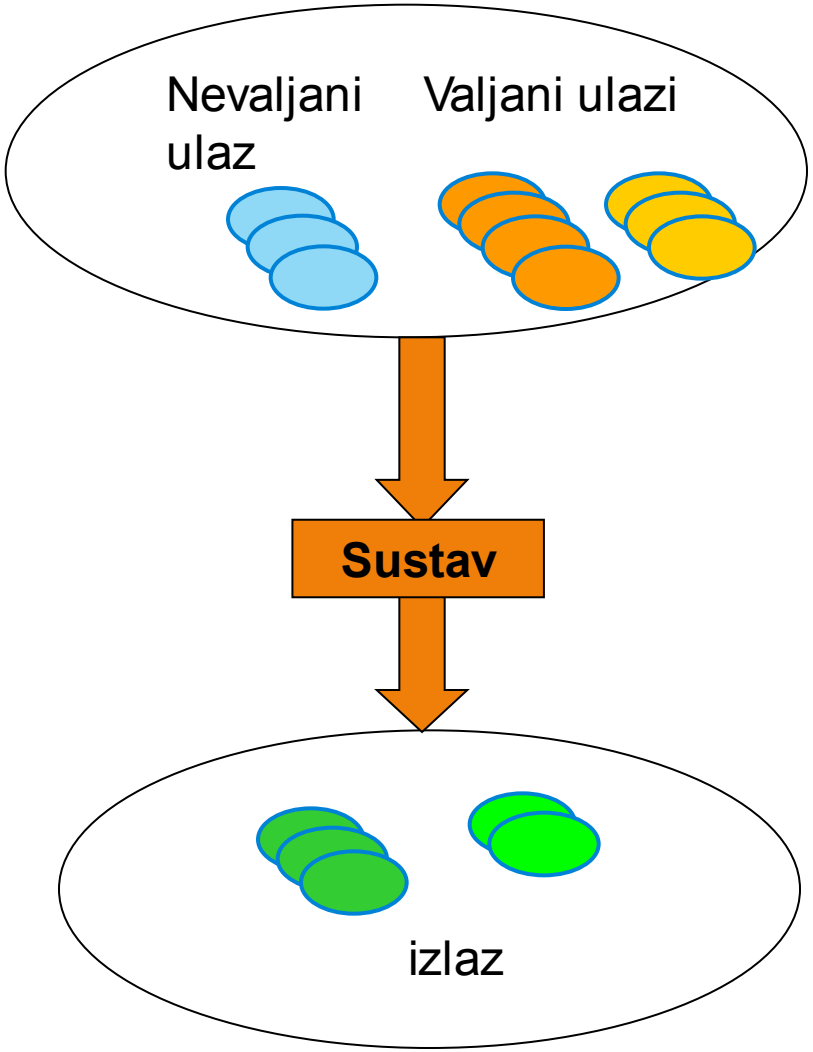
- Ulazni podaci i rezultati često se mogu grupirati u različite klase u kojima je ponašanje članova slično (princip: *Podijeli i vladaj).*
- Svaka od tih klasa predstavlja ekvivalentnu particiju (*engl. equivalence partition*) u kojoj se program ponaša na isti način za sve njegove članove

PRETPOSTAVKA

- Ispitni slučajevi moraju pokrivati sve ekvivalentne particije
  - posebice granične vrijednosti svake particije jer su to najčešći uzroci kvara
- Odabir ekvivalentnih particija za ispitivanje
  - podijeliti ulazne ekvivalentne particije na:
    - Valjana vrijednost
    - Nevaljana vrijednost
  - vrijednosti ulaza su valjane u intervalu –odabrati 3 ispitna slučaja
    - Vrijednost ispod intervala
    - Vrijednost u intervalu
    - Vrijednost iznad intervala

Ekvivalentne podjele

- Ekvivalentne podjele ulaza:
  - Valjani ulazi
  - Nevaljani ulazi
- Izlazne ekvivalentne podjele
  - zajedničke karakteristike



Koraci ispitivanja

1. Odredi particije za sve ulazne varijable.
2. Za sve particije odaberi vrijednosti ispitivanja.
3. Definiraj ispitne slučajeve koristeći odabrane vrijednosti.
4. Odredi očekivane izlaze za odabrane ispitne slučajeve i provedi ispitivanje.

Primjer funkcijskog ispitivanja

- Zadatak: *Ispitati program za odlučivanje o pravu na glasovanje.* Pravo glasovanja ovisi o starosti osobe, minimalna dob je 18 godina.
  - jedna varijabla - dob
  - svi ulazni podaci mogu se podijeliti u dvije ekvivalentne podjele
    - 0-17 - ne smije glasovati
    - ≥ 18 - smije glasovati
  - ispitni slučaj 1
    - Odabir reprezentativnih podataka iz svake podjele
  - ispitni slučaj 2
    - Odabir podataka na granicama podjela

Ispitni slučaj 1

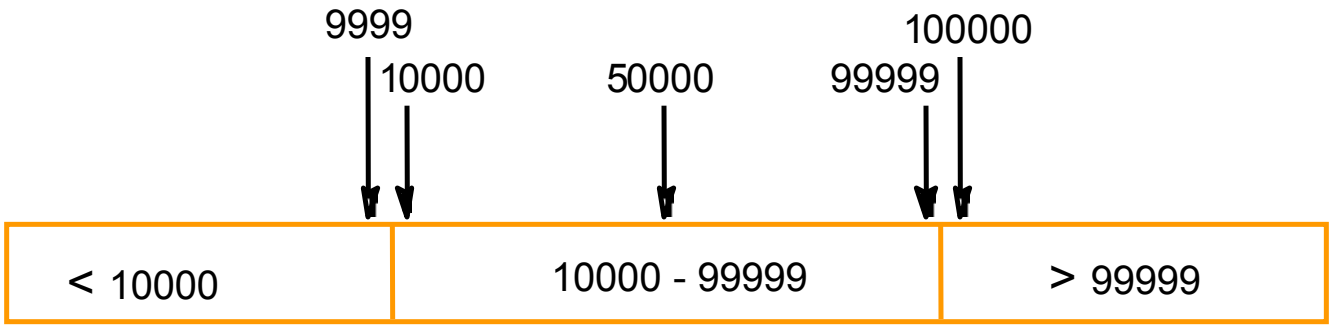
Test	Podatak	Rezultat
1	12	NE
2	21	DA

Ispitni slučaj 2

Test	Podatak	Rezultat
3	17	NE
4	18	DA

Ekvivalentne podjele

- Primjer: Ulaz 5-znamenkasti broj vrijednosti između 10000 i 99999
  - <10,000
  - 10,000-99,999
  - >99,999



Primjer: komponenta za pretraživanje

- Pretražuje se sekvencija (niz) elemenata za dani element (ključ - key) zadana slijedećom apstraktnom specifikacijom (ne implementacijom):

```
Public static void trazi (int [] element, int kljuc,
                                  bool nadjen, int indeks)
```

Preduvjet - istinit prije poziva komponente

- rutina radi (pretražuje) samo za neprazne sekvencije - ulazni niz ima najmanje jedan element

```
element'first <= element'last
```

Rezultat - istinit nakon izvođenja komponente

- varijabla **nadjen** = T ako je traženi element u nizu (na mjestu L):

```
Nadjen and element(L) = kljuc
```

Ili

- *traženi element se ne nalazi u nizu:*

```
Not nadjen and not (exists i, element'first >= i <=
                    element'last, element (i) = kljuc)
```

