Testiranje grafičkih korisničkih interfejsa (GUI)

Uvod

- GUI je sredstvo korišćenja tj. interakcije sa sistemom
- GUI se mogu naći svuda, čak i u sistemima koji su kritični sa stanovišta sigurnosti.
- Funkcionišu na različitim vrstama uređaja (web, pc, tablet, palm, mobile)
- GUI interaguju sa ostatkom aplikacije putem poziva metoda ili poruka
- GUI mogu izvršavati “udaljeni” programski kod
- GUI reaguju na korisničke događaje (na primer, klik mišem)
  - GUI spadaju u sisteme pokretane događajima
- Testiranje korektnosti GUI je važno za sigurnost, upotrebljivost i robusnost sistema (odgovor na nelegalne ulaze).
- Ceo sistem se može izvršavati posredstvom GUI
- It is estimated that an average of 48% of the application code and 50% of the time spent with implementation are dedicated to the user interface.
- The testing phase of the software life cycle may consume around 50% of the total time of the project.

Pristupi testiranju GUI

- **Manuelno**
  - Zasnovano na poznavanju funkcija aplikacija i domena korišćenja od strane korisnika ili testera
- **Slučajno (random input)** testiranje poznato i kao majmunsko testiranje.
- **Capture and Replay tehnika**
  - Zasnovano na snimanju i reprodukciji korisničkih sesija
- **Zasnovano na modelu**
  - Korisničke sesije za izvršavanje testa biraju se na osnovu formalnog modela GUIja
    - Model stanja
    - Model događaja

Manuelno testiranje GUI

- Tzv. **slučajno** (slobodno, neupravljano) **testiranje** od strane krajnjih korisnika (random human testing). Npr. beta izdanja testiraju odbrane grupe korisnika nekoliko nedelja. Ne nudi garancije da su pokrivene sve funkcije
- Sistematičniji pristupi:
  - **Inspekcija:** grupa specijalista pregleda interfejs u odnosu na pravilnik. Pravila se mogu odnositi na konkretne fizičke odlike GUIa (dimenzije, boje,...) zatim na opštija pravila vezana za upotrebljivost npr. kako organizovati strukturu menija, podrška za undo itd.
  - **Upitnici**: Korisnici mogu da eksperimentišu sa sistemom i potom odgovaraju na upitnike o svojim iskustvima. Pitanja mogu biti i subjektivna i objektivna, npr. “da li vam se sistem dopada” do “[ta biste promenili u sistemu” ili pitanja u vezi izgleda i prijave grešaka.
  - **Test upotrebljivosti**: GUI se koristi pod kontrolisanim uslovima od strane krajnjih korisnika pod nadzorom evaluatora, koji prikupljaju podatke o karakteristikama interakcije sa sistemom: vreme koje korisniku treba da obavi zadatak, broj grešaka koji je korisnik napravio tokom obavljanja zadatka, koliko vremena korisniku treba da ponovo obavi isti zadatak i emotivni odziv, kako se korisnik oseća po obavljanju zadatka.

Nedostaci manuelnog testiranja GUI

- Puno zavisti od sposobnosti testera i njegovog poznavanja aplikativnog domena
- Monotono, frustrirajuće, podložno ljudskim greškama
- Previše napora za konstruisanje, izvršavanje i analizu rezultata testa, kao i za regresivno ponavljanje testa posle izmena koda
- Slabi kriterijumi pokrivenosti (sekvence dodađaja nisu dovoljno dugačke da izazovu greške)

