Testiranje softvera

ETF BEOGRAD, 2019/2020.

VEŽBE #10, ASISTENT: DR DRAŽEN DRAŠKOVIĆ

Testiranje objektno orijentisanog softvera

OBJECT-ORIENTED TESTING

Testiranje OO softvera

- OO softver: najmanja jedinica testiranja je klasa tj. objekti koji obuhvataju podatke i funkcije za manipulaciju tim podacima
- Testiranje (unutar) klase za OO softver je ekvivalent jediničnog testiranja kod konvencionalnog softvera
- Dva glavna nivoa:
  - testiranje unutar klase (eng. *intra-class testing*), koje odgovara jediničnom testiranju kod konvencionalnog proceduralnog softvera
  - međuklasno testiranje (eng. *inter-class testing*), koje odgovara integracionom testiranju kod konvencionalnog softvera

OO softver - glavne karakteristike

- Ponašanje zavisno od stanja
- Enkapsulacija
- Nasleđivanje
- Polimorfizam i dinamičko vezivanje
- Apstraktne klase
- Obrada izuzetaka
- Konkurentnost

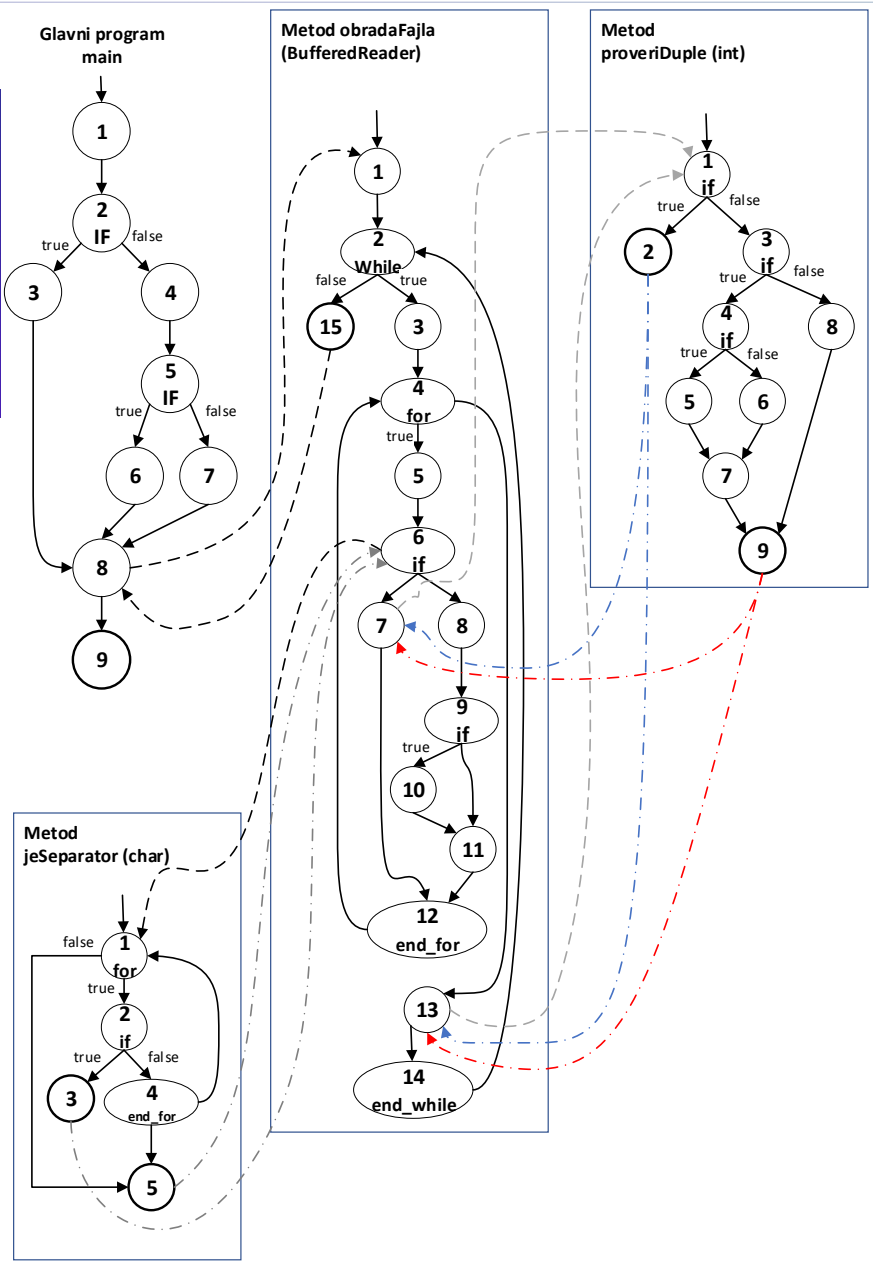
Zadatak 1 - Uređivač teksta

