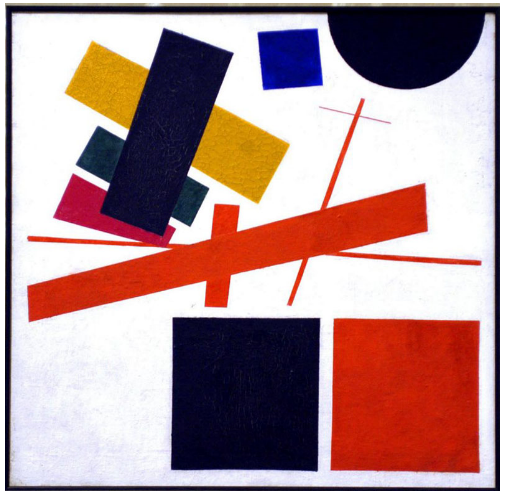
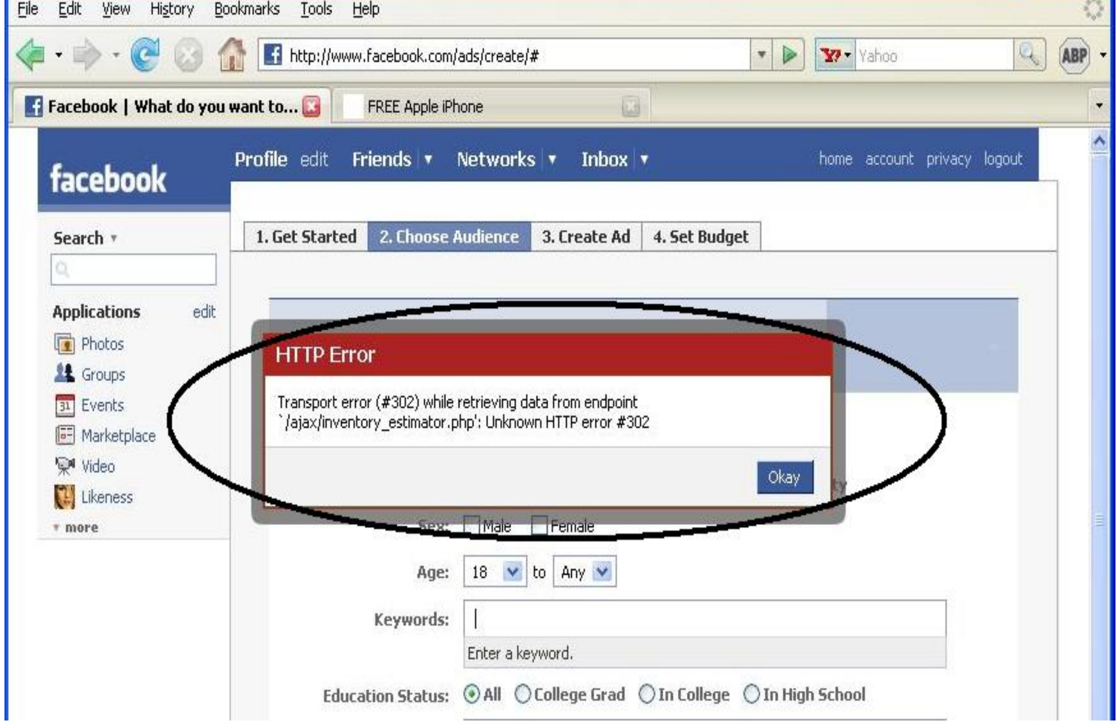
CPEN 422
Software Testing and Analysis



GUI Testing

Graphical User Interfaces (GUI)


File Edit View History Bookmarks Tools Help
http://www.facebook.com/ads/create/#
Yahoo
Facebook | What do you want to...
FREE Apple iPhone
facebook
Profile edit Friends Networks Inbox
home account privacy logout
1. Get Started
2. Choose Audience
3. Create Ad
4. Set Budget
Search
Applications edit
Photos
Groups
Events
Marketplace
Video
Likeness
more
Okay