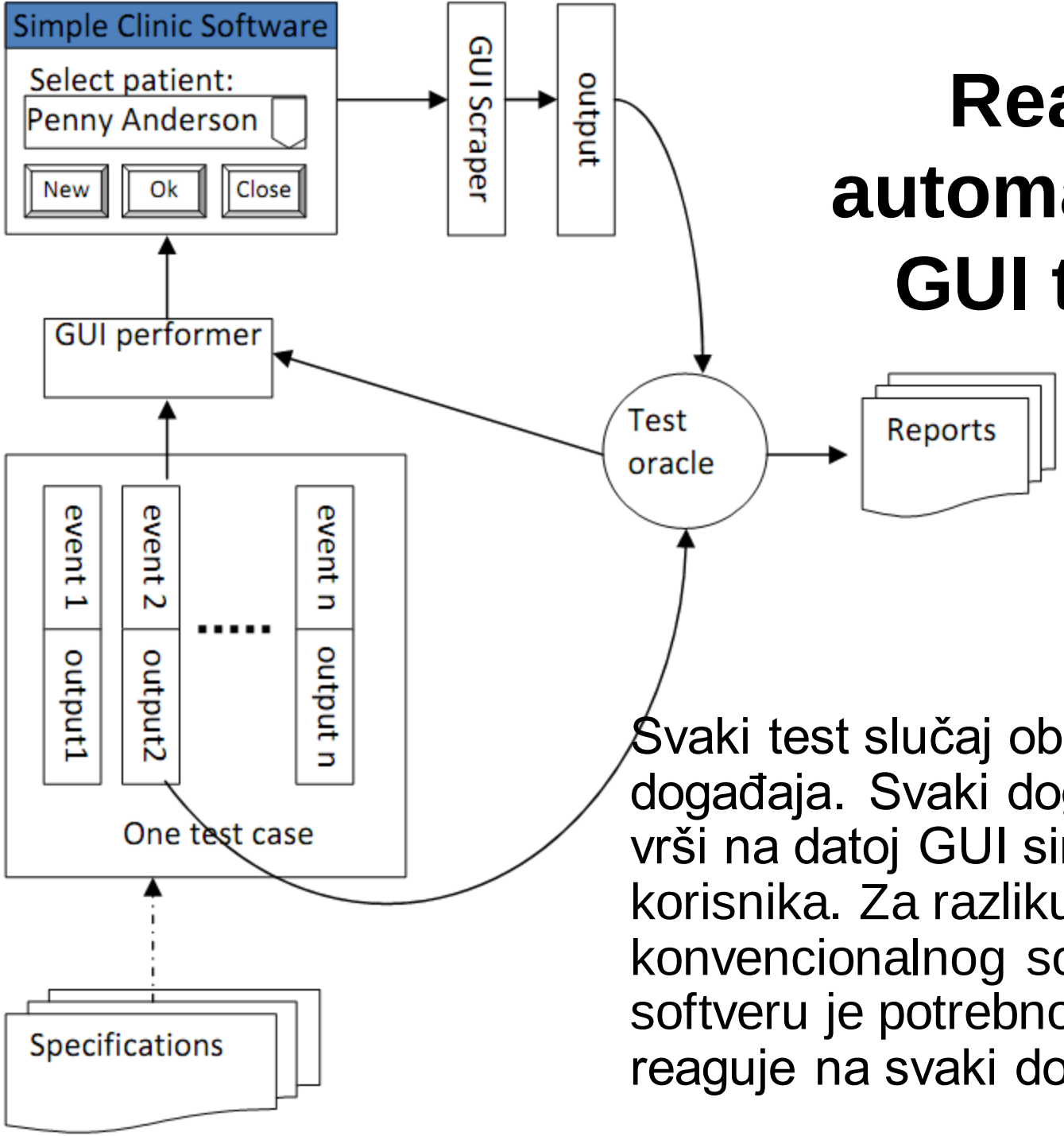
Slučajno (stohastičko) testiranje

- Kao da neko ko ne zna šta radi sedi ispred računara i interaguje sa tastaturom i mišem. Microsoft tvrdi da se 10-20% grešaka u njihovim projektima nađe na ovaj način
- Glupi “majmun” ne poznaje tekuće stanje aplikacije niti legalne i nelegalne ulaze. Ne može prepoznati pogrešan izlaz, već mu je jedini cilj da obori aplikaciju. Primer: Rational Test Factory
- Pametniji majmuni imaju neko znanje o aplikaciji koju testiraju, npr. u obliku modela stanja i znaju da provere da li je ciljno stanje ono koje je očekivano prema modelu.
- Glavni nedostatak slučajnog testiranja je slaba pokrivenost koda.

Capture and Replay

- U ovoj vrsti alata, test skriptovi se konstruišu tako da se beleži interakcija testera (unos sa tastature, pokreti i akcije miša,...) sa GUI, u cilju kasnijeg reprodukovanja.
- Ovi alati imaju režim **snimanja**, u kojem sve korisnikove akcije bivaju sačuvane u test skriptu, i režim **reprodukcije**, u kojima se ti test skriptovi izvršavaju.
- Ovi alati često koriste **jezik za skriptovanje** koji inženjeri mogu koristiti za održavanje test skriptova. Oni mogu, na primer, snimiti osnovni scenario za testiranje i modifikovati kasnije ručno da ga učine efikasnijim, dodaju dodatne verifikacije, parametrizuju ga za različite podatke itd.

Realizacija automatizovanog GUI testiranja



Svaki test slučaj obuhvata niz događaja. Svaki događaj treba da se vrši na datoj GUI simulirajući rad korisnika. Za razliku od konvencionalnog softvera GUI softveru je potrebno vreme da se reaguje na svaki događaj.

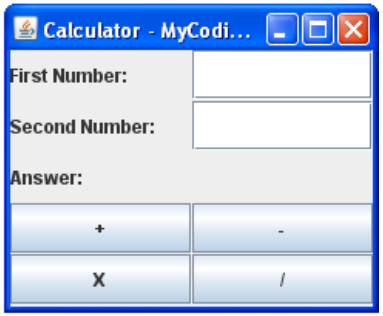
Realizacija automatizovanog GUI testiranja

Nakon što GUI završi reakciju, GUI skrejper čita status svih komponenti i poredi ih sa unapred sačuvanim očekivanim informacijama koje su vezane za događaj pomoću test prediktora. Neslaganja u rezultati mogu biti vraćena nazad GUI interpretatoru, tako da može da odluči da li ostale događaje ovog testa treba izvršiti ili ne. Sve neslaganja će biti upisana izveštaj o defektima za kasniju analizu.