- Dat je softver za uređivanje teksta, realizovan na programskom jeziku Java. Cilj date implementacije je da u tekstualnom fajlu (sa ekstenzijom .txt), koji se čita kao argument poziva programa ili kao ime fajla sa standardnog ulaza, pronađe reči (dve ili više) koje se ponavljaju. Za svaku od takvih reči potrebno je napisati koje su reči u pitanju, u kojim redovima teksta su pronađene i na kojim pozicijama u okviru tih redova.
- Realizovati:
- a) Graf kontrole toka za metodu koja vrši obradu fajla (i eventualno druge metode koje poziva).
- b) Graf stanja i prelaza ovog softverskog sistema.
- c) Test primere koje obuhvataju sva stanja i prelaze iz grafa pod b).

Zadatak 1 - Uređivač teksta - rešenje (1)

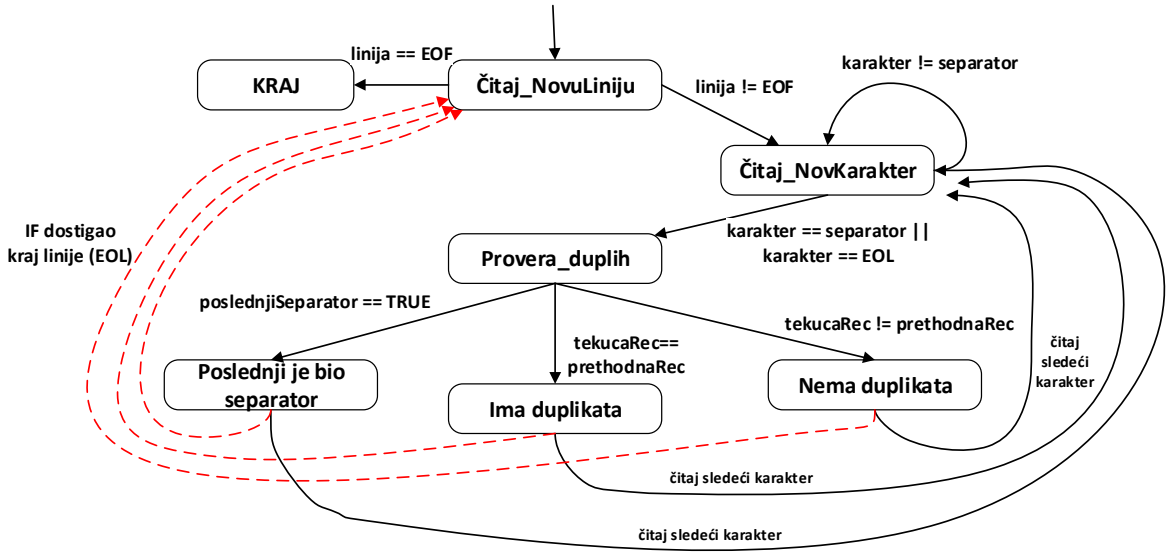
- Konstrukcija konačnog automata (FSM) - graf kontrole toka glavnog programa i 3 preostale metode
- Nije u potpunosti FSM:
  - Čvorovi nisu stanja
  - Metode moraju da vrate rezultat na mestima poziva tih metoda
    (grafovi onda sadrže ugrađene nedeterminizme)
  - Implementacija mora biti kompletno završena pre nego što pređemo da realizujemo graf
    (naš cilj: pripremiti testove što je ranije moguće)
  - Nesrazmernost grafa veličini softvera => Komplikovan graf

Zadatak 1 - Uređivač teksta - rešenje (2)



