中山大學
SUN YAT-SEN UNIVERSITY

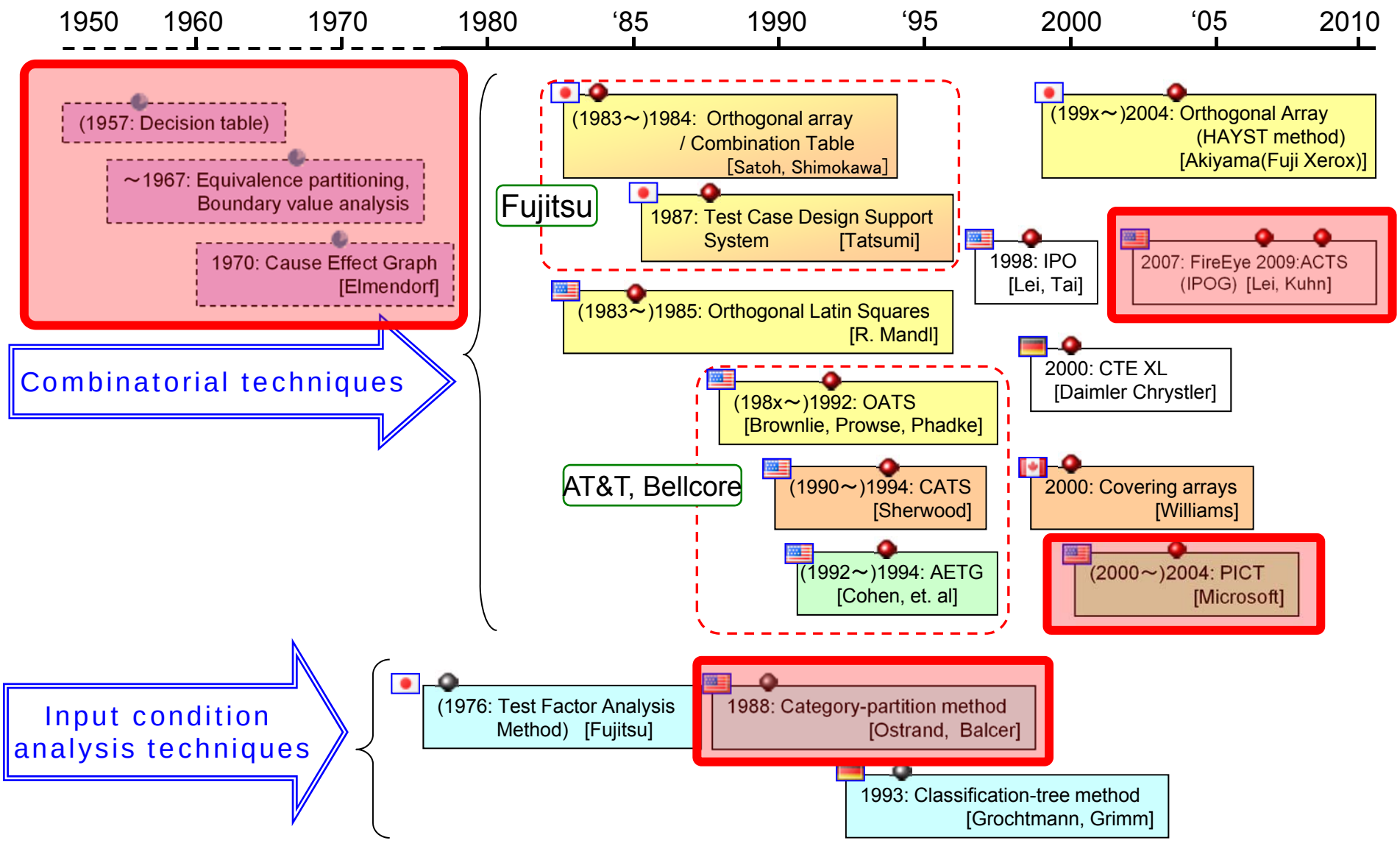
Part II [Problem]
5. Test Adequacy (Blackbox, level-3)

SE-307 Software Testing Techniques

http://my.ss.sysu.edu.cn/wiki/display/SE307/Home

Instructor: Dr. Wang Xinming, School of Software, Sun Yat-Sen University

Review: Blackbox Test Adequacy

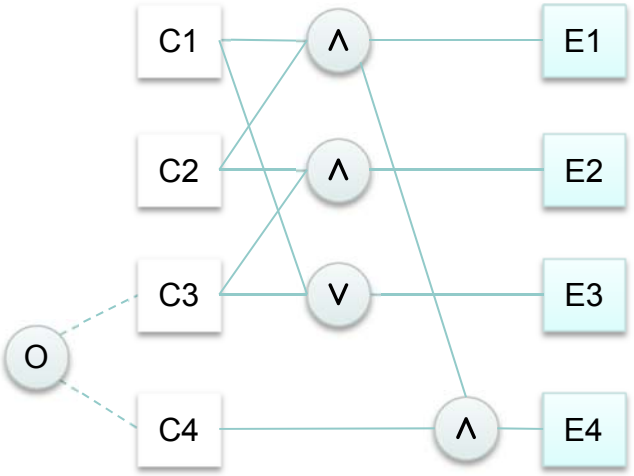


Review: Level-2 Techniques

- From the values of one single parameter to the value combinations of multiple parameters.
  - The problem: too many combinations to cover all of them.
- Combinatorial Coverage
  - ACTS
  - PICT
  - Category-partition Testing (TSL)
- Constrained Combinational Coverage
  - Why we introduce constraints? Rule out infeasible and redundant combinations.
  - Property constraint
  - Single/error constraint

Review: Level 1 vs. Level 2

- Cause effect graph vs. Category partition testing
  - **What are their similarities and differences?**