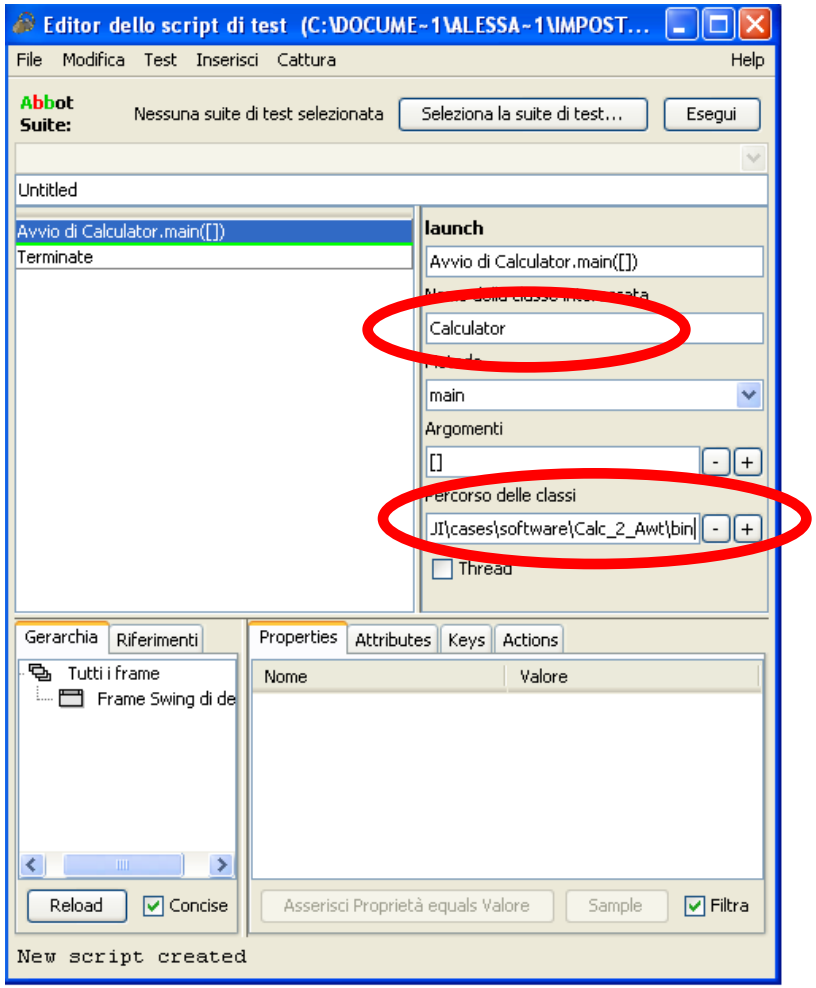
Primer alata: Abbot

- Is a tool that helps writing System/GUI tests for **Java AWT/Swing** applications
- Consists of a recorder, player, and an editor (via Costello, built on top of Abbot)
- Records tests script in Java
- Allows to write test cases directly from Java code (named programmatic GUI testing)
- Allows to insert assertions in the script easily
- Shows testing results using JUnit's control bar
  - Red/green
- Two main building blocks:
  - ComponentReferences to get a handle on a GUI component
  - Robot: to perform user-level actions on various GUI components
- **http://abbot.sourceforge.net**

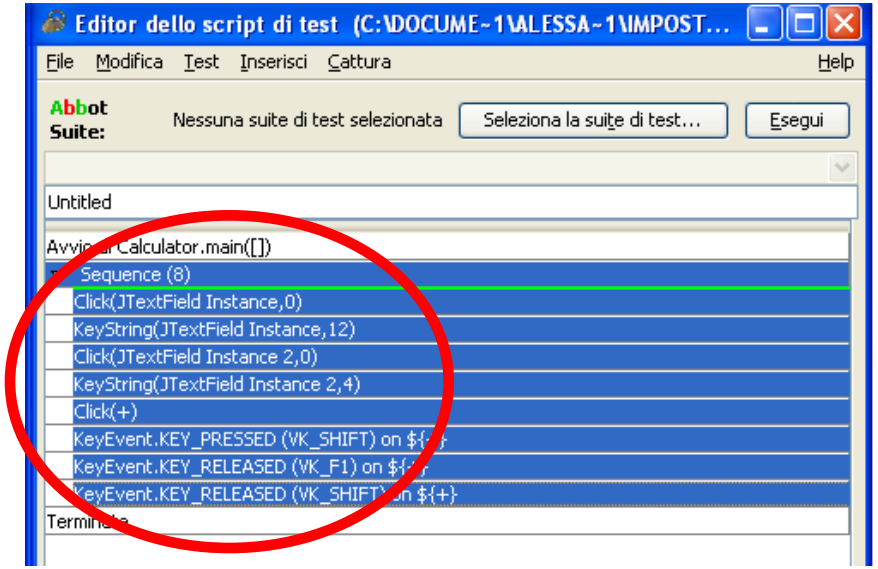
Abbot primer (1)