Primjer: podjela particija

- Podjela particija temeljem specifikacije:
  - ulazi koji zadovoljavaju preduvjete
    - Ulazi kod kojih je ključ element zadanog niza
    - Ulazi kod kojih je ključ nije element zadanog niza
  - ulazi koji ne zadovoljavaju preduvjete
    - Ulazi kod kojih je ključ element zadanog niza
    - Ulazi kod kojih je ključ nije element zadanog niza
- Naputak za provedbu ispitivanja:
  - ispitaj pretraživanje na slijedećim nizovima:
  - niz je jedan broj (programeri obično pretpostavljaju više)
  - niz ima paran broj vrijednosti
  - niz ima neparan broj vrijednosti

- Preporučeni slijed ispitivanja

1. Ispitati jednostruke vrijednosti
2. Sekvence (nizove) različitih duljina neophodno ispitati zasebno
3. Ispitne slučajeve oblikovati na taj način da obuhvaćaju prvi, srednji i zadnji element sekvence (niza)
4. Ispitati sekvence (nizove) duljine 0

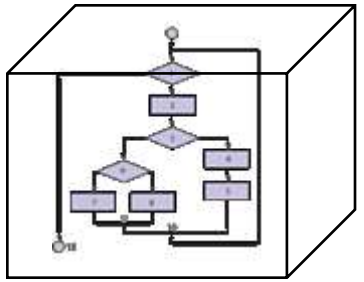
Primjer:

element (ulazni niz)	kljuc	nadjen, L=mjesto	Opis
17	17	T,1	1 element niza Traženi element je u nizu
17	0	F,?	1 element niza Traženi element nije u nizu
17,21,23,29	17	T,1	Više elementa niza Traženi element prvi u nizu
9,16,18,30,31,41,45	45	T,1	Više elementa niza Traženi element zadnji u nizu
17,18,21,23,29,38,41	23	T,4	Više elementa niza Traženi element u sredini niza
17,18,21,23,29,38,41	21	T,3	Više elementa niza Traženi element u sredini niza
12,18,21,23,32	23	T,4	Više elementa niza Traženi element u sredini niza
21,23,29,33,38	25	F,?	Više elementa niza Traženi element nije u nizu

Funkcijsko ispitivanje - zaključci

- Izražen problem kombinatorne eksplozije ispitnih slučajeva
  - posebice izraženo kod potrebe ispitivanja valjanih i nevažećih podataka
- Često nije jasno da li su odabrani ispitni slučajevi dobro oblikovani za otkrivanje ciljane pogreške
  - nepoznavanje unutarnje strukture!

- Jednostavno za uporabu
- Brzi razvoj ispitnih slučajeva
- Ispitni slučajevi se rade iz perspektive korisnika
- Neovisno o jeziku implementacije

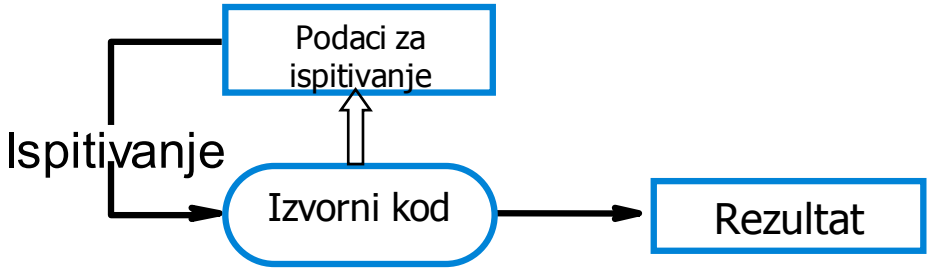


Tehnike ispitivanja

STRUKTURNO ISPITIVANJE

Strukturno ispitivanje

- *engl. Structural testing, White Box testing*
- Ispitivanje očekivanog ponašanja zasnovano na svojstvima programa ili oblikovanju (tj. strukturi programa)



- Cilj ispitivanja je pokrivanje izvođenja svih mogućih naredbi i uvjeta programa najmanje jednom
- Primjer:
  - ispitivanje komponenti, strukturno ispitivanje *(engl. statement coverage, branch coverage, unit testing, i sl.)*
- Oblikovanje ispitnih slučajeva (*engl. test case design*)
  - zasnovano na strukturi programa (poznamo strukturu)

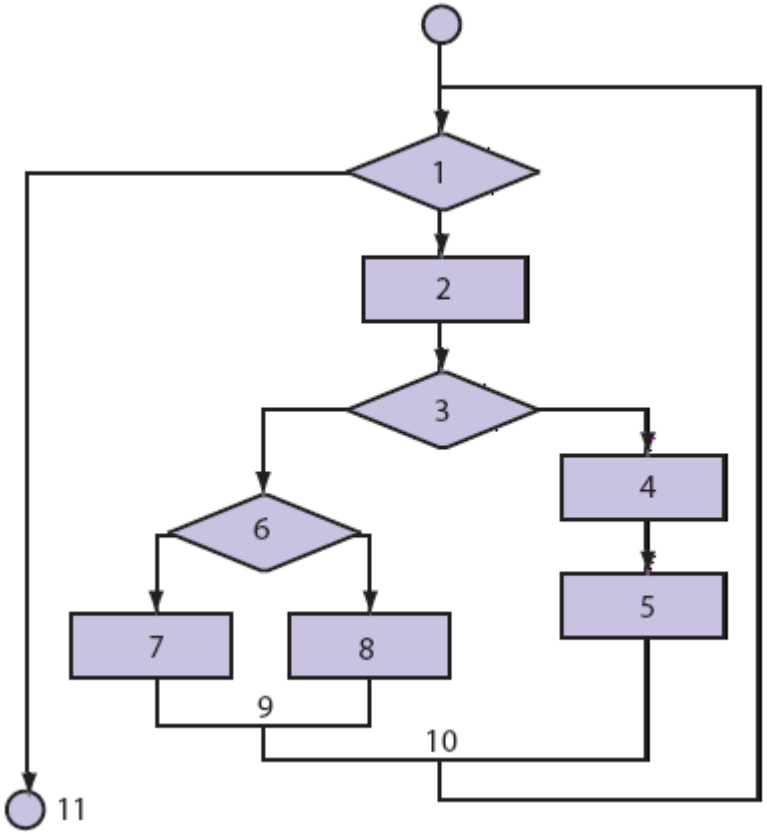
Prikaz programa

- apstrakcija programskog koda
- Graf tijeka programa
  - *engl. control flow graph*
  - Graf. reprezentacije tijeka programa
- Čvorovi – procesi ◯, programske odluke ◯ ili ◇
- Lukovi – kontrola tijeka