Zadatak 1 - Uređivač teksta - rešenje (3) - b

- FSM baziran na softverskoj strukturi prikazuje opšti tok procesiranja u programu (apstrahovani graf kontrole toka), što predstavlja glavnu prednost takvog konačnog automata



Zadatak 1 - Uređivač teksta - rešenje (4)

- Modelovanje stanja promenljivih:
  - Prvi korak je da odredimo koje promenljive mogu učestvovati u stanjima, zatim izabrati one relevantne za FSM, na primer globalne i klasne promenljive
- Klasa *Uredjivac* definiše 4 promenljive: *poslednjiSeparator*, *tekucaRec*, *prethodnaRec* i *separatori*. Promenljiva *separatori* je definisana na nivou klase, pa ne treba da bude odabrana kao deo stanja.
- Teoretski: svaka kombinacija ove 3 promenljive definiše različito stanje
  U praksi: beskonačan broj stanja. Na primer promenljive *tekucaRec* i *prethodnaRec* su tekstualni podaci i imaju neograničen skup vrednosti.
- Rešenje: identifikovati vrednosti / opsege vrednosti koji će biti zastupljeni kao stanja

Zadatak 1 - Uređivač teksta - rešenje (5)

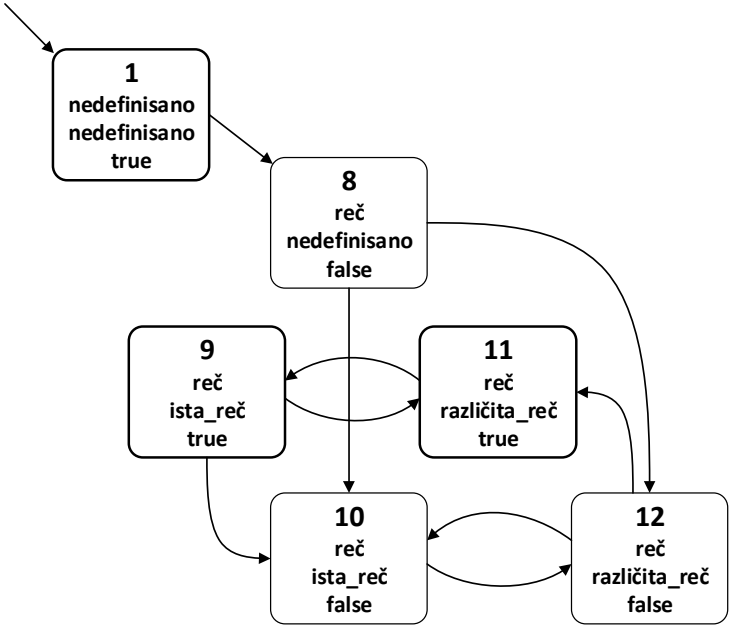
- Stanja:
  - *tekucaRec*: nedefinisano, reč
  - *prethodnaRec*: nedefinisano, ista_reč, različita_reč
  - *poslednjiSeparator*: *true*, *false*
  - gde su reči: reč, ista_reč i različita_reč, promenljive tekstualnog tipa (string)

Zadatak 1 - Uređivač teksta - rešenje (6)

1. (nedefinisano, nedefinisano, *true*)
2. (nedefinisano, nedefinisano, *false*)
3. (nedefinisano, ista_reč, *true*)
4. (nedefinisano, ista_reč, *false*)
5. (nedefinisano, različita_reč, *true*)
6. (nedefinisano, različita_reč, *false*)
7. (reč, nedefinisano, *true*)
8. (reč, nedefinisano, *false*)
9. (reč, ista_reč, *true*)
10. (reč, ista_reč, *false*)
11. (reč, različita_reč, *true*)
12. (reč, različita_reč, *false*)