C1	T	F	T	T	T	F	F
C2	T	T	F	T	T	F	F
C3	F	T	T	T	F	F	T
C4	T	F	F	F	F	T	F
E1	•			•			
E2		•		•	•		
E3	•	•	•				•
E4	•						

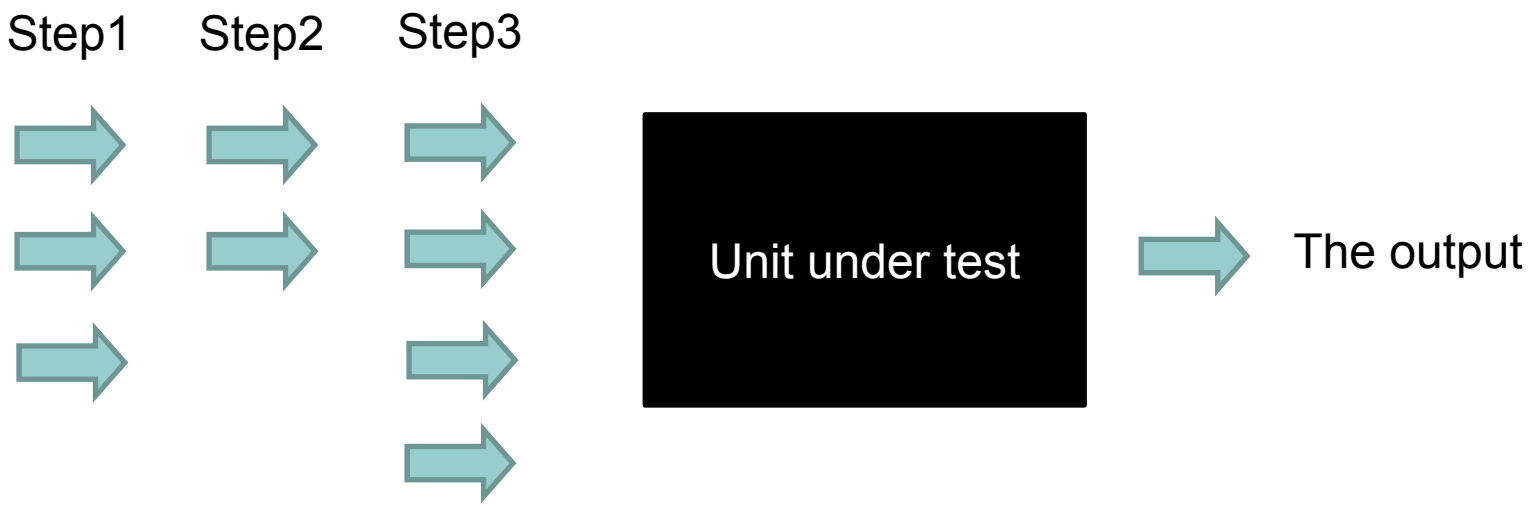
```
Parameters:
    Pattern size:
        empty                                   [property Empty]
        single character                        [property NonEmpty]
        many character                          [property NonEmpty]
        longer than any line in the file        [error]

    Quoting:
        pattern is quoted                       [property Quoted]
        pattern is not quoted                   [if NonEmpty]
        pattern is improperly quoted            [error]

    Embedded blanks:
        no embedded blank                       [if NonEmpty]
        one embedded blank                      [if NonEmpty and Quoted]
        several embedded blanks                 [if NonEmpty and Quoted]

    Embedded quotes:
        no embedded quotes                      [if NonEmpty]
        one embedded quote                      [if NonEmpty]
        several embedded quotes                 [if NonEmpty] [single]

    File name:
        good file name
        no file with this name                  [error]
        omitted                                 [error]

Environments:
    Number of occurrences of pattern in file:
        none                                    [if NonEmpty] [single]
        exactly one                             [if NonEmpty] [property Match]
        more than one                           [if NonEmpty] [property Match]

    Pattern occurrences on target line:
    #  assumes line contains the pattern
        one                                     [if Match]
        more than one                           [if Match] [single]
```

Review: Similarity and Difference

- Similarities
  - Address the same problem: inputs are subject to complex conditions.
  - Causal-effect graph considers the combinations of *conditions*, while category-partition testing considers the combinations of *choices*.
    - **However, choices and conditions are basically the same thing.**
  - Both use constraints to rule out infeasible combinations.

- Differences:
  - Causal-effect graph use cause-effect relation to rule out redundant combinations while category-partition testing use property constraints.
  - Category-partition testing tries to cover all feasible combinations, while causal-effect graph does not require covering all combinations.

Level-3: Adequacy for Sequence



Why Sequence?

- Many faults are triggered by the order of input events.
  - '*failure occurred when A started if B is not already connected*'.
- Real world example: A fault in the GPS system for a car
  - plug in GPS; ignition off; ignition on; boot screen; unplug GPS -> **screen locks**

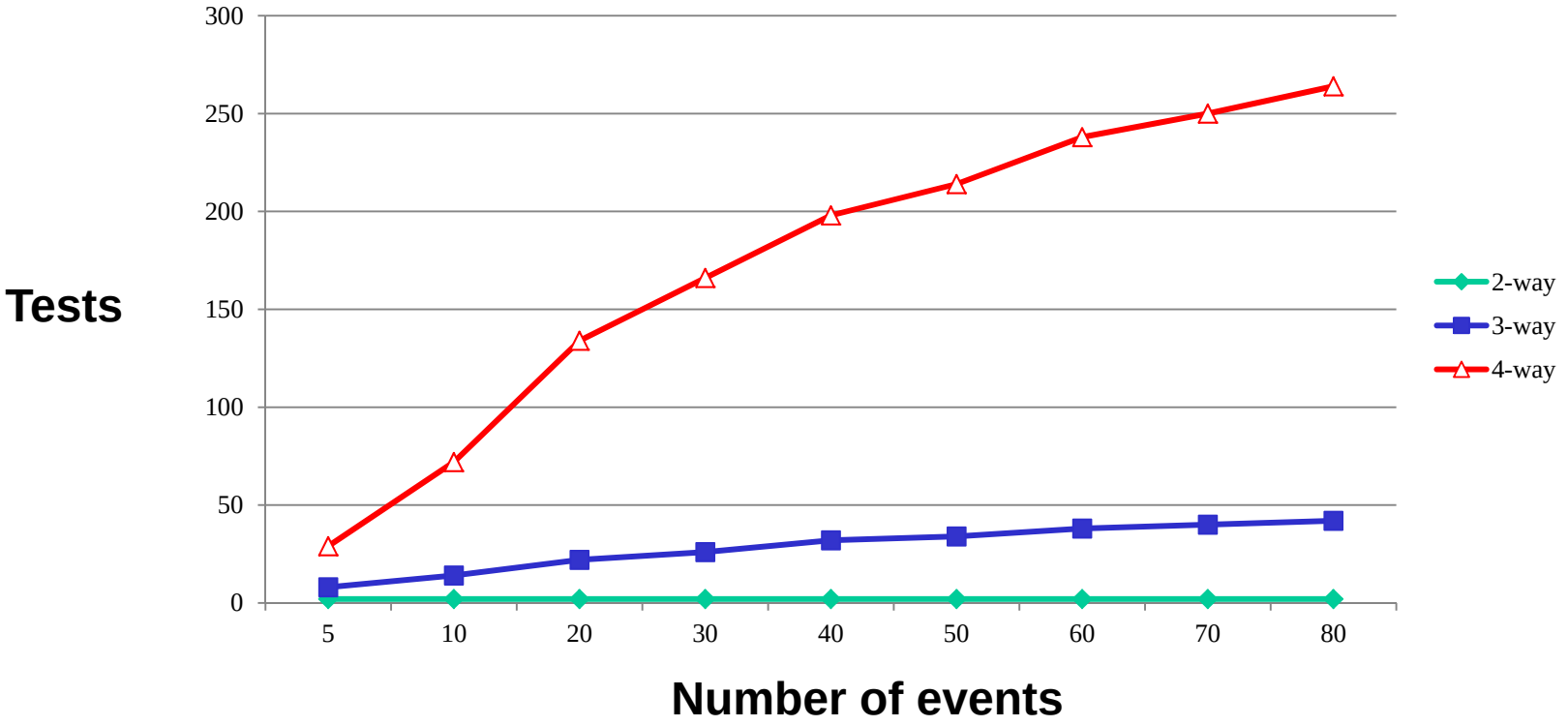
Sequence Coverage for Simple Event

- Extension of combinatorial coverage:
  - T-wise Combinatorial Coverage ⇒ T-wise Combinatorial Sequence Coverage
  - T-way interaction ⇒ T-way sequence
- Suppose we have 6 simple events: all sequences = 6! = 720 tests
  - Only 10 tests are needed for all 3-way sequences

Test	Sequence					
1	a	b	c	d	e	f
2	f	e	d	c	b	a
3	d	e	f	a	b	c
4	c	b	a	f	e	d
5	b	f	a	d	c	e
6	e	c	d	a	f	b
7	a	e	f	c	b	d
8	d	b	c	f	e	a
9	c	e	a	d	b	f
10	f	b	d	a	e	c