HTTP Error

Transport error (#302) while retrieving data from endpoint `/ajax/inventory_estimator.php': Unknown HTTP error #302

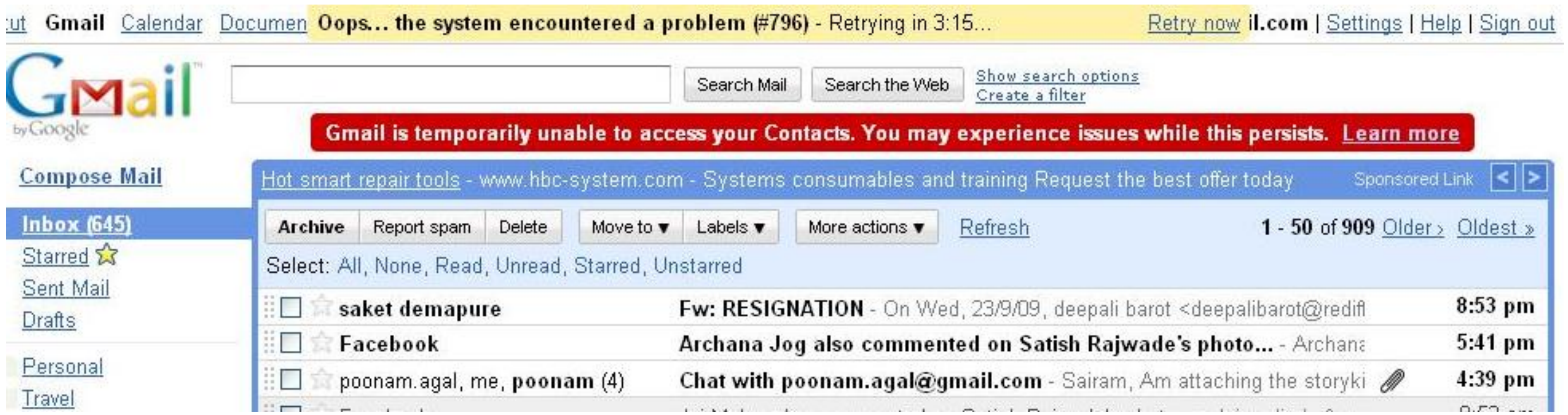
Sex: Male Female

Age: 18 to Any

Keywords:

Enter a keyword.

Education Status: [✓ All] College Grad In College In High School



Oops... the system encountered a problem (#796) - Retrying in 3:15...

Gmail is temporarily unable to access your Contacts. You may experience issues while this persists. Learn more

Hot smart repair tools - www.hbc-system.com - Systems consumables and training Request the best offer today Sponsored Link