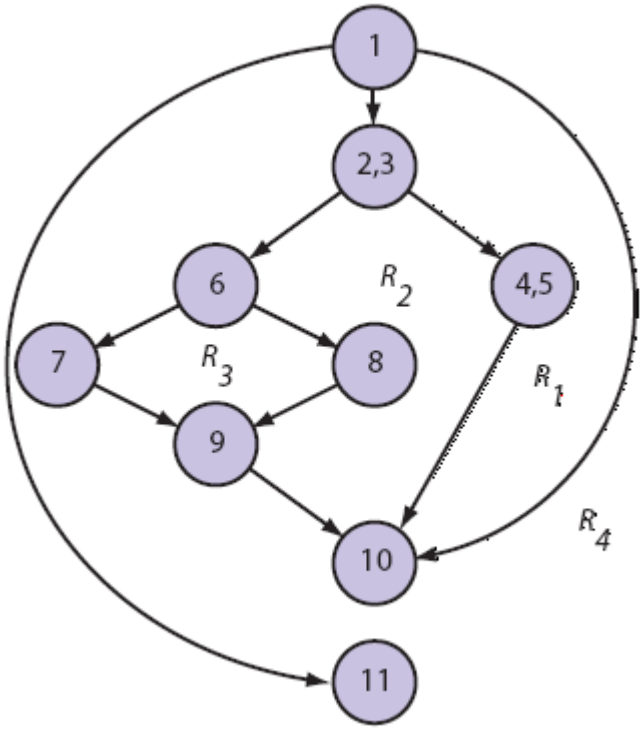
```
public roots (double a, b, c)
{
double q = b * b - 4 * a * c;
if (q > 0 && a ≠ 0) {
        …
        …
    }
    else if (q == 0) {
      x = (-b) / (2 * a);
    }
    else {
        …
    }
    else {
        …
        …
    }
}
```

Primjer: Dijagram toka i grafa tijeka programa

- Dijagram toka
  - *engl. flowchart*



- Graf tijeka programa



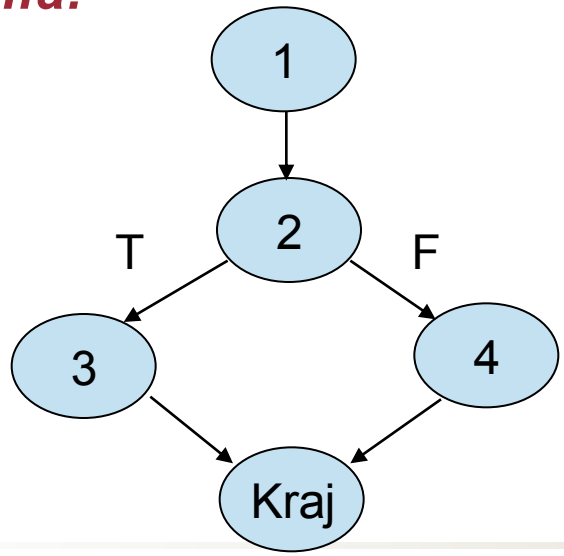
Primjer strukturnog ispitivanja

- Provjera prava glasovanja

```
        učitaj (dob );
        ako (dob>= 18) tada {
            ispiši (“smije glasovati”);
        }
        inače {
            ispiši (“ne smije glasovati”);
        }
```

1, 2, 3, 4

- ***Graf tijeka programa:***

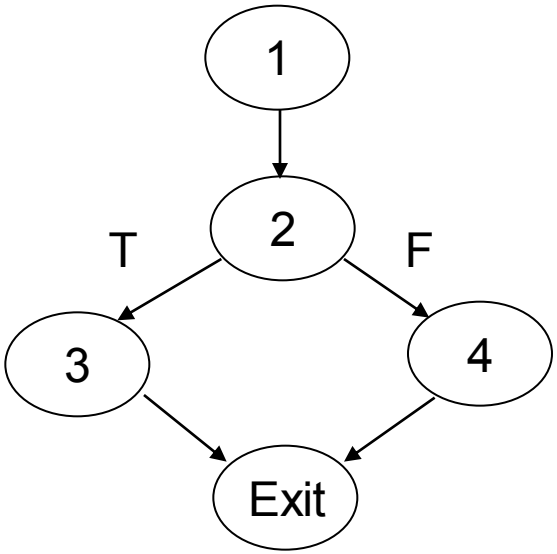


Mogući putovi:

1. $1 \rightarrow 2 \rightarrow 3 \rightarrow$ kraj
2. $1 \rightarrow 2 \rightarrow 4 \rightarrow$ kraj

Primjer strukturnog ispitivanja

Graf tijeka programa



Analiza putova

Mogući putovi:

1. $1 \rightarrow 2 \rightarrow 3 \rightarrow$ kraj
2. $1 \rightarrow 2 \rightarrow 4 \rightarrow$ kraj

Ispitni slučaj 1: osigurati izvođenje oba puta tijekom ispitivanja

Test	Podatak	Rezultat
1	12	NE
2	21	DA

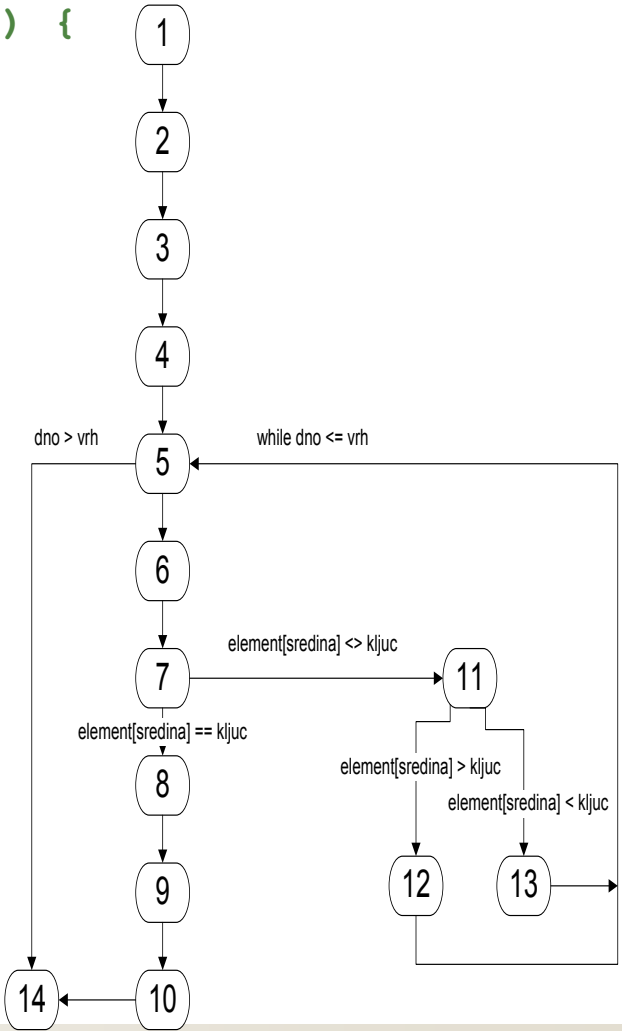
Ispitni slučaj 2: osigurati ispravan rad *if* naredbe na graničnim vrijednostima

Test	Podatak	Rezultat
3	17	NE
4	18	DA

Primjer: binarno pretraživanje

- Ulaz uređen niz elemenata i ključ, izlaz oznaka nadjen i indeks elementa u nizu