Sequence Covering Array Properties

- 2-way sequences require only 2 tests
  - Write events in any order, then reverse.
- For any t, number of test cases required to cover all t-ways sequences grows with log(n), for n events

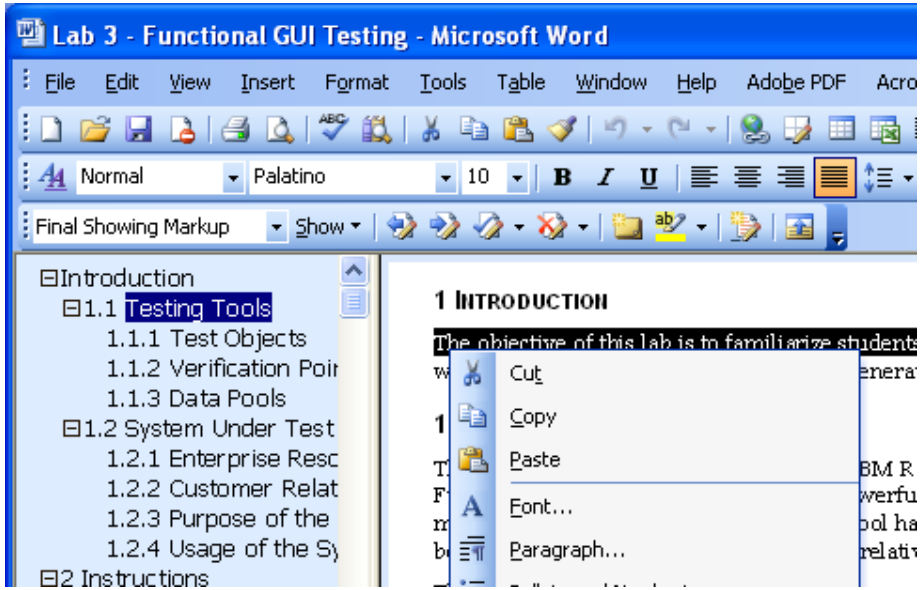


Sequence Coverage for GUI

- Input to a GUI program inherently involve a sequence of parameter combinations
  - A GUI is a **hierarchical**, **graphical** front end to a software system
  - GUI allows the user to interact with the software systems.
  - GUI are nowadays almost ubiquitous.
- Example: login → select items → checkout → payment
  - **Login Dialog** inputs four parameters: user name, password, server address, “remember-me” checkbox
  - **Select Items Window** inputs one complex parameter: a list of items the user want to buy.
  - **Checkout Window** inputs five parameters: the delivery address, the delivery date, the phone number of the user, the payment method (cash, credit card, or online), whether coupons are used.
  - Depending on which payment method is chosen, the **Payment Window** inputs different set of parameters.

Challenges of GUI Testing (1)

Complex event, many sequences



- GUI is event-driven
- The event-driven nature of GUIs presents the first serious testing difficulty.
- Because users many click on any pixel on the screen, there are many, many more possible user inputs that can occur.
- The user has an extremely wide choice of actions.
- At any point in the application, the users may click on any field or object within a window.

Challenges of GUI Testing (2)

- Complex dependencies between events
  - If a checkbox is set to true, a text box intended to accept a numeric value elsewhere in the window may be made inactive or invisible.
  - If a radio button is clicked, a different validation rule might be used for a data field elsewhere on the window.
  - A window contains “date of last order”. In a different window, user submits an order. Should the “date of last order” be updated?
  - Where are these dependencies?

Challenges of GUI Testing (3)

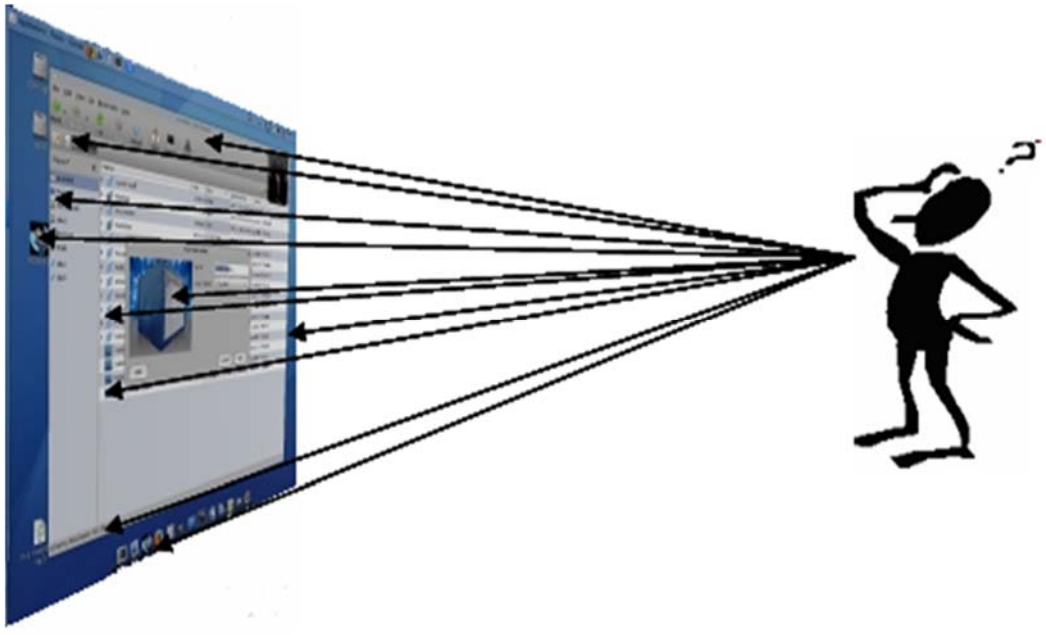
- Unsolicited events may occur at any time
  - Printer goes off-line → dialog box from OS
  - Message from some middleware component to "redraw a diagram"
- Testing unsolicited events is difficult
  - May need special test drivers
  - Need to know about all unsolicited events

Challenges of GUI Testing (4)

- Typically, there are multiple ways to achieve the same result.
  - Keyboard shortcut
  - Menu options
  - Click on another window
- Multiple ways of selecting options
  - Keyboard shortcuts
  - Function keys
  - Mouse movements (buttons or menus)
- Should the feature be tested 3 times over?

We Need a Model of GUI

- What is a model of GUI?
  - From programmer's perspective
  - From tester's perspective