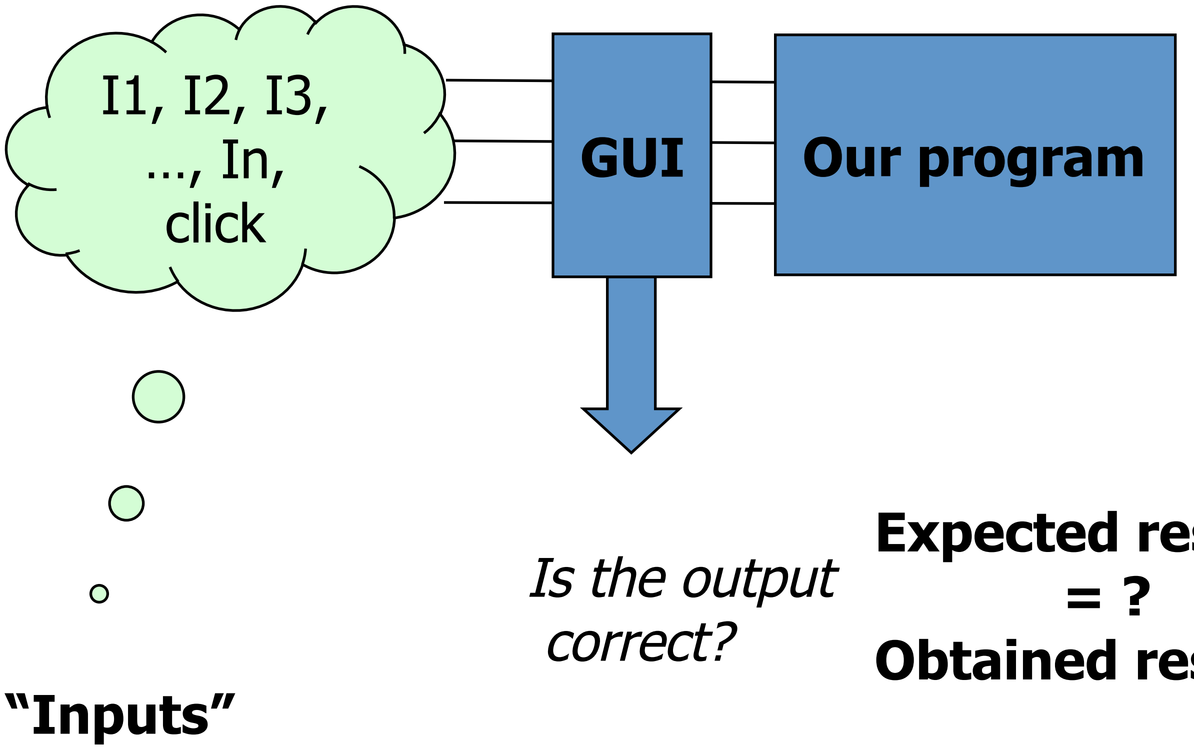
1 - 50 of 909

Select: All, None, Read, Unread, Starred, Unstarred

saket demapure	Fw: RESIGNATION - On Wed, 23/9/09, deepali barot <deepalibarot@redifl	8:53 pm
Facebook	Archana Jog also commented on Satish Rajwade's photo... - Archana	5:41 pm
poonam.agal, me, poonam (4)	Chat with poonam.agal@gmail.com - Sairam, Am attaching the storyki	4:39 pm

GUI-based Testing (Black-box)

One of the practical methods commonly used **to detect *failures*** in a program is to exercise it through its Graphical User Interface.



GUI …. what?

- GUIs are nowadays almost **ubiquitous**, even in safety critical systems
- Different types of **devices** (web, pc, tablet, mobile, watch)
- GUI responds to user **events** (e.g., clicks)
  - GUIs are event-driven systems
- GUI interacts with the underlying code by method **calls** or **messages**
- Testing GUI correctness is critical for system usability, robustness and safety
  - If the user cannot see it, it is not important!

GUI …. more formally

A GUI (**Graphical User Interface**) is a hierarchical, graphical front end to a software system.

A GUI contains **graphical objects** *w*, called widgets, each with a set of properties *p*, which have discrete values *v* at run-time.

At any time during the execution, the values of the properties of each widget of a GUI define the GUI state: {… (w, p, v), …}

A **GUI event** *e* is a state transducer, which yields the GUI from a state S to the next state S'.

How is GUI testing different from non-GUI testing?

Non-GUI testing

- test cases that invoke methods of the system and catch the return value/s
  - (for example unit tests);

GUI-based Testing

test cases that are:

- able to **identify** the components of a GUI;
- able to **exercise** GUI events (e.g., clicks);
- able to provide **inputs** to the GUI components (e.g., filling text fields);
- able to test the functionality underlying a GUI set of components (indirectly);
- able to assert the GUI properties to see if they are consistent with the expectations;

Which type of GUI-based testing?