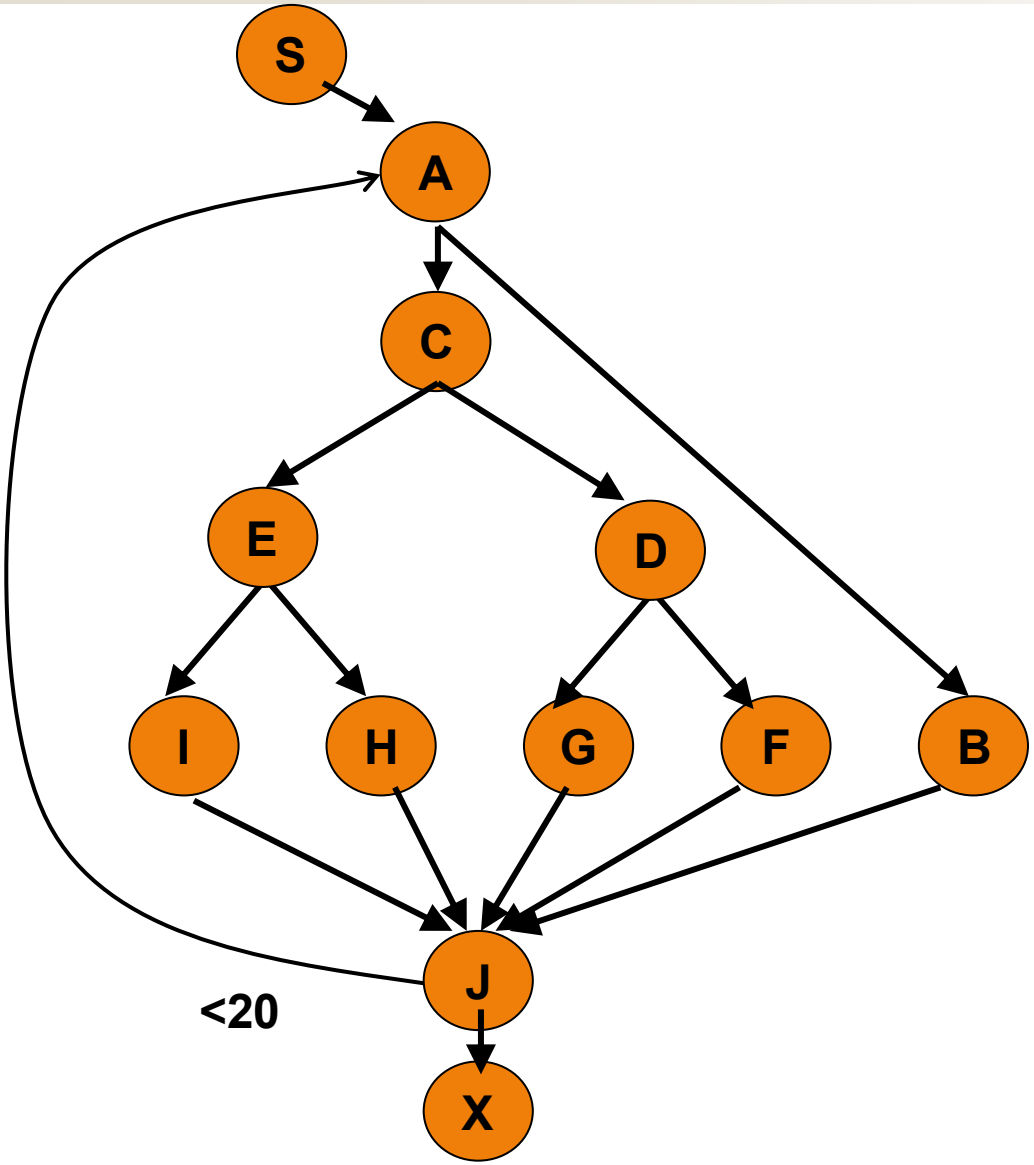
```
class Pretraga {
public static void trazi (int [] element, int kljuc,
                          bool nadjen, int indeks) {
 int dno = 0 ;                                    (1)
 int vrh = element.length - 1 ;                   (2)
 int sredina ;
 nadjen = false ;                                 (3)
 indeks = -1 ;                                    (4)
 while ( dno <= vrh )                             (5)
 {
  sredina = (vrh + dno) / 2 ;                     (6)
  if (element [sredina] == kljuc)                 (7)
  {
    indeks = sredina ;                            (8)
    nadjen = true ;                               (9)
    return ;                                      (10) (kraj!)
  }
  else {
    if (element[sredina] < kljuc)                 (11)
        dno = sredina + 1 ;                       (12)
    else
    vrh = sredina - 1 ;                           (13)
  }
 } //while
 } // void trazi                                   (14) (kraj!)
} // class Pretraga
```



Analiza putova

- Problem predstavlja postojanje programskih petlji
  - veliki broj putova
  - putovi koji prolaze kroz petlju više puta su ekvivalentni
  - uobičajeno velik broj mogućih kombinacija
- Ispituju se:
  - svi neovisni putovi
  - logički izrazi (T,F)
  - petlje (rubovi, sredina)
  - interni podaci - struktura
- Izvođenje ispitnog slučaja
  - osigurava prolaz kroz određeni put

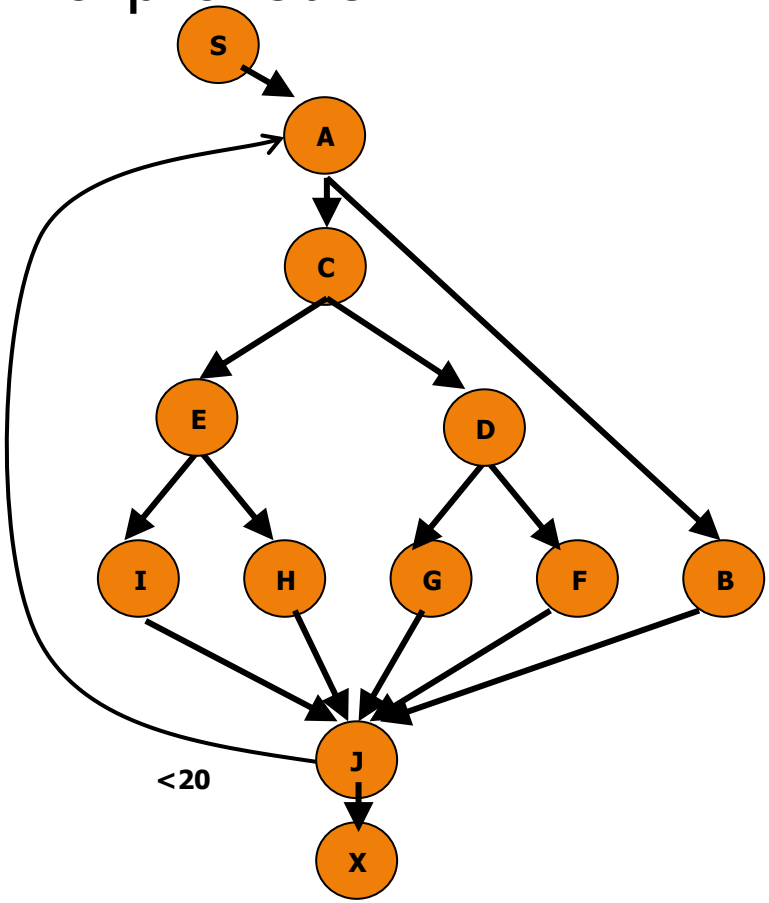
Primjer sekvenci



- Putovi od S ⇒X : 5
- Cilj ispitati petlju od 1 do 20 puta sa **SVIM** kombinacijama putova u ponavljanju petlje
  - 5×5×5... ×5
  - $5^1 + 5^2 + ... + 5^{19} + 5^{20} = 10^{14}$
- Iscrpno ispitivanje
- *engl. exhaustive testing*
- Ispituje sve moguće kombinacije
  - da li je ostvarivo?
- ***Trajanje??***
  - ako ispitni slučaj traje1 ms
    - ⇒ 3170 godina