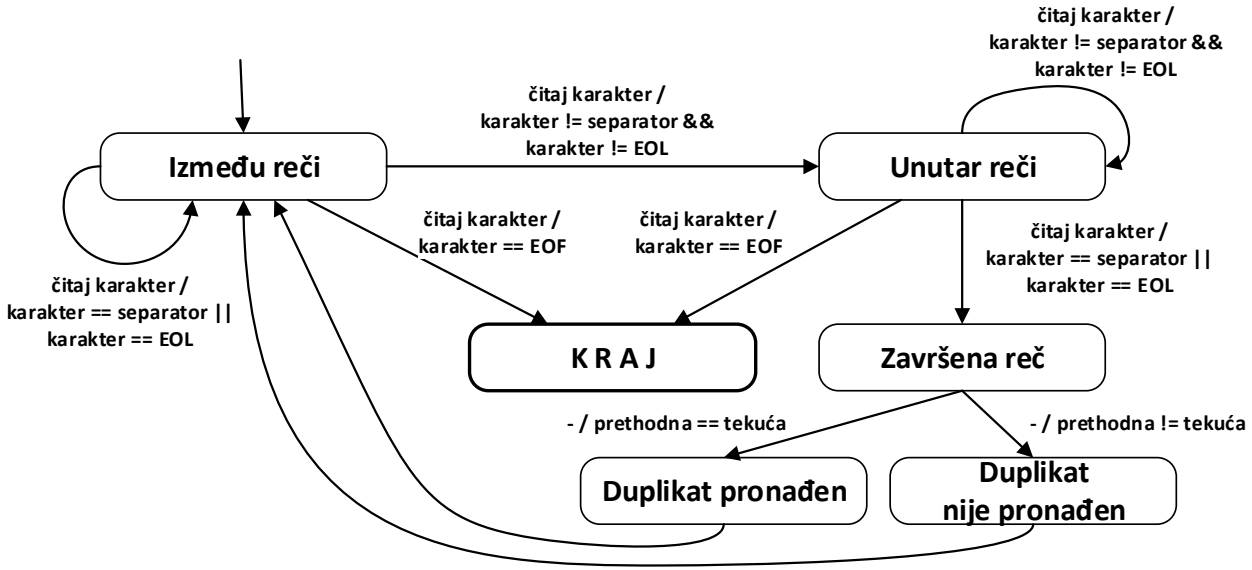
Zadatak 1 - Uređivač teksta - rešenje (7)

- Konačni graf stanja i prelaza, za 3 promenljive (*tekucaRec*, *prethodnaRec*, *poslednjiSeparator*) izgleda ovako:



Zadatak 1 - Uređivač teksta - rešenje (8)

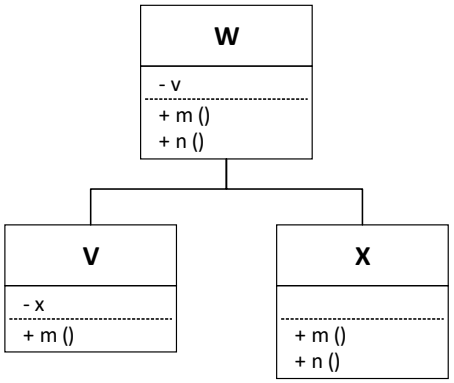
- Poslednja metoda za dobijanje FSM zasniva se na eksplicitinim zahtevima ili formalnoj specifikaciji koja opisuje ponašanje tog softvera, a koju testeru dostavlja projektant softvera (na primer UML dijagram stanja). Za posmatrani program, ovakav FSM mogao bi da ima sledeći izgled:



Zadatak 2 - Testiranje redefinisanih metoda

- Dat je segment jednog programa i UML klasni dijagram, koji prikazuje klase koje se koriste u prikazanoj metodi f. V i X su izvedene klase iz klase W, klasa V redefiniše (eng. *overrides*) metodu m(), a klasa X redefiniše metode m() i n(). Minus (-) označava atribut sa privatnim pravom pristupa, a plus (+) označava javno pravo pristupa. Komentarisati šta je potrebno testirati u ovom programu, da bismo bili sigurni da se sve metode izvršavaju korektno.

```
1  void f(boolean b)
2  {
3        W obj;
4        ...
5        if(b)
6              obj = new V();
7        else
8              obj = new W();
9        ...
10       obj.m();
11 }
```