- **System testing**
  - Test the whole system
- **Acceptance testing**
  - Accept the system
- **Regression testing**
  - Test the system w.r.t. changes

Approaches for GUI-based testing

- **Manual**
  - Based on the domain and application knowledge of the tester
- **Capture and Replay**
  - Based on capture and replay of user sessions
- **(Automated) Model-based testing**
  - Which type of model to use?
    - Event-based model
    - State-based model
    - Domain model
  - How to obtain the model to be used?
    - Manually define from specs
    - Model inferred from the application (code and runtime)
    - User session logs

Coverage criteria for GUI-based testing

- Conventional code-based coverage **cannot** be adequate;
- GUIs are implemented in terms of **event-based** systems. Mapping between GUI events and system code can not be so easy.
- **Possible coverage criteria?**
  - **Event-coverage**: all events of the GUI need to be executed at least once
  - **Widget-coverage:** all widgets of a particular state are covered at least once
  - **State-coverage**: “all states” of the GUI need to be exercised at least once
  - **Functionality-coverage**: .. using a functional point of view
  - GUI Code coverage? Might be useful

Event-based Model

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

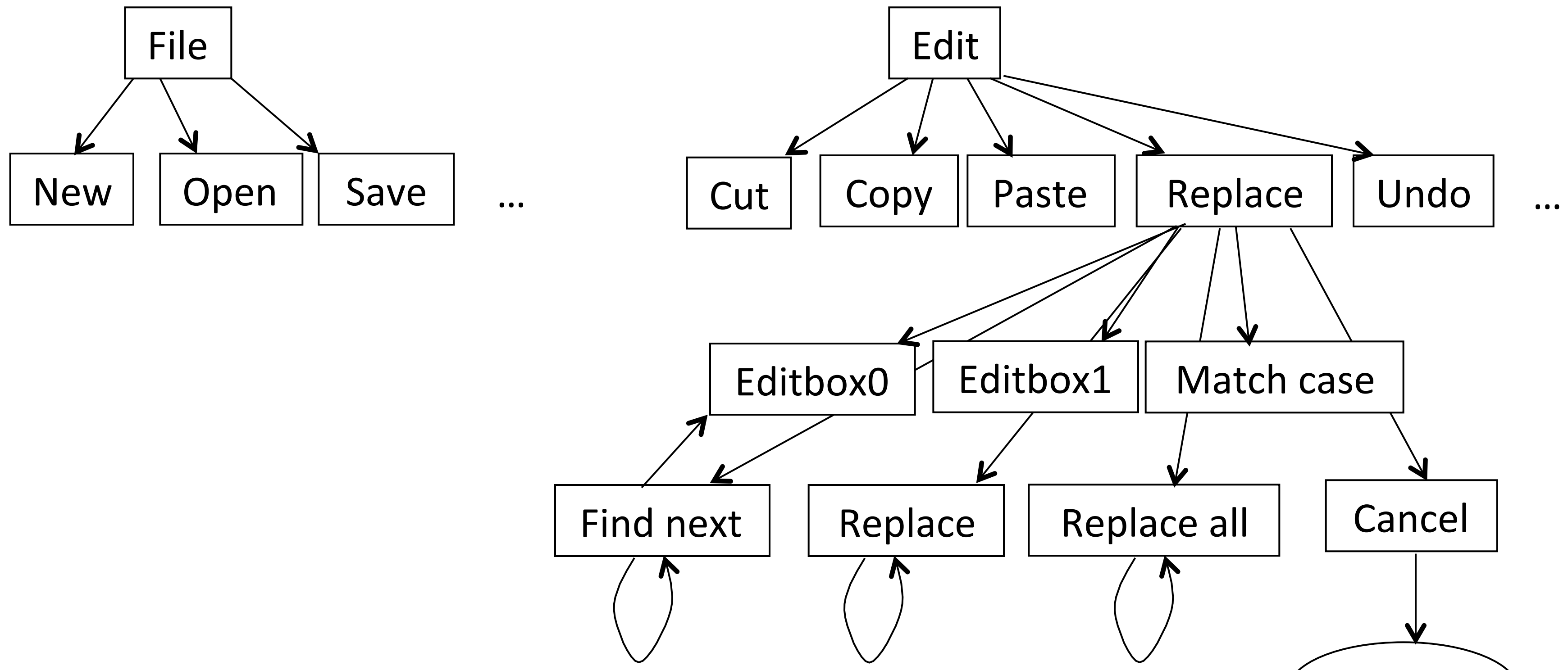
Given a GUI:

1. create a graph model of all the possible sequences that a user can execute

2. use the model to generate event sequences

Event-based Model

“Event-flow graph”



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

Event-based Models

Model Type:

- Complete event-model
- Partial event-model

Event types:

- Structural events (*Edit, Replace*)
- Termination events (*Ok, cancel*)
- System interaction events (*Editbox0, Find next*)

Coverage criteria

- Event coverage
- Event coverage according to the exercised functionality
- Coverage of semantically interacting events
- 2-way, 3-way coverage

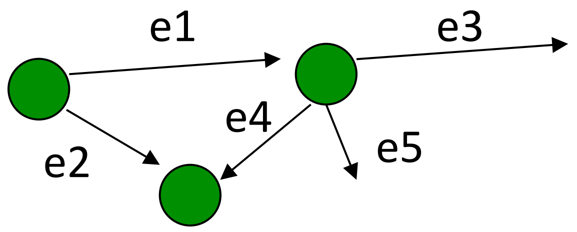
State-based Models

Model the space of **GUI event interactions as a state model**, e.g., by using a finite state machine (FSM):

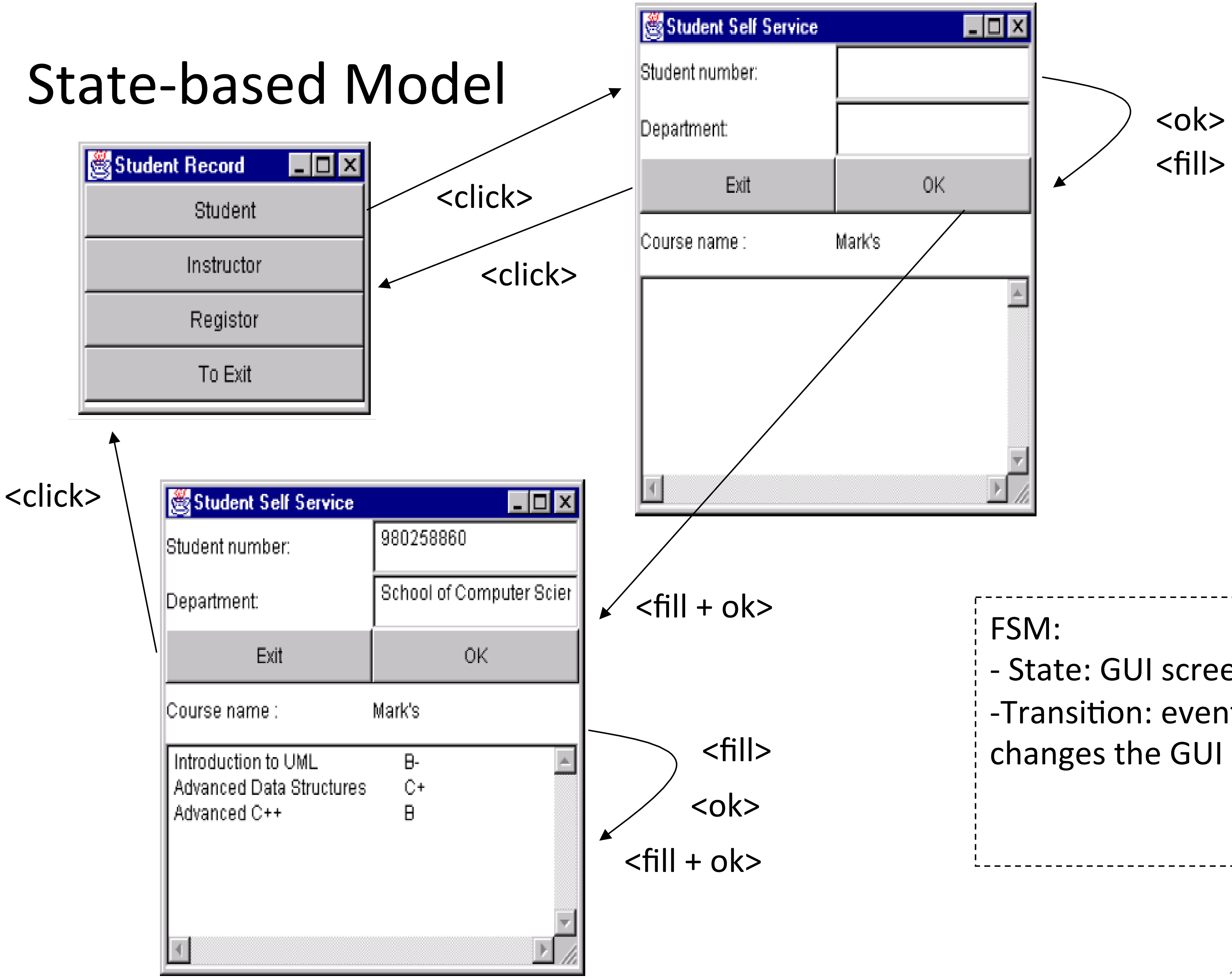
- States are distinct snapshots of the GUI
- Transitions are GUI events that change the GUI state