Selektivno ispitivanje

- Ispitivanje svih kombinacija je najčešće nemoguće
- **Pokrivanje koda (*engl. Code coverage*)** predstavlja pokrivenost izvornog koda programa provedenim ispitivanjima
- Suzujemo prostor ispitivanja na:
  - ispitivanje temeljnih putova
    - *engl. Basis path testing*
  - ispitivanje uvjeta
    - *engl. Condition testing*
  - ispitivanje petlji
    - *engl. Loop testing*
  - ispitivanje protoka podataka
    - *engl. Dataflow testing*



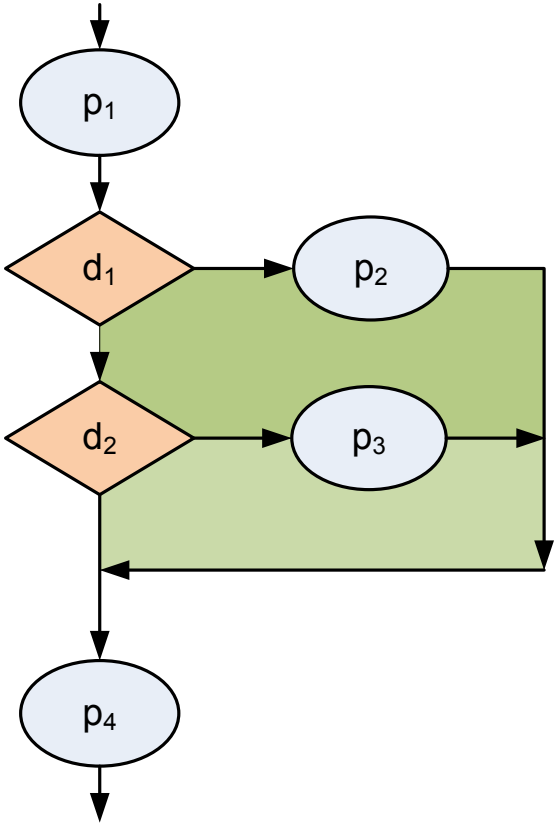
Ispitivanje temeljnih putova

- Temeljni skup (*engl. basis set*)
  - Skup putova koji minimalno jednom pokrivaju izvođenje svih naredbi i uvjeta
    - Ne mora biti jednoznačan
- Teorija grafova – izračun broja linearno neovisnih putova
  - CV(G) - Ciklomatička složenost *(engl. Cyclomatic complexity)*
  - Broj neovisnih putova u temeljnom skupu
- Primjena za mjerenje količine logike odlučivanja u programskom modulu – 1974.g. McCabe
  - CV(G) = Lukovi – Čvorovi + 2*P
    - Lukovi = broj lukova u grafu
    - Čvorovi = Broj čvorova u grafu
    - P = Broj povezanih komponenti (potprograma, prekidnih rutina i sl.)
  - gornja granica broja ispitnih slučajeva koja garantira potpuno pokrivanje svih naredbi programa

Ciljevi pokrivanja

- Ispitivanje nema za cilj povećati postotak pokrivanja
- Pokrivanje nam govori u kojoj smo fazi ispitivanja
  - veći postotak ne dokazuje da u programu nema kvarova
- Nikada se ne traže načini povećanja pokrivanja samo radi pokrivanja
  - često se može postići relativno jednostavnim ispitivanija za koje znamo da nemaju veliku šansu otkrivanja kvara
  - u nekim slučajevima ipak može biti dio zahtjeva
  - npr. ako postoji kvar koji izaziva zatajenje pri svakom izvođenju nema smisla ponavljati ispitivanja
- Za svako pokrivanje mora postojati obrazloženje
  - tj. obrazložiti otkrivanje kojih kvarova se propušta nepokrivanjem

Primjer: Izračun ciklomatičke složenosti



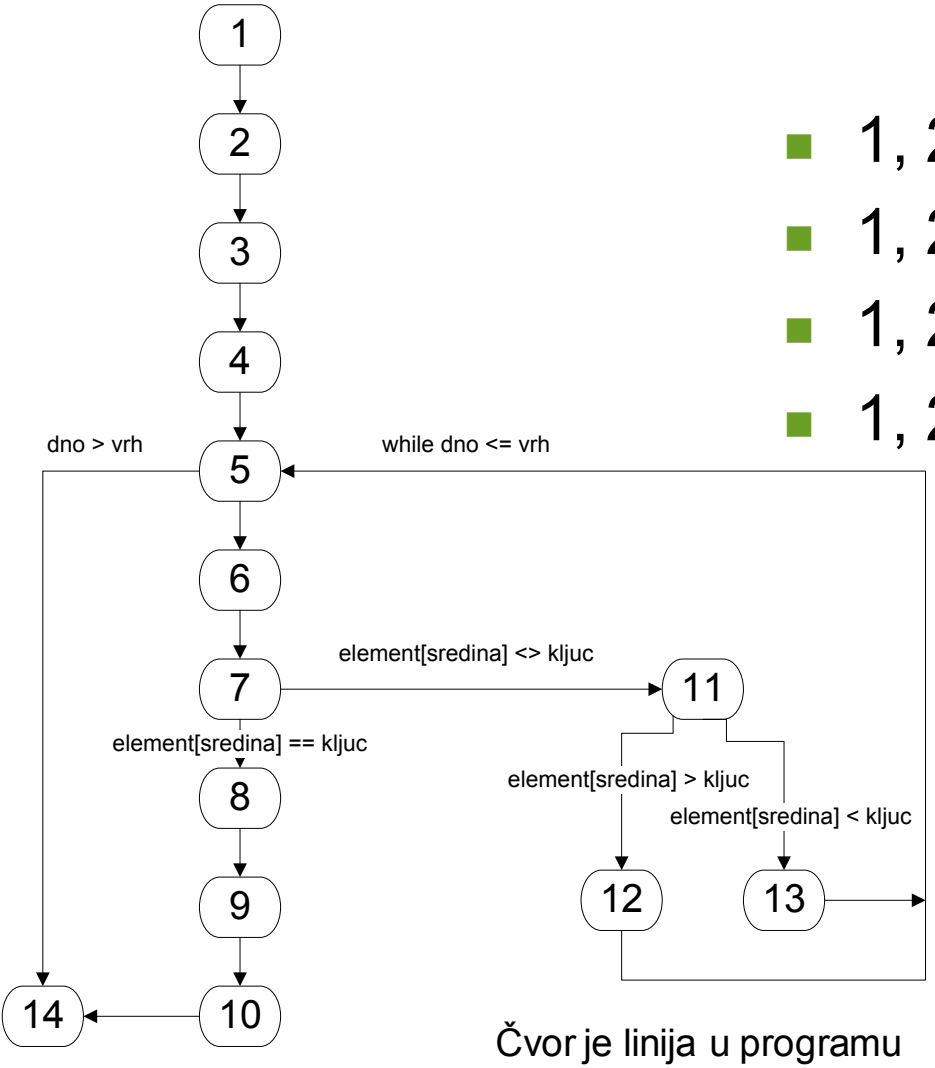
- Procesi=4
- Odluke=2
- Lukovi=7
- Čvor= 4+2=6
- P = 1 (nema dodatnih komp.)