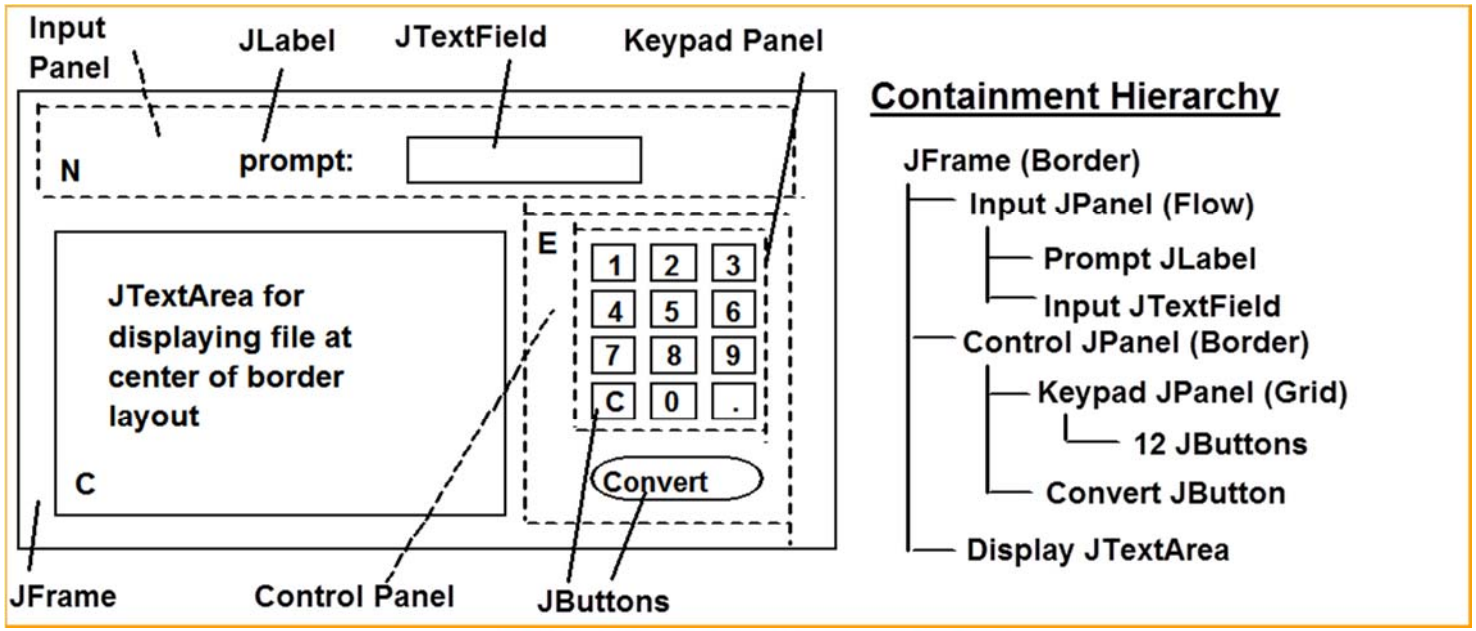


GUI from Programmer's Perspective

- A GUI programs contains **GUI objects**
  - **Widgets** in X-Windows: Gnome, KDE, …
  - **Controls** in Windows: ActiveX, MFC, Form, …
  - **Components** in Java: AWT, Swing, SWT, …
- Two most important parts when you write a GUI program
  - GUI Object library
  - Even-driven execution model

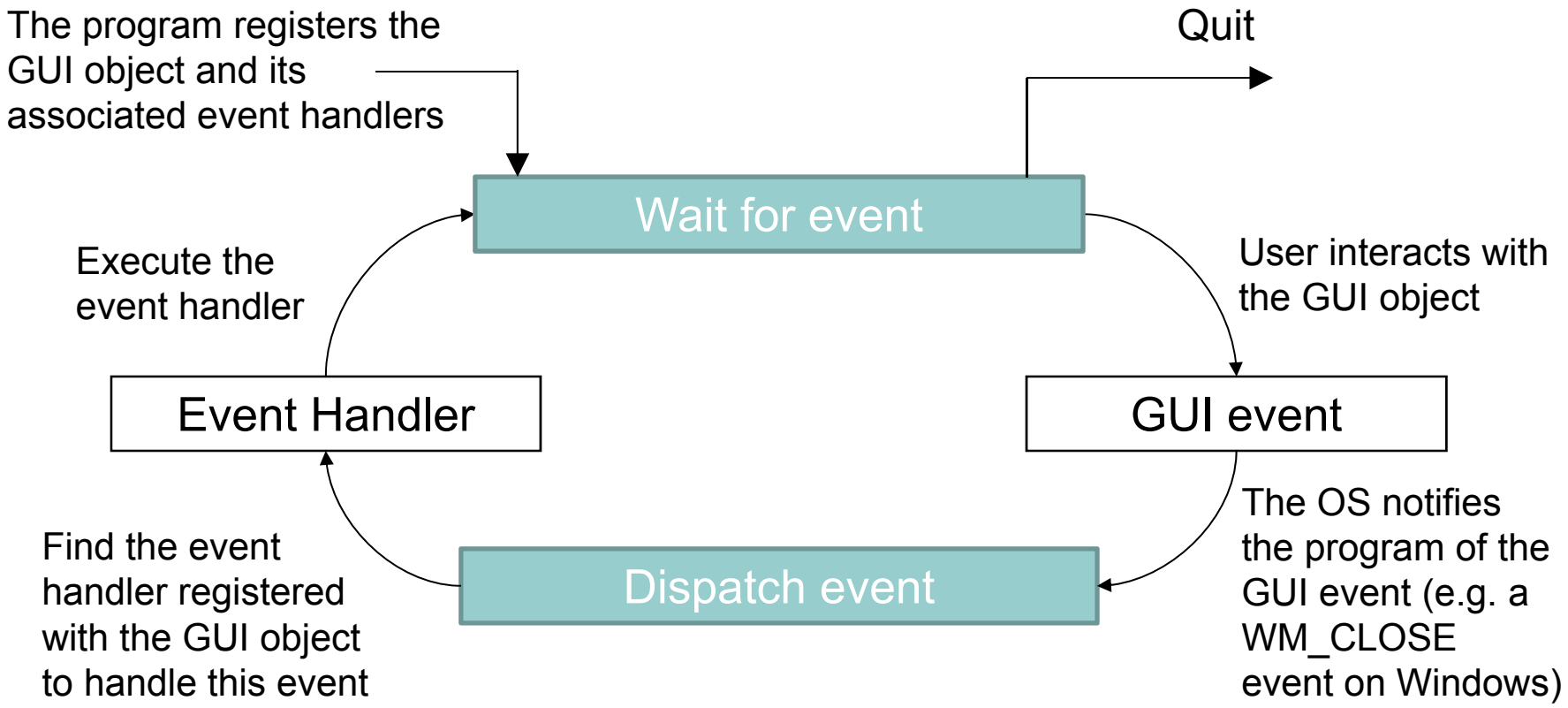
GUI Object Library

- A library of pre-defined GUI objects, each with a pre-defined set of *properties*, *methods*, and *events*.
  - Example: Menu, Button, Label, TextArea, Slidebar, Canvas, …
- There are two hierarchy relations
  - Containment: e.g. a button is contained in a panel
  - Inheritance: e.g. JMenuItem inherits from JAbstractButton



Event-driven Execution Model

- Order of execution is governed by user
- Program responds to events generated by user interaction with GUI objects
- Developers write *event handlers* to implement the application logic.