Given a GUI:

1. create a FSM of the possible sequences that a user can execute, considering the GUI state
2. use the FSM to generate event sequences



State-based Model



Student number: 980258860

Department: School of Computer Scier

Course name :	Mark's
Introduction to UML	B-
Advanced Data Structures	C+
Advanced C++	B

FSM:
- State: GUI screenshot
-Transition: event that changes the GUI state

Test oracles for GUI-based testing

- It could be difficult to detect faults looking at the GUI
  - Crash testing is often used in automated test generation;

- In a GUI test case, an incorrect GUI state can take the user to an **unexpected/wrong interface screen** or it can make the **user unable to do a specific action**;
  - e.g., "after the click of a button, we try to click the button again but we fail since the button no longer exists, after the first click."

- Assertions can be made on the components expected to be part of the GUI in a give time and their state/value
  - e.g., window position, GUI objects, GUI title, GUI content, GUI screenshots,

GUI errors: examples

- Incorrect action flow
  - e.g., e2 should be enabled after e1
- Missing commands
  - e.g., send email is missing
- Incorrect GUI screens/states
  - The absence of mandatory UI components (e.g., text fields and buttons)
  - Incorrect **default values** for fields or UI objects
  - Data validation errors
  - Incorrect messages to the user, after errors
  - Wrong layout (UI construction)
- ….

Capture and Replay

A capture and replay testing tool captures user sessions (user inputs and events) and stores them in scripts (one per session) suitable to be used to replay the user session.

An infrastructure is needed to intercept GUI events, GUI states, thus storing user sessions and also to be able to replay them.

Capture and Replay: the process

1. The tester interacts with the system GUI to run the system, thus generating sessions of sequence of mouse clicks, UI and keyboard events;
2. The tool captures and stores the user events and the GUI screenshots;
    - a script is produced per each user session
3. The tester can automatically replay the execution by running the script
    - the script can be also changed by the tester
    - the script can be enriched with expected output, checkpoints
    - the script can be replicated to generate many variants (e.g., changing the input values)
4. In case of GUI changes, the script must be updated

Tools for GUI-based testing

- **Marathon**
- **Abbot**
- **Guitar**

Java

- **HtmlUnit**
- **Selenium IDE**
- **CasperJS**

Web

- **Sikuli: screen-level testing**

Capture and Replay for Web applications

Selenium http://seleniumhq.org/

Selenium IDE: a Firefox add-on that will do simple record-and-playback of interactions with the browser

Selenium WebDriver: a collection of language specific bindings to drive a browser -- the way it is meant to be driven, through programming scripts (Java, JS, Python, etc)

Capture and Replay for Web: Issues

- Manual effort
- Maintenance nightmare: fragile test cases
  - Easily break after each revision

.