CV(g)=7- 6 + 2=3

Preporuka:

- Ciklomatička složenost modula < 10

Primjer: Binarno pretraživanje



Čvor je linija u programu

CV(G) = Lukovi – Čvorovi + 2*P

=16 - 14 + 2 = 4

- 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 14
- 1, 2, 3, 4, 5, 14
- 1, 2, 3, 4, 5, 6, 7, 11, 12, 5, …
- 1, 2, 3, 4, 6, 7, 2, 11, 13, 5, …

Ako se obiđu svi navedeni putovi:

- svaka naredba je ispitana najmanje jednom
- ispitano je svako grananje

Ispitivanje uvjeta

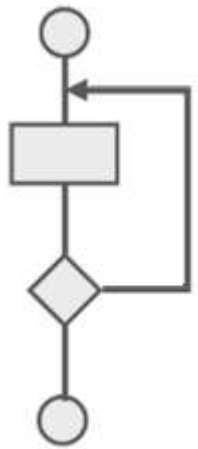
- Ispitivanje svih logičkih uvjeta u modulu
  - svaki uvjet poprima vrijednosti istinitosti: T i F
  - jednostavni uvjeti
    - Booleve varijable *True*, *False*
    - Jednostavni relacijski izrazi *a<b, a>=b* .....
  - složeni izrazi
    - *((a=b) & (c>d))*
- Metode ispitivanja
  - ispitivanje grana (svaka grana svakog uvjeta ispituje se bar jednom)
  - ispitivanje domene
    - Npr. za relaciju a<b, treba provesti 3 ispitna slučaja: a<b, a=b, a>b
    - Booleov izraz sa n varijabli $\Rightarrow 2^n$ ispitnih slučaja

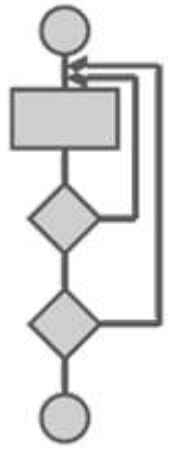
Ispitivanje petlji

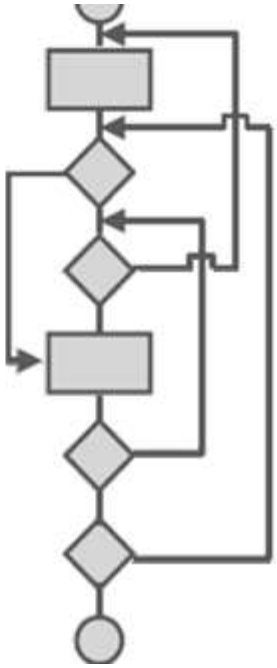
- Kritične točke programa
- Fokus na valjanost primjene petlje

```
while X < 20 loop
    {....}
end loop;
```

- Klasifikacija petlji
  a) Jednostavna (*engl. simple loop*)
     - n = max broj prolaza
  b) Ugnježđena (*engl. nested loop*)
     - Iznutra prema van (geom. složenost)
  c) Ulančane (*engl. concatenated loop*)
     - Pristup kao jednostavne ili ugnježđene
  d) Nestrukturirane (*engl. unstructured loops*)
     - Loše programiranje





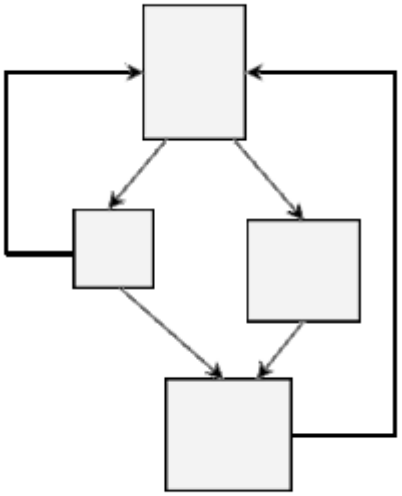


Primjer ispitivanja petlji

- Ispitni slučaj za jednostavnu petlju:
  - odrediti n= maksimalno dozvoljeni broj prolaza
  1. preskočiti petlju
  2. jedan prolaz
  3. dva prolaza
  4. m prolaza (m< n)
  5. n-1 prolaz
  6. n+1 prolaz
- Kako ispitati ugnježđene petlje?
- Kako ispitati ulančane petlje?

Ispitivanje protoka podataka

- Ispitivanje upravljačkog toka obzirom na uporabu varijabli i uočavanje kvarova provjerom uzorka uporabe podataka



- Za svaku cjelinu odrediti uporabu varijabli
  - ***definiranje*** *(D)* - *d*eklaracija varijable (objekta), konstruktor, pridjeljivanje vrijednosti
  - ***uporaba*** *(U)* - bez mijenjanja vrijednosti
  - ***brisanje*** *(K)* - *v*arijabla postaje nedefinirana, oslobađanje memorije (*engl. garbage collection*)
  - ***nebitno*** (X – prijašnje akcije i X~ nebitno nakon)
- Analiza slijeda akcija nad varijablama
- Npr. ispravni sljedovi – DD, DU, UU, UD, UK

Strukturno ispitivanje -zaključci

- Potencijalno beskonačan broj putova za ispitivanje
- Često ispituje ono što je implementirano, umjesto što bi trebalo biti
- Uvid u izvor pogreške
- Povećana ponovna uporaba ispitnih slučajeva
- Mogućnost ciljanog ispitivanja kritičnih dijelova

- Osjetljivo na promjene izvornog koda

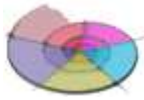
Ispitivanje sive kutije

- *engl. Gray box testing*
- Funkcijsko i strukturno ispitivanje predstavljaju ekstremne slučajeve
- Uobičajeno se funkcijsko ispitivanje kombinira s poznavanjem koda za potvrdu očekivanih rezultata
  - najčešće interne strukture podataka i algoritmi implementacije

AUTOMATIZACIJA ISPITIVANJA

Automatizacija ispitivanja

- Ispitivanje i ispravljanje pogrešaka skupo i vremenski zahtjevno
  - 1995. g. “50% of my company employees are testers, and the rest spends 50% of their time testing!” -Bill Gates
- Obzirom na prirodu programa pogodi su za ispitivanje primjenom drugih programima
  - *engl. Computer Aided Software Testing*
  - razvijeni mnogobrojni specijalizirani alati radne okoline
- Prednosti automatizacije
  - brže i jeftinije
  - poboljšanje točnosti
  - stabilno ispitivanje kvalitete
  - automatizirano dokumentiranje prijava pogrešaka i izvješćivanje
  - smanjenje ljudskog rada



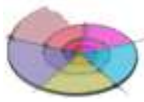
Programska podrška ispitivanju

- Automatizacija na različitim razinama ispitivanja:
- Komponenta
  - npr. izgradnja upravljačkih programa i, analiza pokrivanja
- Integracije
  - sučelja i protokoli
- Sustava
  - automatizirani izvođenje ispitivanja i performanse alata
- Prihvatljivosti
  - analiza zahtjeva, implementacije, upotrebljivosti …

- Pomoćne aktivnosti ispitivanja
  - praćenje kvarova (*engl. Bug Trackig Tools*)
  - upravljanje procesom ispitivanja (*engl. Test Management Tools*)

Automatizacija ručnog ispitivanja

- Visoka cijena ručne provedbe ispitivanja
  - Ponavlja se svakim ispitivanjem, potrebno ponoviti nakon svakog ispravljanja
- Automatizacija ručnog ispitivanja
  1. *generiranje ulaznih podataka i očekivanih rezultata*
  2. *izvođenje ispitivanja*
  3. *evaluacija*
  - jedinično 3-30 puta cijene ručnog ispitivanja
  - zahtjeva formalizirani ručni proces ispitivanja
  - cijena ovisi o postavljenom dosegu ispitivanja
    - Postupak kodiranja ispitnih slučajeva
  - cijena ponavljanja $\cong 0$
- Prednosti
  - Povećana pouzdanost
  - Povećana kvaliteta ispitivanja
    - automatizacija procesa
  - Kraće vrijeme izvođenja ispitnih slučajeva
  - Automatska analiza rezultata ispitnih slučajeva
- Nedostatci
  - Cijena
  - Vrijeme pripreme