- A calculator application was developed (AWT/Swing)
- We want to write tests for the calculator
  - Ex. add two values
- Steps:
  1. create a new empty Abbot test
  2. Initialize the test with the application under test information
  3. build the test (recording)

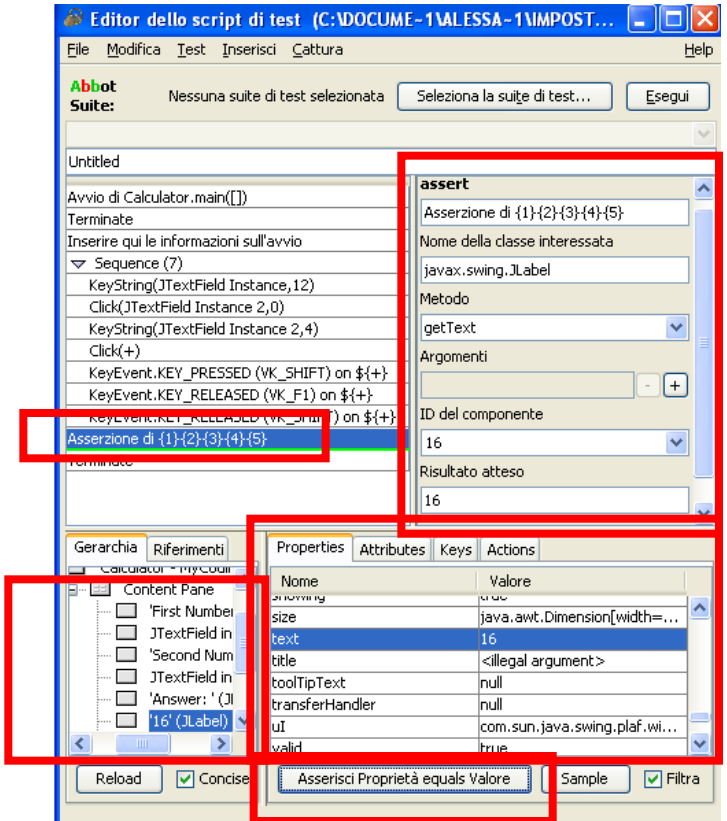


Abbot primer (2)



Recording

1. Start recording of all actions
2. Execute the software
3. Add assertions
   1. select the GUI element with SHIFT+F1
   2. select the property (e.g., (“Text”)
   3. set the expected value
4. Stop recording



Prednosti alata za snimanje i reprodukciju

- Ovi alati mogu imati dobre mogućnosti opservabilnosti izlaza, npr. Optičko prepoznavanje znakova (OCR) i tehnike obrade slike.
- Mogu biti korisni za regresiono testiranje i u drugim kontekstima, kao što su: demonstracije, daljinska podrška; analiza ponašanja korisnika; makro funkcionalnost i obrazovni scenariji.
- Reprodukcija ide znatno brže od manuelnog rada sa aplikacijom

Nedostaci ovih alata

- Odlažu testiranje za kraj procesa razvoja jer mogu biti uptrebljeni jedino kada je GUI sasvim ili delimično implementiran.
- Ne pružaju nikakvu podršu za projektovanje test primera niti za evaluaciju testova u skladu sa nekim kriterijom
- Testiraju samo ono što već radi
- Ako se promeni implementacija, često se moraju svi skriptovi koji se odnose na promenjeni deo snimati iz početka.
- Nekad samo male promene u interfejsu (npr. Rezolucija ekrana) čine sve test skriptove nevalidnim.

Testiranje GUI zasnovano na modelu stanja

GUI (**Graphical User Interface**) je hijerarshijski, grafički prednji kraj softverskog sistema

GUI sadrži **grafičke objekte** *w*, engl. widgets, svaki sa skupom osobina *p*, koji imaju diskretne vrednosti *v* u vreme izvršavanja.

U proizvoljnom trenutku izvršavanja programa, vrednosti osobina svakog GUI objekkta definišu stanje GUI-ja: {… (w, p, v), …}

Grafički događaj *e* uzrokuje promenu stanja, koja GUI prevodi iz stanja S u sledeće stanje S'.