What are some of the challenges of GUI testing?

- Identify 5 challenges
- Hand it in!

Challenges of GUI testing

- **Locating** GUI elements is not always easy in a test case
- Generating proper **events** (drag and drop?)
- Observing and changing GUI states is expensive (time)
  - Generate event, wait for the GUI to update, but **how long**?
- Detecting **new states** is difficult (has the state really changed?)
- UI state/event **explosion** problem
  - A lot of possible states of the GUI
  - Explosion of the possible combinations and orders of events

GUI testing challenges

- GUI test **maintenance** is hard and costly
  - Non-deterministic behavior
  - GUIs are dynamic and change
  - Small structural changes can break the test case (**fragility**)
- Measuring **adequacy** of tests is problematic
  - Have we covered all states? How many states are there?
  - Have we covered all events and their combinations?
- Often GUI test automation is **technology-dependent**
  - GUI tests for Android don't work on iOS

Bad Test, why?

```
  selenium.Open(globalVars.url);
  functions.type("userId", globalVars.studName + "1");
  functions.type(”pw", globalVars.studPass + "1");
  functions.clickAndWait("signInBtn");
  selenium.click("//a[@id='discussions']/span");
  selenium.click("//a[@id='thingsToKnow']/span");
…
  selenium.waitForElementVisible("//div[@id='TTHContainer']/
ul[1]/li[1]");
…
```

Better Test

```
public void testByDateTab() {
 funtions.loginMobi();
 selenium.click("//a[@id='discussions']/span");
 selenium.click("//a[@id='thingsToKnow']/span");
 functions.verifyThingsToKnow();
 functions.verifyThingsHappeningSoon();
 selenium.Select("byDateFilter", "label=Things That
   Happened");
 functions.verifyThingsThatHappened();
}
```

Good Tests

```
LoginPage loginPage = new LoginPage(selenium);
HappeningsPage happeningsPage =
  loginPage.loginAs(Common.stud1Name,
  Common.stud1Password);
happeningsPage.waitToLoad();

Assert.That(!happeningsPage.isByTypePageLoaded());
Assert.That(happeningsPage.isByDatePageLoaded());
…
```

The PageObject: mitigating fragility

PageObjects: “Within your web app's UI there are areas that your tests interact with. A Page Object simply models these as objects within the test code. This reduces the amount of duplicated code and means that if the UI changes, the fix need only be applied in one place.”

Required Reading:
https://code.google.com/p/selenium/wiki/PageObjects

The Page Object:

```
public class HappeningsPage  {
    private String loadingImage = "//
    span[@class='ajaxLoadingHeader']";
    private String moreLink = "more";

    public HappeningsPage(Selenium selenium)
    {
       this.selenium = selenium;
       this.title = "Happenings";
       this.url = "index.html";
       assertPageLoadedCorrectly();
    }

    public HappeningsPage waitToLoad()
    {
       waitForElementNotVisible(_loadingImage );
       return new HappeningsPage(selenium);
    }
   }
```

```
public ContentItemPage goToItem(string type, string name)
    {
       click("//div[@id='" + type + "']//span[text()=\"" + name
    "\"]");
       return new ContentItemPage(selenium);
    }
    public HappeningsPage clickMoreLink()
    {
       click(moreLink);
       return new HappeningsPage(selenium);
    }

    public HappeningsPage filterResults(string id, string nam
    {
       selectDropdown(id, name);
       return new HappeningsPage(selenium);
    }
}
```