Example: A GUI Program

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class CountButtonPushes extends JFrame
                    implements ActionListener {
   private JButton button ;
   private JLabel total;
   private JTextField tally;
   private int sum = 0;
   public CountButtonPushes() {
      super("A Container With Components");
      setSize(500,500);
      button = new JButton("Press me");
      total = new JLabel( "Running total:");
      tally  = new JTextField(10)
      Container cp = getContentPane();
      cp.setLayout(new FlowLayout());
      cp.add(total);
      cp.add(tally);
      cp.add (button);
      button.addActionListener(this);
      show();
   }

   public void actionPerformed(ActionEvent e ) {
      sum = sum + 1;   // add number to sum
      tally.setText(Integer.toString(sum));
   }

   public static void main(String args [] ) {
         new CountButtonPushes();
   }
}
```

Create and register the GUI objects

Establish the containment relation

Register event handler

Handle the event

Start with the main function

GUI from tester's perspective

- Three different points of view on a GUI program from a tester's perspective
  - GUI program = a set of code units
  - GUI program = a set of events
  - GUI program = a set of features
- The first one: use previous whitebox coverage criteria
  - statements, basic block, basis path, …
- The latter two lead to two new sets of blackbox coverage criteria dedicated to GUI testing.

Level-3 Coverage Criteria

- **Event–based Coverage （基于事件的覆盖）**
  - **GUI program = a set of events**

- **State-based Coverage（基于状态的覆盖）**
  - **GUI program = a set of features**

GUI State

- The state of a GUI is modeled using:
  - GUI Objects O = {o_1, o_2, o_3, …, o_m}
  - Properties P = {p_1, p_2, p_3, …, p_l}, where p_i is an n_i-ary (n_i >= 1) ***Boolean*** relation of the form:

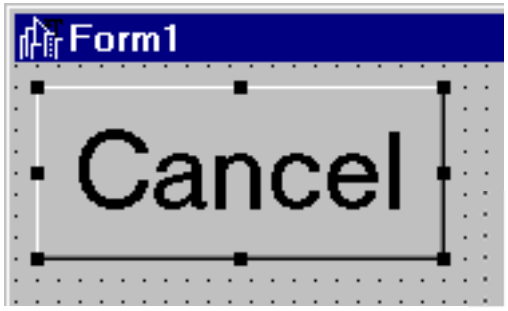
Property(o_1, o_2, o_3, …,o_k, value)

True/False

Property Name

Objects

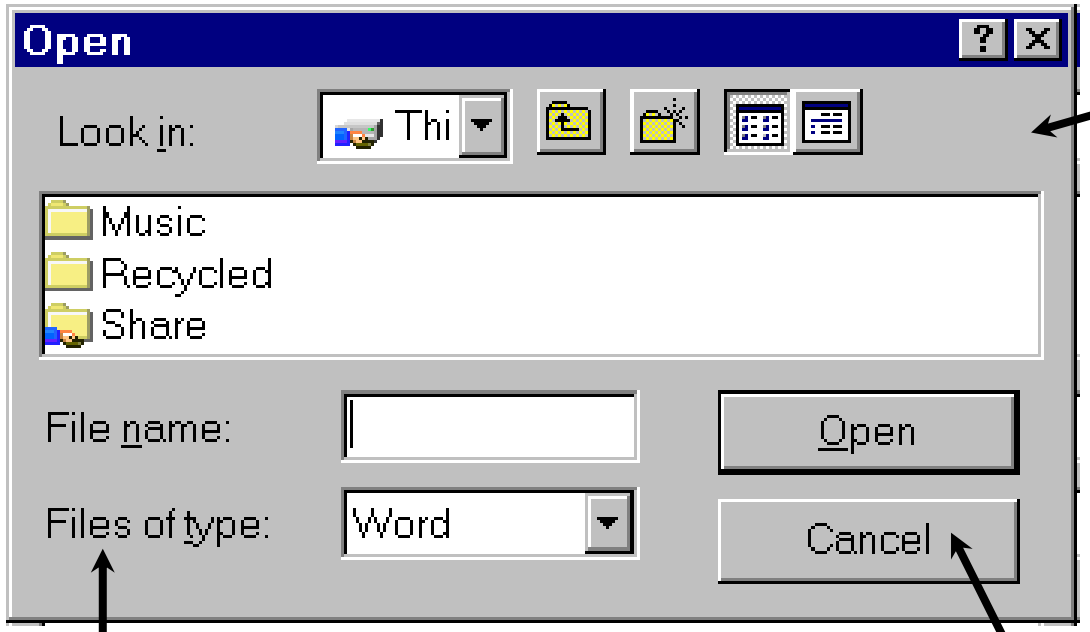
Optional value of Property



Caption(Button, “Cancel”)

GUI’s state: S = {p_1, p_2, p_3,…,p_n}

Example



Form1

WState(Form1, wsNormal)
Width(Form1, 1088)
Scroll(Form1, TRUE)

Label1

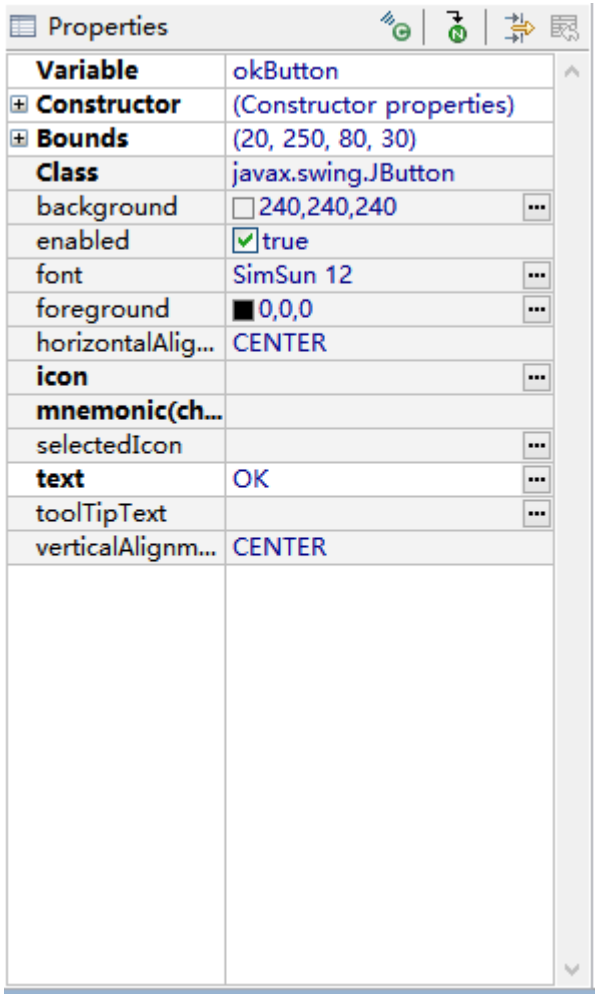
Align(Label1, alNone)
Caption(Label1, “Files of type:”)
Color(Label1, clBtnFace)
Font(Label1, (tfont))

Button1

Caption(Button1, “Cancel”)
Enabled(Button1, TRUE)
Visible(Button1, TRUE)
Height(Button1, 65)