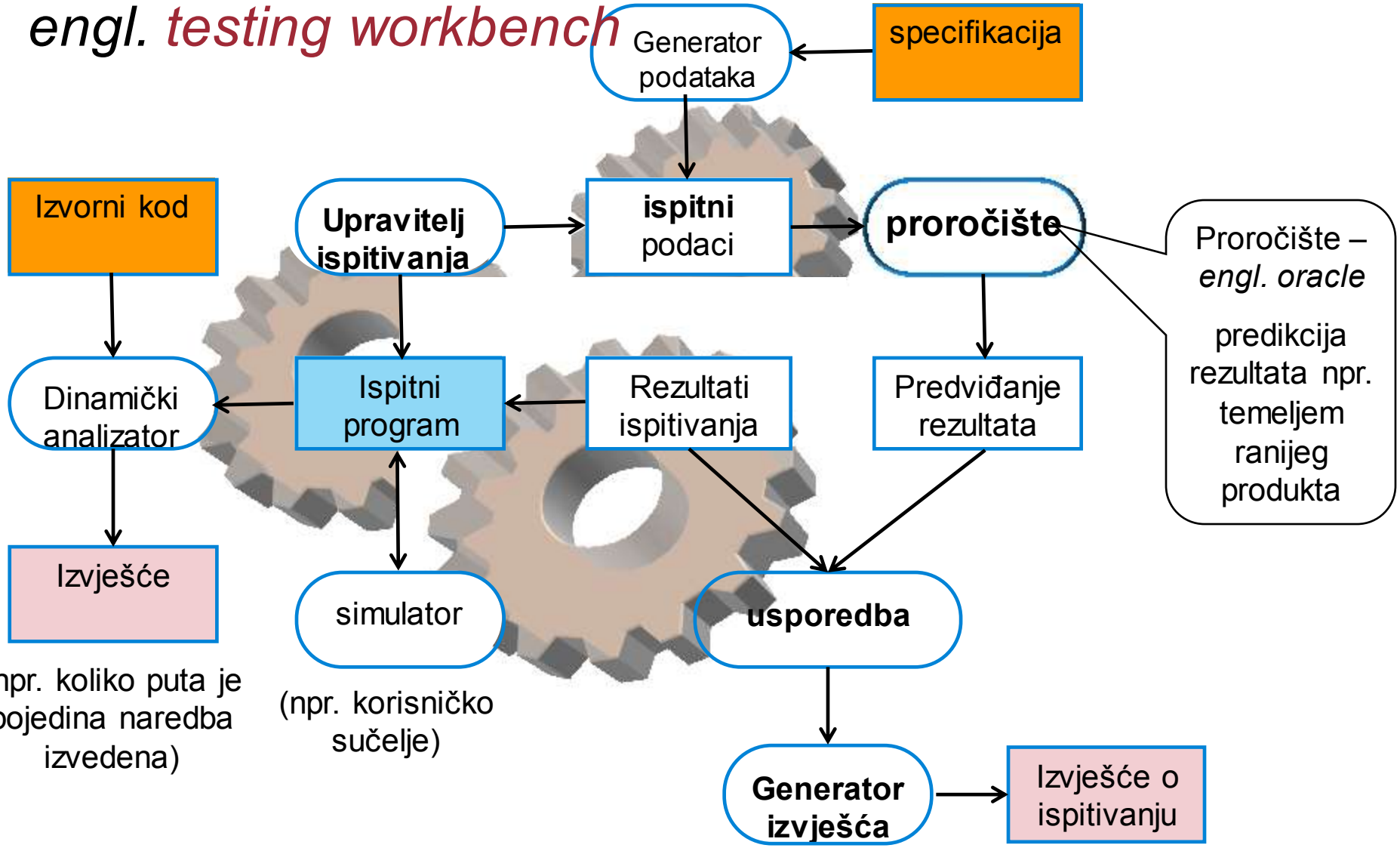


Razine automatizacije ispitivanja

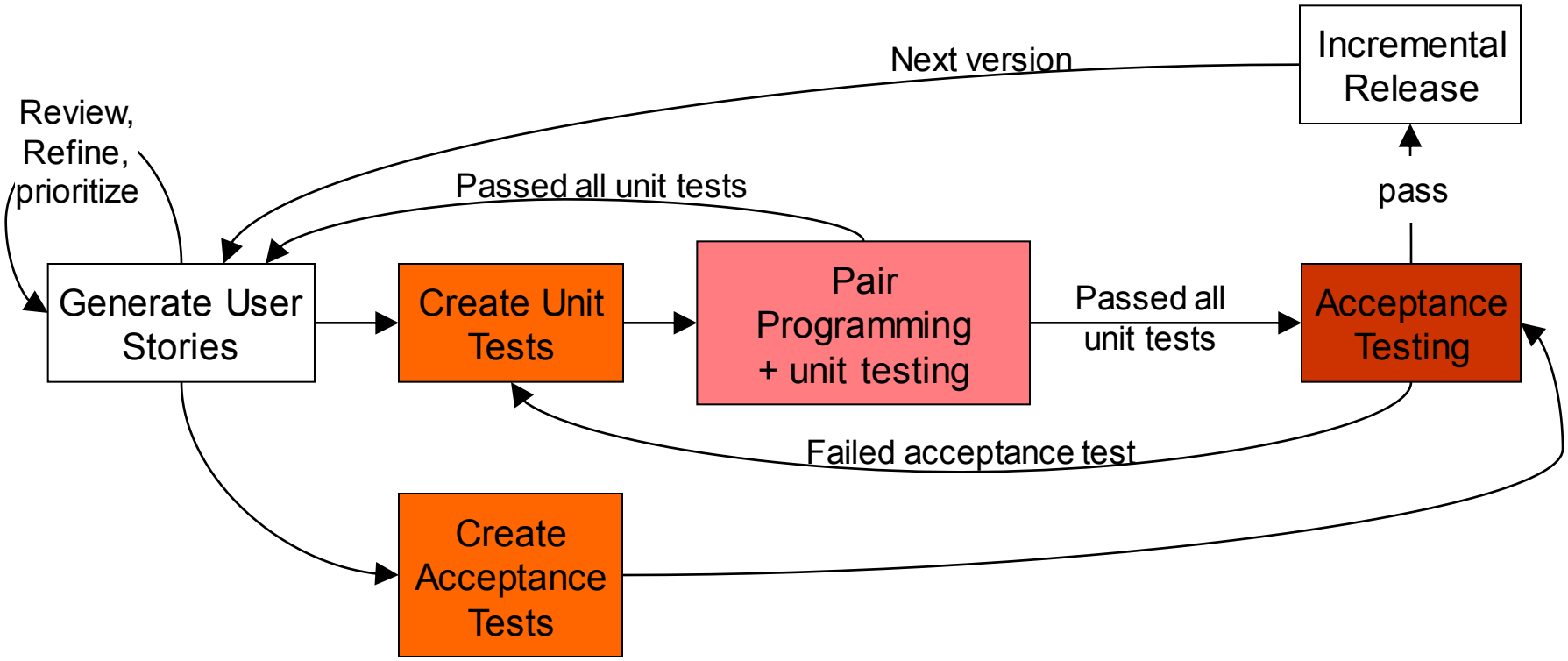
- Osnovna automatizacija
  - *metodologija snimanja i reprodukcije*
  - *engl. Record&Playback Methodology*
    - skripte generirane bilježenjem korisnikovih akcija
  - *podatkovno upravljana metodologija*
  - *engl. Data-Driven Methodology*
    - Podaci nisu ukodirani u skriptu - čitaju se iz datoteka
- Napredna automatizacija
  - funkcionalna dekompozija
    - Male skripte predstavljaju ispitne module i funkcije
  - radni okviri - *Framework*
    - *Test Framework*
    - *Process Framework*
    - *Hybrid Framework*