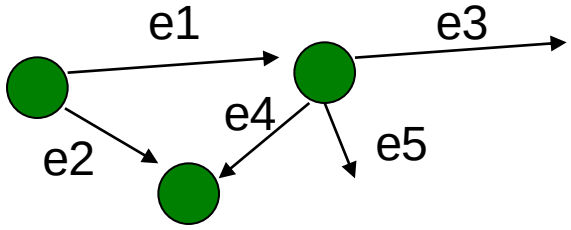
Model stanja

Konstrukcija modela stanja GUI-ja, tj. modela konačnog automata (FSM Finite State Model):

- Stanja su slike ekrana tj. trenutna reprezentacija GUI-ja
- Prelazi su GUI događaji koji menjaju stanje

Metod testiranja na osnovu modela stanja:

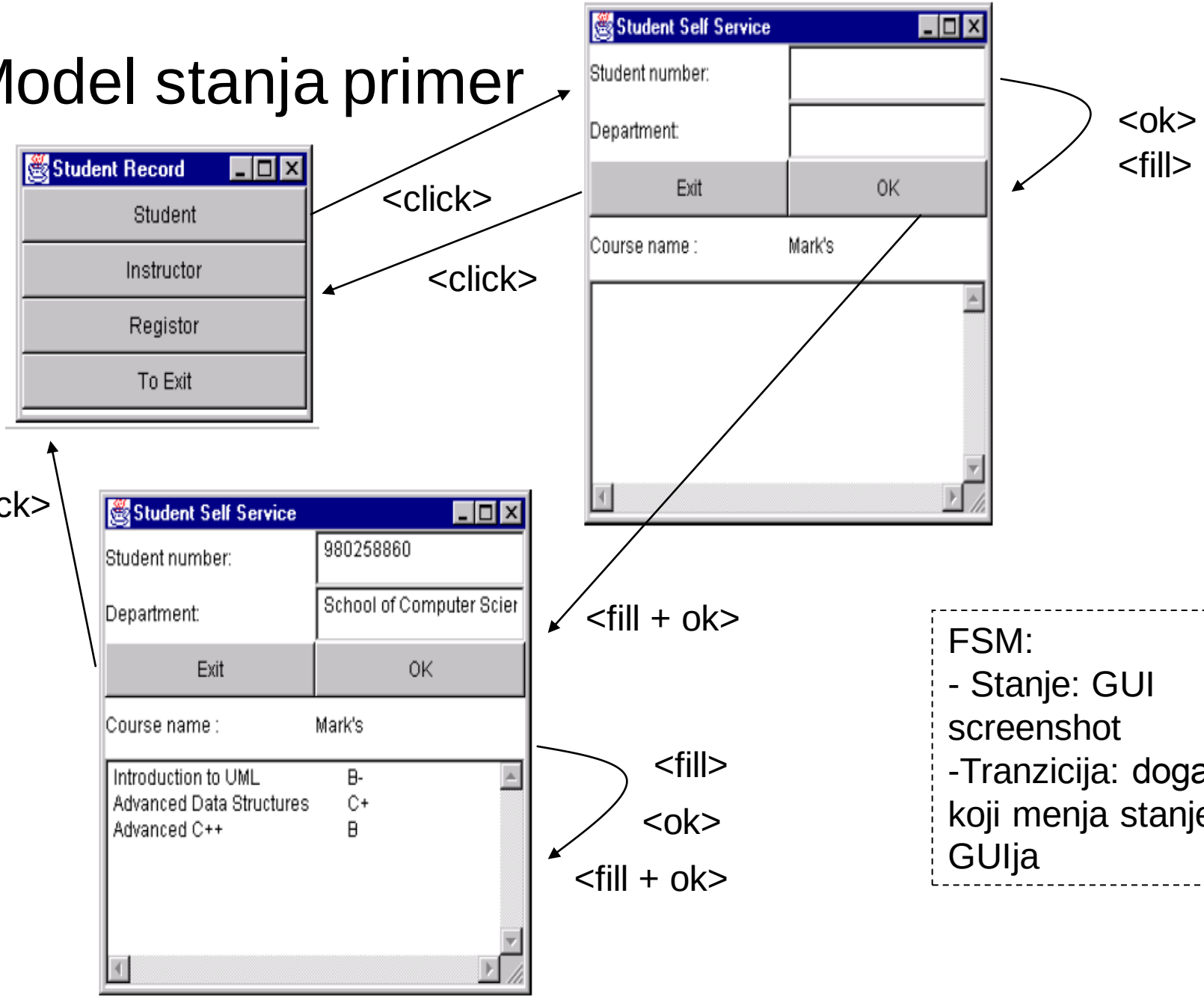
1. Kreirati FSM legalnih sekvenci koje korisnik može da izvrši.
2. Na osnovu FSMa generisati sekvence događaja: kriterijum - pokrivanje legalnih prelaza (takozvani Interaction pairs). Dobijeni testovi se zovu *Complete Interaction Sequences*.