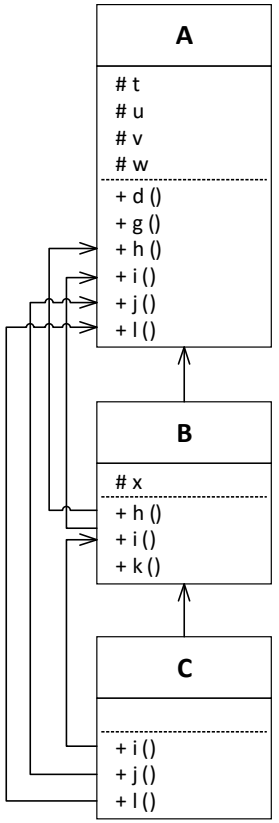


Zadatak 2 - Testiranje redefinisanih metoda - Rešenje

- U slučaju kada je ulazni argument metode *f(boolean)*, b = *true*, izvršiće se linija 6, odnosno pozvaće se konstruktor V(), pa će stvarni tip, u liniji 10, biti V, i tada će se pozvati metoda *m()*, koja je redefinisana u klasi V. U slučaju kada je ulazni argument metode *f(boolean)*, b = *false*, izvršiće se linija 8, odnosno pozvaće se konstruktor W (konstruktor osnovne klase), pa je zaključak da će se tada, u liniji 10, kao stvarni tip pozvati objekat nad klasom W, odnosno izvršiće se metoda *m()*, koja je realizovana u osnovnoj klasi W.
- Redefinisana metoda *m()* u izvedenoj klasi X se u ovom zadatku neće izvršiti, bez obzira na odabrani test primer.

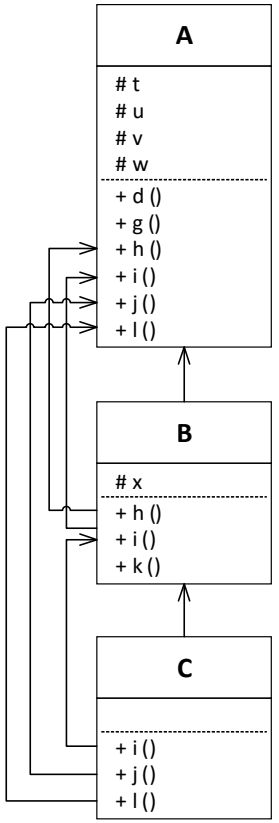
Zadatak 3 - Nasleđivanje i Yo-Yo graf (1)

- Neka je data sledeća struktura programa sa 3 realizovane klase: A, B i C. Promenljive koje su označene sa # imaju zaštićeno (*protected*) pravo pristupa, a date su i metode sa javnim (*public*) pravom pristupa koje su označene sa +. Korena klasa A sadrži stanja za 4 promenljive i 6 metoda. Promenljive su zaštićene što znači da su dostupne u klasama potomcima klase A, odnosno u klasama B i C. Klasa B deklariše stanje 1 promenljive i 3 metode, a klasa C samo 3 metode. Strelice na slici pokazuju važnost metoda: metoda B::h() redefiniše (nadjačava) metodu A::h(), metoda B::i() redefiniše metodu A::i(), a metoda C::i() redefiniše metodu B::i(), C::j() redefiniše A::j(), i C::l() redefiniše A::l(). Tabela pored dijagrama prikazuje stanje definisanih promenljivih i u kojim metodama se koriste te promenljive.
- U implementaciji klase A, pretpostavimo da metoda d() poziva metodu g(), g() poziva h(), h() poziva i() i i() poziva j(). Dalje pretpostavimo da kod implementacije B, metoda h() poziva metodu i(), i() poziva metodu i() iz svoje roditeljske klase (A), a k() poziva l(). Konačno, kod C imamo metodu i() koja poziva roditeljsku metodu i() iz klase B, i metoda j() koja poziva k().
- Nacratati *yo-yo* graf za opisanu specifikaciju programa i klasni dijagram sa slike. Definicije i upotrebe promenljivih unutar metoda pojedinih klasa date su u tabeli.



Zadatak 3 - Nasleđivanje i Yo-Yo graf (2)

Metod	Definicije	Upotrebe
A :: h( )	{ A :: u, A :: w }	
A :: i( )		{ A :: u }
A :: j( )	{ A :: v }	{ A :: w }
A :: l( )		{ A :: v }
B :: h( )	{ B :: x }	
B :: i( )		{ B :: x }
C :: i( )	{ C :: y }	
C :: j( )		{ C :: y }
C :: l( )		{ A :: v }



Zadatak 3 - Nasleđivanje i Yo-Yo graf - Rešenje