Determining Properties of Objects

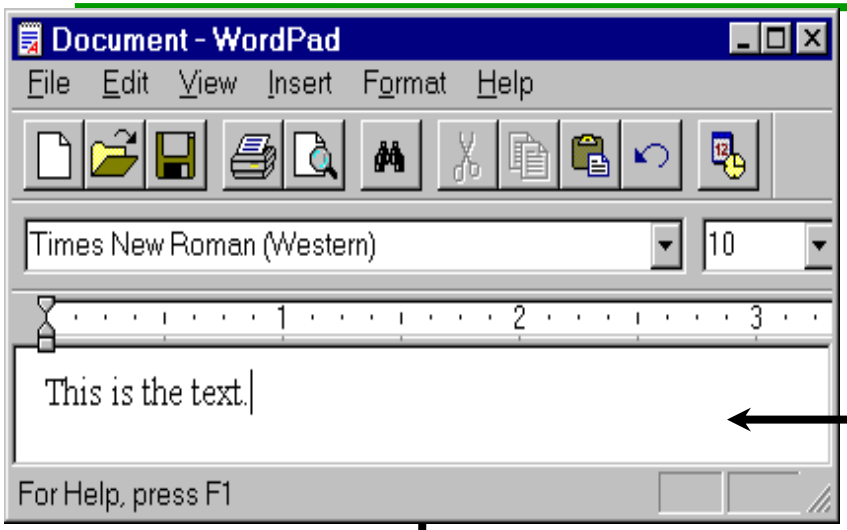
- Manual Examination
- Specifications (reduced set)
  - GUI being tested
- Toolkit/language (complete set)
  - All available properties



GUI Events

- Events change the GUI's state
- Events E = {e_1, e_2, e_3, …, e_n}, associated with a GUI are functions from one GUI state S_i to another state S_j
- Notation: $S_j = e_i(S_i)$

Example: An Event



$S_j = e(S_i)$

State: S_i

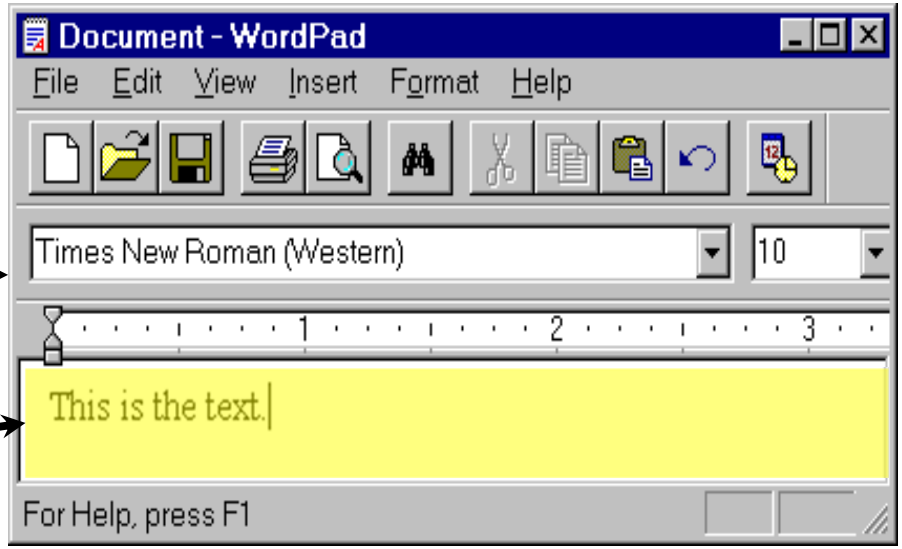
Background color is ***not*** yellow

w19

set-background-color (w19, yellow)

Event: **e**

Background color is yellow



State: S_j

Representing Events

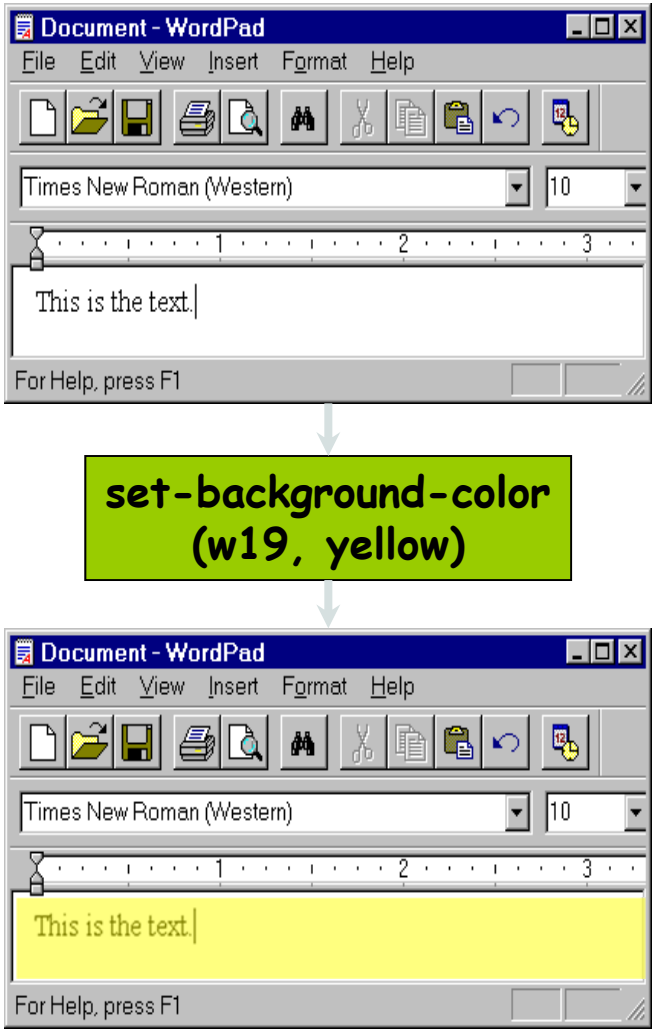
- There can be infinitely many states of GUI
  - Infeasible to give enumerate events as a state mapping function.

- Model the GUI events using **event operators**, which specify their preconditions and effects

(Event) Operators

- An Operator is a triple
  - <Name, Precondition, Effects>
  - Name identifies an operator and its parameters
  - Precondition is a set of positive literals
  - Effects is a set of positive or negative literals
- Operator Op is applicable in any state S_i in which:
  - All the literals in Precondition(Op) are TRUE