Ispitna radna klupa

- *engl. testing workbench*



Primjer: Extreme Programming (XP)



Primjer: JUnit

- JUnit – framework
  - okvir za ispitivanje Java programa
- Princip: "Code a little, test a little"
  - erich Gamma, Kent Beck
  - zasnovan na Javi
    - Java programi mogu ispitivati vlastiti kod
  - JUnit
    - Definira i izvodi ispitne slučajeve
    - Formalizira zahtjeve
    - Pisanje i ispravljanje koda
    - Integrira se s ispitnim slučajem
  - razvojne okoline
    - BlueJ, JBuilder, Eclipse, NetBeans
  - više: JUnit.org http://www.junit.org/
    - Resources for Test Driven Development
    - "JUnit is a simple framework for writing and running automated tests. As a political gesture, it celebrates programmers testing their own software. "

Primjer: JUnit

```
public class Primjer {
  static public int Zbroji(int a, int b) {
    return a + b;
  }
}
```

← izvorni kod

dodaci za ispitivanje

```
import junit.framework.Test ;
import junit.framework.TestSuite ;

public class testPrimjer{
 @Before public void setUp() {
  System.out.println("Testiranje počinje");
 }
 @After public void tearDown() {
  System.out.println("Završeno!!");
 }
 @Test public void testZbroji(){
  int i = Primjer.zbroji(2,3);
  assertEquals(5,i);
 }
 @Test public void testZbrojiL(){
  int i = Primjer.Zbroji(3,5);
  assertTrue("Pogrešna suma",i!=5);
 }
}
```

```
<junit printsummary="yes"
haltonfailure="yes"
showoutput="yes">
 <classpath>
  <pathelement path="${build}"/>
 </classpath>
 <batchtest fork="yes"
 todir="${reports}/raw/">
  <formatter type="xml"/>
  <fileset dir="${src}">
  <include name="**/*Test*.java"/>
  </fileset>
 </batchtest>
</junit>
```

Statistika uklanjanja kvarova

- Analize izvora pogrešaka ukazuju na tipičnu raspodjelu: 60% pogreška oblikovanja, a 40% pogrešaka kodiranja
- Efikasnost uklanjanja pogrešaka ovisi o kvaliteti razvojnog tima i primijenjenoj metodologiji

metoda	Potencijal pogrešaka	Efikasnost uklanjanja	Pogreške*
TSP	2.70	97%	0.08
CMMI 5	3.00	96%	0.12
RUP	3.90	95%	0.20
CMMI 3	4.50	93%	0.32
XP	4.50	92%	0.38
Agile	4.70	91%	0.42
CMMI 1	5.00	85%	0.75

Tip	Isporučeno pogrešaka*	KESLOC
System Software	0.4	1
Commercial Software	0.5	8
Information Software	1.2	10
Military Software	0.3	<1

Izvor: Jones, C.: Applied Software Measurement: Global Analysis of Productivity and Quality

***Defects per function point** – broj pogrešaka u odnosu na ukupan broj korisničkih funkcionalnosti
Capability Maturity Model Integration (**CMMI**) – definira razine zrelosti organizacije u primjeni metodologije poboljšanja procesa
Team Software Process (TSP) - definira procese organizacije s ciljem poboljšanja razine kvalitete i produktivnosti tima

Zaključak

- Ispitivanje je složena i zahtjevna aktivnost oblikovanja programske podrške
  - ***kritični dio*** razvoja sustava
  - provodi se ***na svim razinama*** i planira kao ***sastavni dio razvojnog procesa***
  - ***problem strategije***
    - veliki broj ciljeva ⇒ oblikovanje strategije za postizanje ciljeva
    - ***problem odluke*** – *engl. The Oracle Problem*
      - Da li je program prošao ispitivanje ili ne?
  - nemogućnost potpunog ispitivanja
  - loše oblikovani sustavi otežavaju ispitivanje
- Ispitivanje može pokazati postojanje pogreške, a ne može dokazati njihovo nepostojanje!
- Postoje razvijena pravila i razne dobre prakse
  - Upotrebljavati iskustvo i smjernice za oblikovanje ispitnih slučajeva
- Trend predstavlja integracija ispitivanja i razvoja
- Alternativa: uporaba formalnih metoda

Diskusija

Ispitni slučajevi za SimpleChat

Test Case 2002

System: SimpleChat Phase: 2

Client startup check without a login

Severity: 1

Instructions:

1. At the console, enter: java ClientConsole.

Expected result:

1. The client reports it cannot connect without a login by displaying:

   ERROR - No login ID specified. Connection aborted.
2. The client terminates.

Cleanup: (if client is still active)

1. Hit CTRL+C to kill the client.

Test Case 2003

System: SimpleChat Phase: 2

Client startup check with a login and without a server

Severity: 1

Instructions:

1. At the console, enter: java ClientConsole <loginID>

   where <loginID> is the name you wish to be identified by.

Expected result:

1. The client reports it cannot connect to a server by displaying:

   cannot open connection. Awaiting command.
2. The client waits for user input

Cleanup: (if client is still active)

1. Hit CTRL+C to kill the client.

Izvor: Lethbridge T.C., Laganière, R.: *Object-Oriented Software Engineering: Practical Software Development using UML and Java*

Ispitni slučajevi za SimpleChat

Test Case 2007

System: SimpleChat Phase: 2

Server termination command check

Severity: 2

Instructions:

1. Start a server (Test Case 2001 instruction 1) using default arguments.
2. Type #quit into the server's console.

Expected result:

1. The server quits.

Cleanup (If the server is still active):

1. Hit CTRL+C to kill the server.

Test Case 2013

System: SimpleChat Phase: 2

Client host and port setup commands check

Severity: 2

Instructions:

1. Start a client without a server (Test Case 2003).
2. At the client's console, type #sethost <newhost> where

<newhost> is the name of a computer on the network

3. At the client's console, type #setport 1234.

Expected result:

1. The client displays

Host set to: <newhost>

Port set to: 1234.

Cleanup:

1. Type #quit to kill the client.

Ispitni slučajevi za SimpleChat

Test Case 2019

System: SimpleChat Phase: 2

Different platform tests

Severity: 3

Instructions:

1. Repeat test cases 2001 to 2018 on Windows 95, 98, NT or 2000, and Solaris

Expected results:

1. The same as before.

Ispitne teme

- Definicija i terminologija ispitivanja programske podrške
- Proces ispitivanja programske podrške
- Svojstva ispitljivih programa
- Organizacija ispitivanje programske podrške
- Ispitivanje komponenti
- Specifičnost ispitivanja objektno usmjerenih sustava
- Integracijsko ispitivanje
- Strategije ispitivanja: ispitivanje patricija, iscrpno ispitivanje, ispitivanje temeljnih putova
- Osnovne tehnike ispitivanja: funkcijsko i strukturno ispitivanje
- Svojstva i organizacija automatizacije ispitivanja