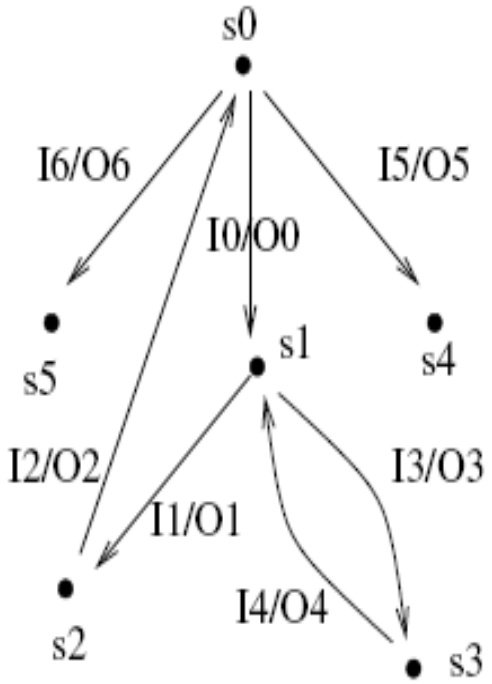
Nelegalni prelazi

- Model: faulty/incompatible interaction pair (FIP)
- Dodati sledeće grane u dijagram stanja:
  - Dodati granu u suprotnom smeru gde god postoji grana u jednom smeru
  - Dodati povratnu granu stanju koje je nema
  - Dodati obostrane grane između stanja koja nemaju granu
- Izvršiti pokrivanje svih ovako dobijenih prelaza. Dobijene sekvence se nazivaju Faulty Complete Interaction Sequence (FCIS)

Model stanja primer



..primer



Problem: ekspozija stanja!

→ Rešenja (informativno): Apstrakcija više realnih stanja u jedno, zatim varijabilni model stanja (VFSM) koji pored FSM dozvoljava i skup globalnih promenljivih.

I0: click *Student* button

O0: *Student Self Service* window shows up

I1: input student number 980258860 (an existing student number)

O1: *department name* shows *School of Computer Science*, and course-mark shows
Software Engineering: B-
Data Structures: C+
Programming Language with C++: B

I2: click *Exit* button

O2: *Student Self Service* window closed and *Student Record* window shows up

I3: input student number 999999999 (an invalid student number)

O3: information dialog box appears

I4: click *close* button in the information dialog box

O4: information dialog box closed and *Student Self Service* window shows up

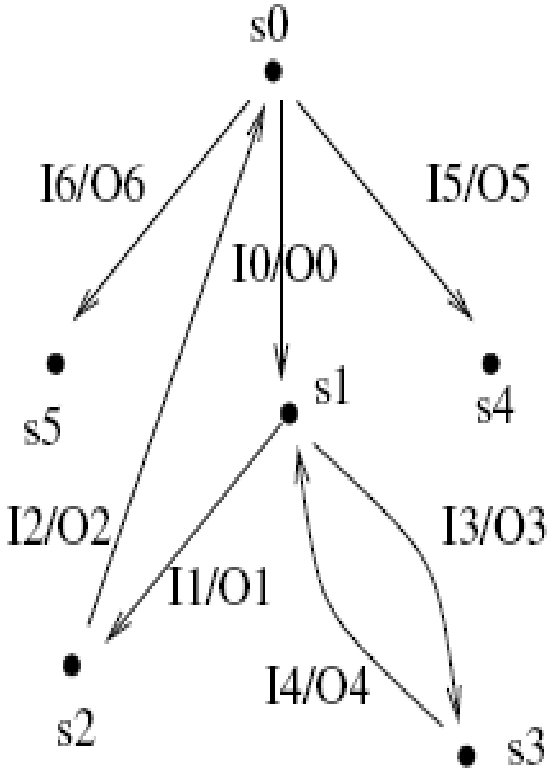
I5: click *Instructor* button

O5: *Instructor password* window shows up

I6: click *Registrar* button

O6: *Registrar password* window shows up

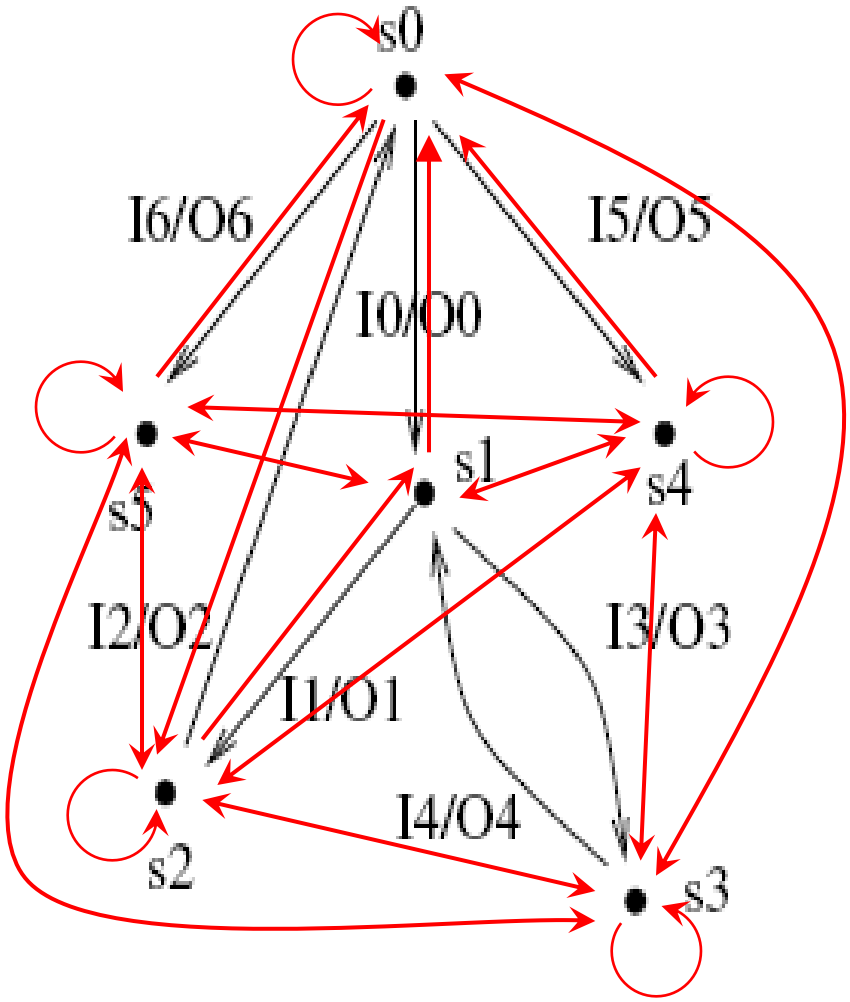
Primer



- Pokrivanje svih legalnih prelaza (ovde se prelaz naziva Interaction pair IP, a test primer koji pokriva prelaze Complete Interaction Sequence, CIS):

1. s0, I6/O6, s5
2. s0, I5/O5, s4
3. s0, I0/O0, i3/O3, I4/O4, I1/O1, I2/O2, s0

Nelegalni prelazi



- Model: faulty/incompatible interaction pair
  - Dodati granu u suprotnom smeru gde god postoji grana u jednom smeru
  - Dodati povratnu granu stanju koje je nema
  - Dodati obostrane grane između stanja koja nemaju granu
- Treba pokriti sve crvene grane (FIP) test sekvencama (FCIS)

Model zasnovan na događajima

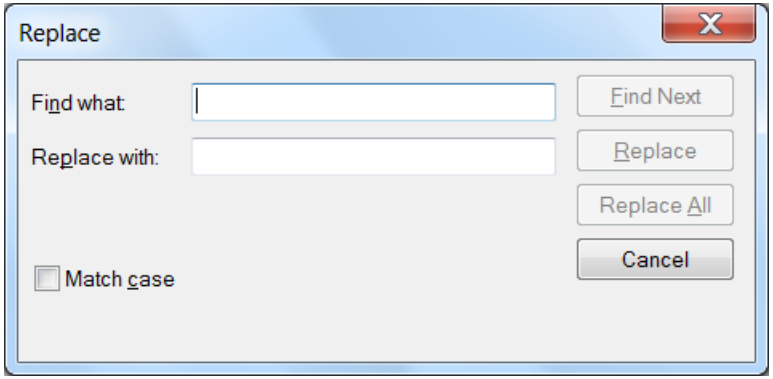
Model the space of **GUI event interactions** as a graph

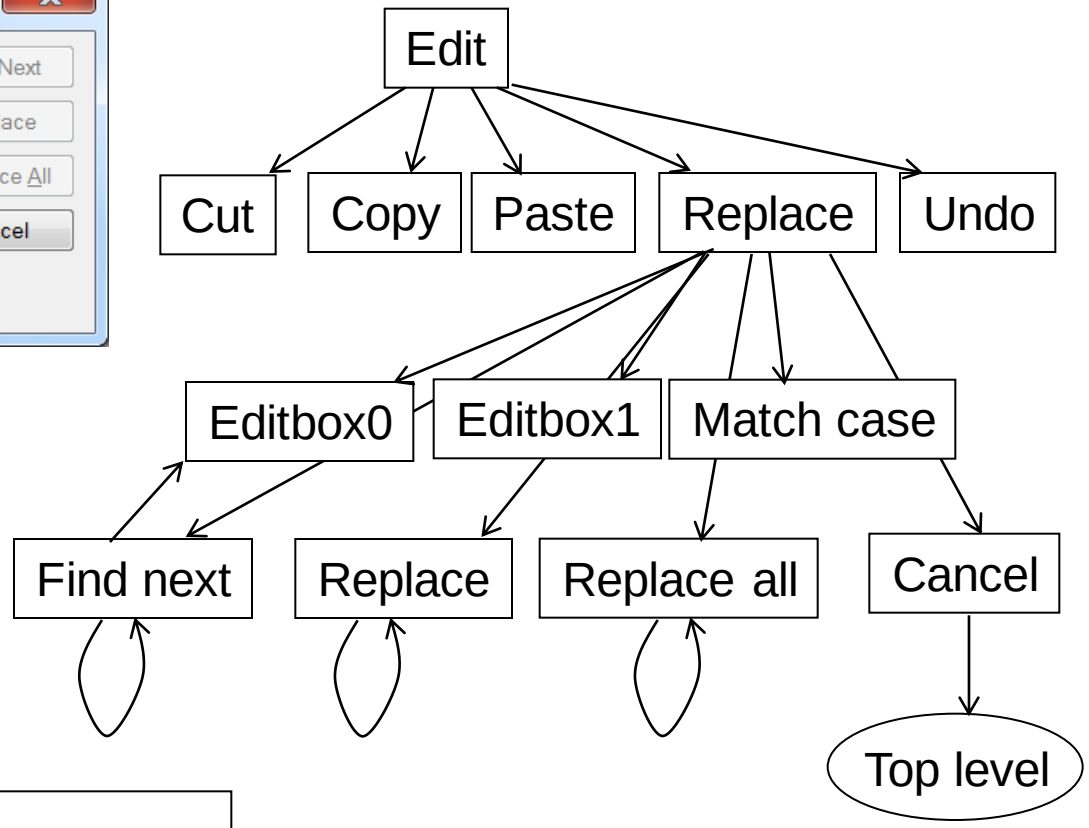
Given a GUI:

1. create a graph model of all the possible sequences that a user can execute
2. use the model to generate event sequences nekom tehnikom obilaska ovako dobijenog grafa: npr. ograničena pretraga po širini (sve sekvence dužine ≤n), random walk, ciljno usmerena pretraga (fokusirana na određenu funkcionalnost).

Graf toka događaja (EFG)

Notepad Replace funkcionalnost





TC: <S0, event1, event2, …>
Oracle: <State1, State2, …> & !CRASH

Kako konstruisati model?

- Manuelno, na osnovu specifikacije softvera
- Alernativno, automatski na osnovu same aplikacije (reverzni inženjering)
  1. Na osnovu traga izvršavanja (na nivou poziva metoda) može se konstruisati model
  2. Model može da se dotera manuelno, po potrebi

Prediktori za testirane GUI-ja

- Nije uvek lako uočiti defekte u GUI-ju
  - Često se gleda samo da li dolazi do pada aplikacije;
- Stanje GUIja može se predstaviti komponentama za koje se očekuje da budu deo GUIja u zadato vreme i njihovim stanjima tj. vrednostima
  - Na primer, pozicija prozora, GUI objekti (meni, tekst boks,...), naslov prozora - caption, slika ekrana kao bitmapa
- U test primeru za GUI, an nekorektno stanje GUI može da odvede korisnika na neočekivani ili pogrešan ekran ili onemogućiti korisnika da napravi određenu akciju;

Primer alata: Guitar

- A testing framework implementing the “Event-flow graph” approach
- Four main components:
  - The GUIRipper: extract GUI information from a program
  - The GUIStructure2Graph: build a traversable graph representation of the GUI elements
  - The TestCaseGenerator: create an extensive set of test cases based on the graph
  - The GUIReplayer: run the program as instructed by these tests
  - It is a research tool…
  - http://guitar.cs.umd.edu
    - It requires ANT to be executed

Guitar (1)

Four main components:

(1) GUIRipper: GUI information extraction

>ant -Dproperties=jfcripper.properties -f jfcripper.xml

(2) GUIStructure2Graph: Event-flow graph inference

>ant -v -f GUIStructure2GraphConvert.xml

```
- <EFG>
 - <Events>
  + <Event>
  - <Event>
      <EventId>e1</EventId>
      <WidgetId>w17</WidgetId>
      <Type>SYSTEM INTERACTION</Type>
      <Initial>true</Initial>
      <Action>edu.umd.cs.guitar.event.JFCEditableTextHandler</Action>
    </Event>
  + <Event>
  + <Event>
  + <Event>
  + <Event>
  + <Event>
  </Events>
```

```
- <EventGraph>
  - <Row>
      <E>1</E>
      <E>1</E>
      <E>1</E>
      <E>1</E>
      <E>1</E>
      <E>1</E>
      <E>0</E>
    </Row>
  + <Row>
  + <Row>
  + <Row>
  + <Row>
  + <Row>
  + <Row>
  </EventGraph>
</EFG>
```

Guitar (2)

(3) TestCaseGenerator: Test cases generation traversing the graph

>ant

```
- <TestCase>
  - <Step>
      <EventId>e1</EventId>
      <ReachingStep>false</ReachingStep>
    </Step>
  - <Step>
      <EventId>e0</EventId>
      <ReachingStep>false</ReachingStep>
    </Step>
  </TestCase>
```

(4) **GUIReplayer**: Run the program GUI according to the generated test cases

>ant -Dproperties=jfcreplayer.properties -f jfcreplayer.xml

```
- <TestCase>
  - <Step>
      <EventId>e1</EventId>
      <ReachingStep>false</ReachingStep>
    - <GUIStructure>
      - <GUI>
        + <Window>
        + <Container>
        </GUI>
      </GUIStructure>
    </Step>
  - <Step>
      <EventId>e0</EventId>
      <ReachingStep>false</ReachingStep>
    - <GUIStructure>
      - <GUI>
        + <Window>
        + <Container>
        </GUI>
      </GUIStructure>
    </Step>
  </TestCase>
```