- The resulting state S_j is determined by using Effects(Op)
  - All positive literals in Effects(Op), and
  - All literals that were TRUE in S_i, except
  - Those that are negative in Effects(Op)

Operator Example



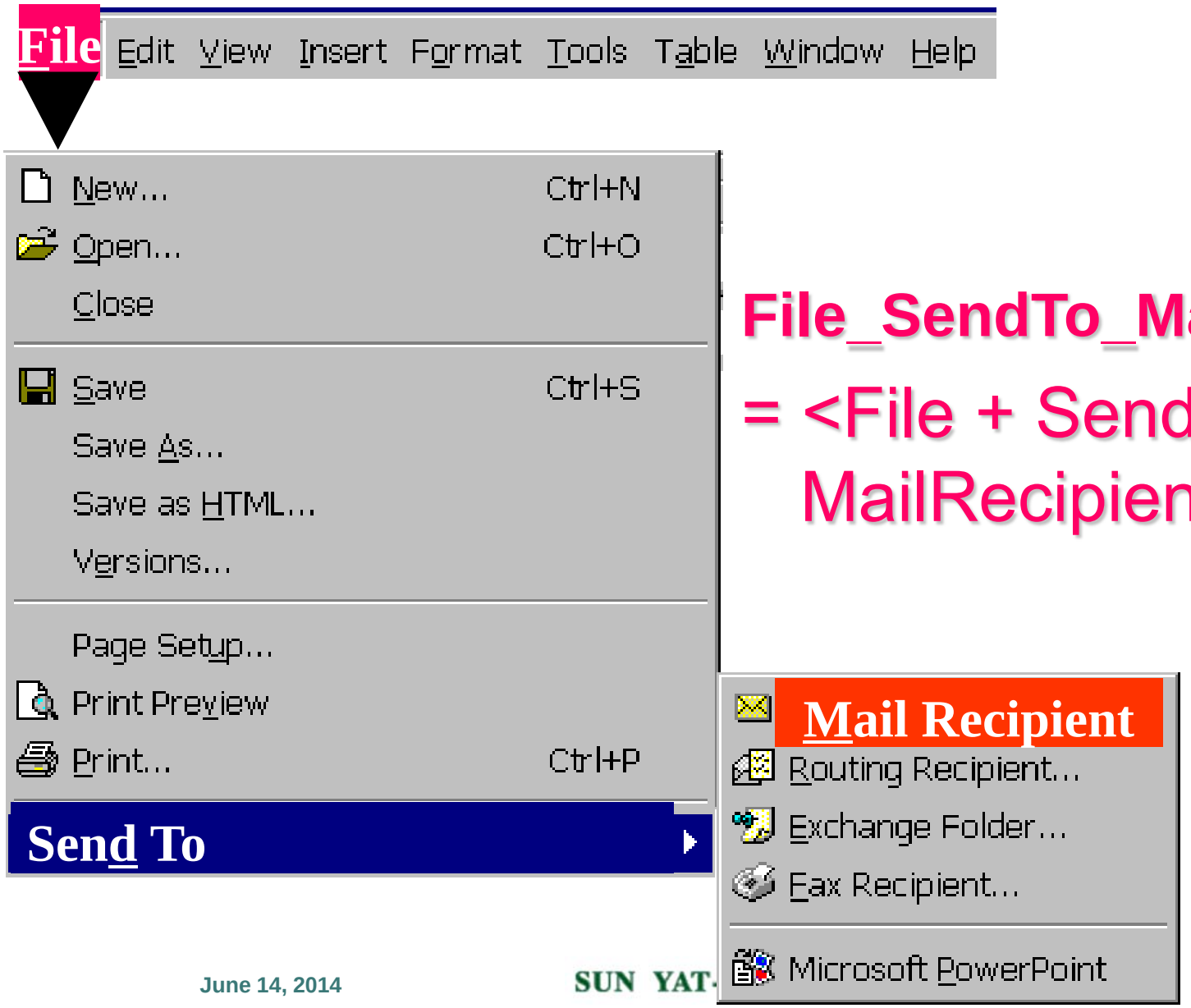
Operator :: *set-background-color*
Parameters: wX: window; col: color;
Precondition:
isCurrent(wX),
background-color(wX, oldColor),
oldColor != col.

Effects:
background-color(wX, col).

Exploit the GUI's Structure

- Use operator abstraction to reduce the number of operators

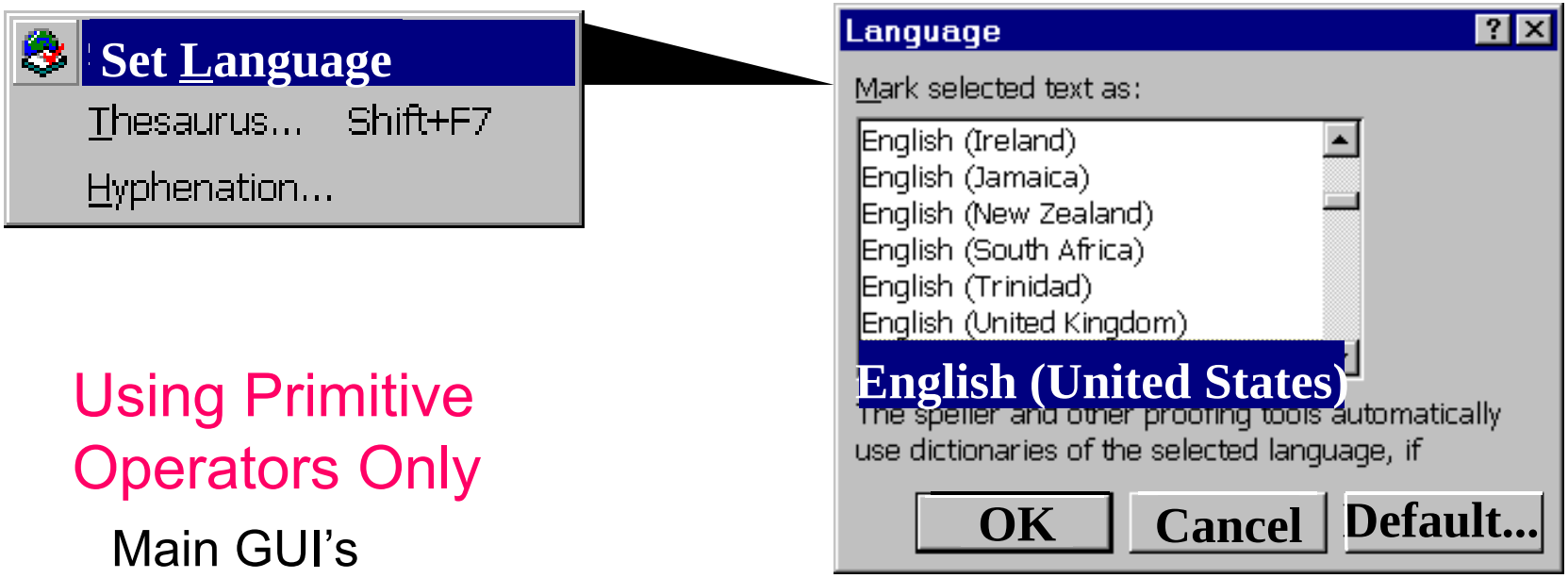
Operator abstraction



File_SendTo_MailRecipient

= <File + SendTo + MailRecipient >

Operator abstraction



Using Primitive Operators Only

Main GUI's Operator Set

…
Set Language
SelectFromList()
Default
OK
Cancel
...

Using Abstraction

Main GUI's Operator Set

…
Set Language
...

Language Window's Operator Set

SelectFromList()
Default
OK
Cancel

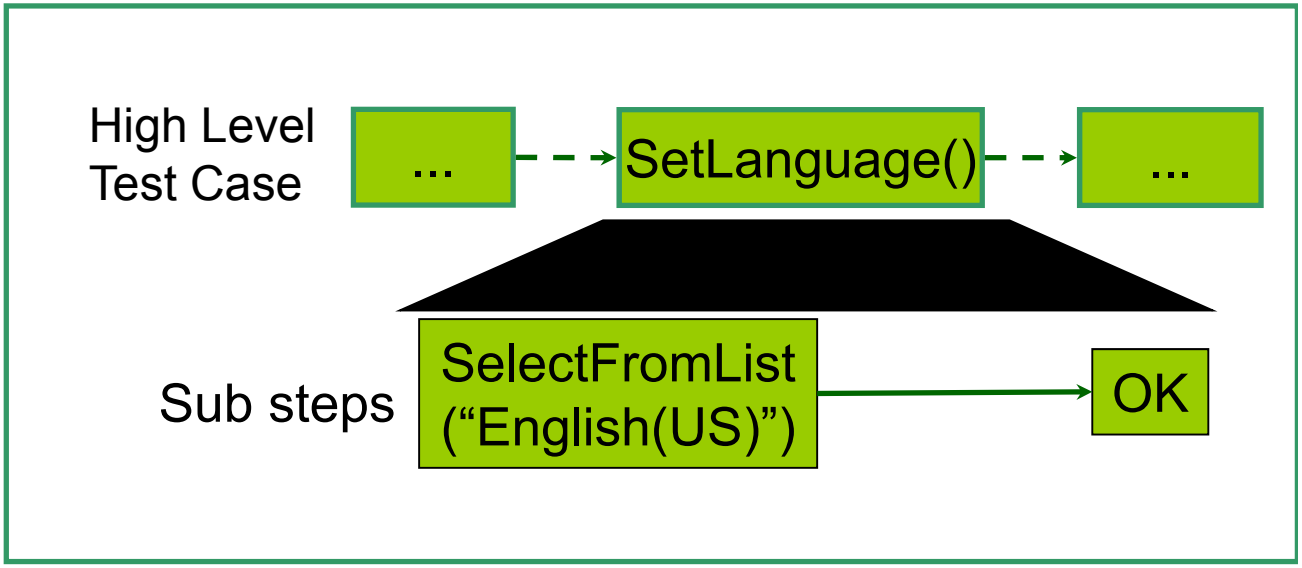
Operator abstraction

Language Window's Operator Set

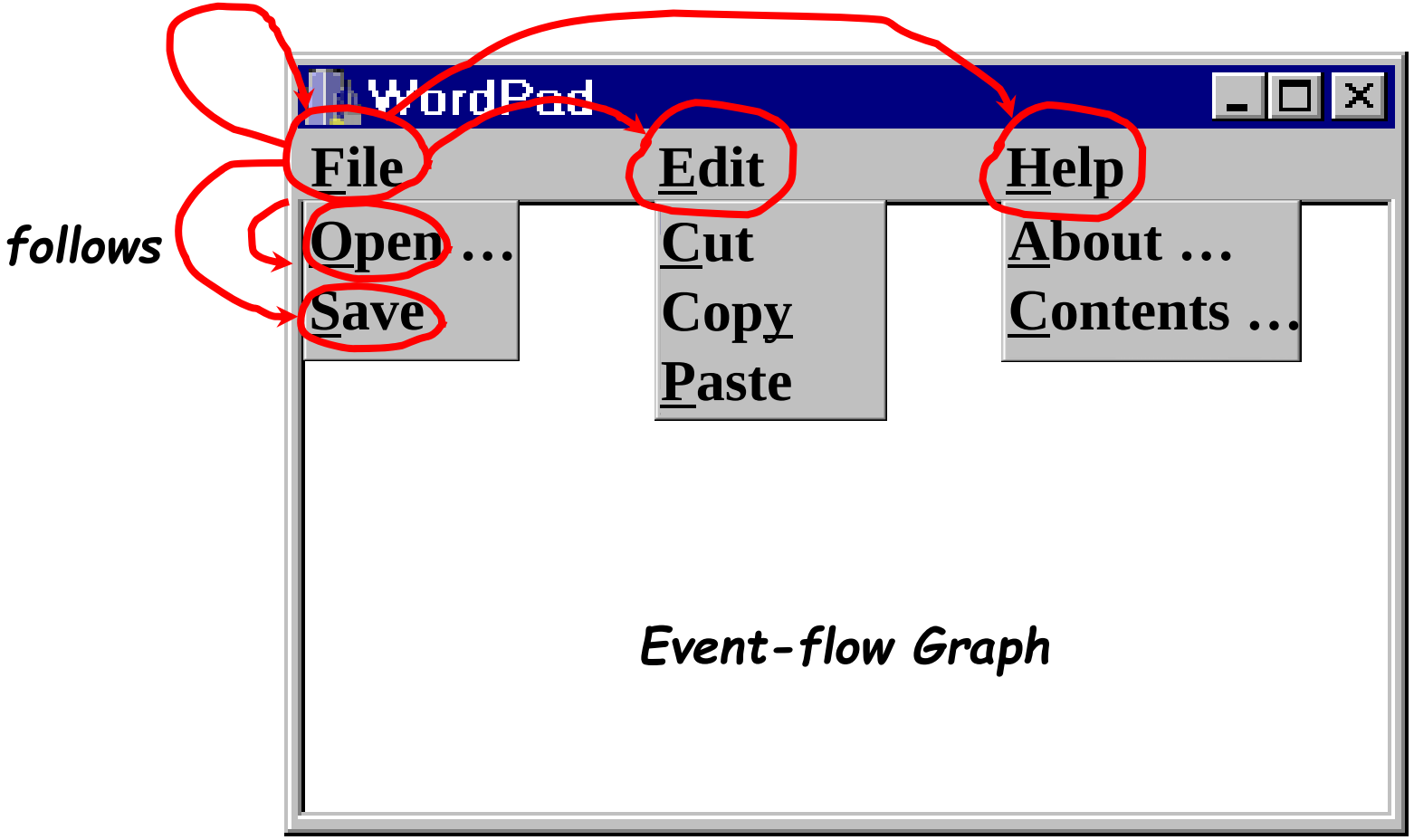
SelectFromList()
Default
OK
Cancel

Define Abstraction

SetLanguage()

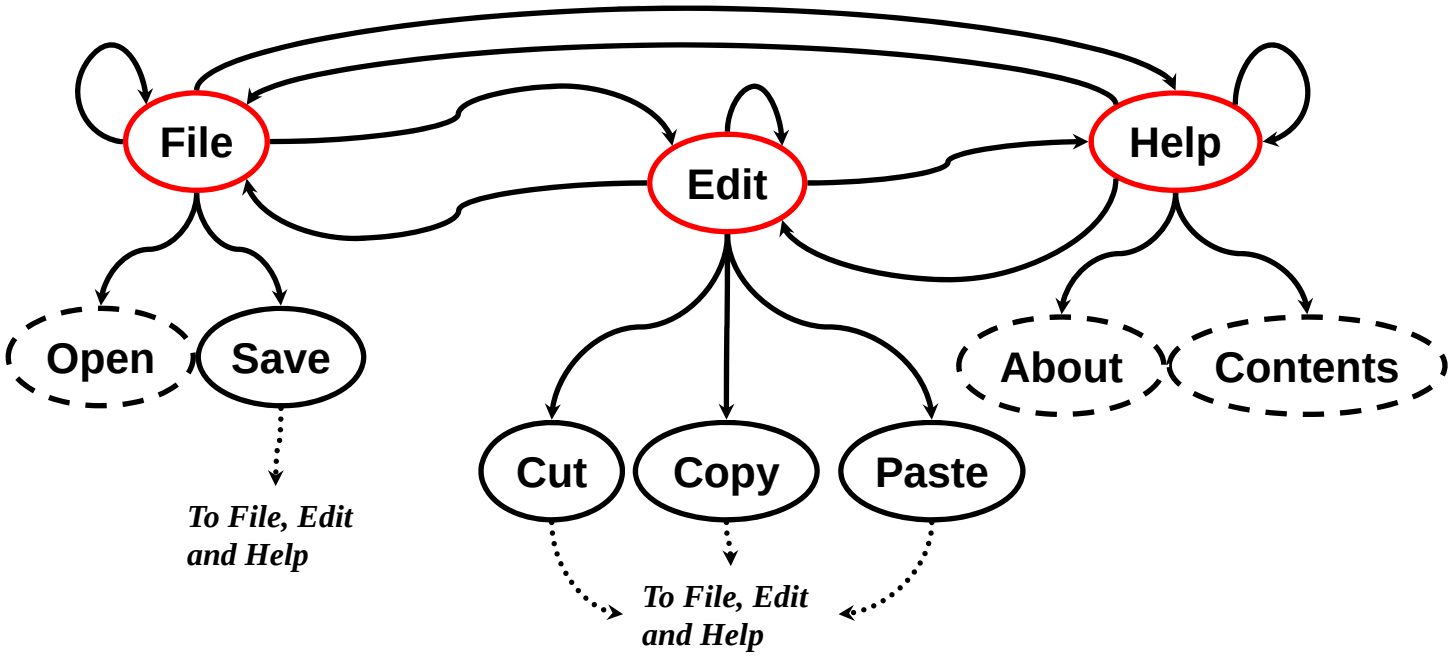


Representing a Window



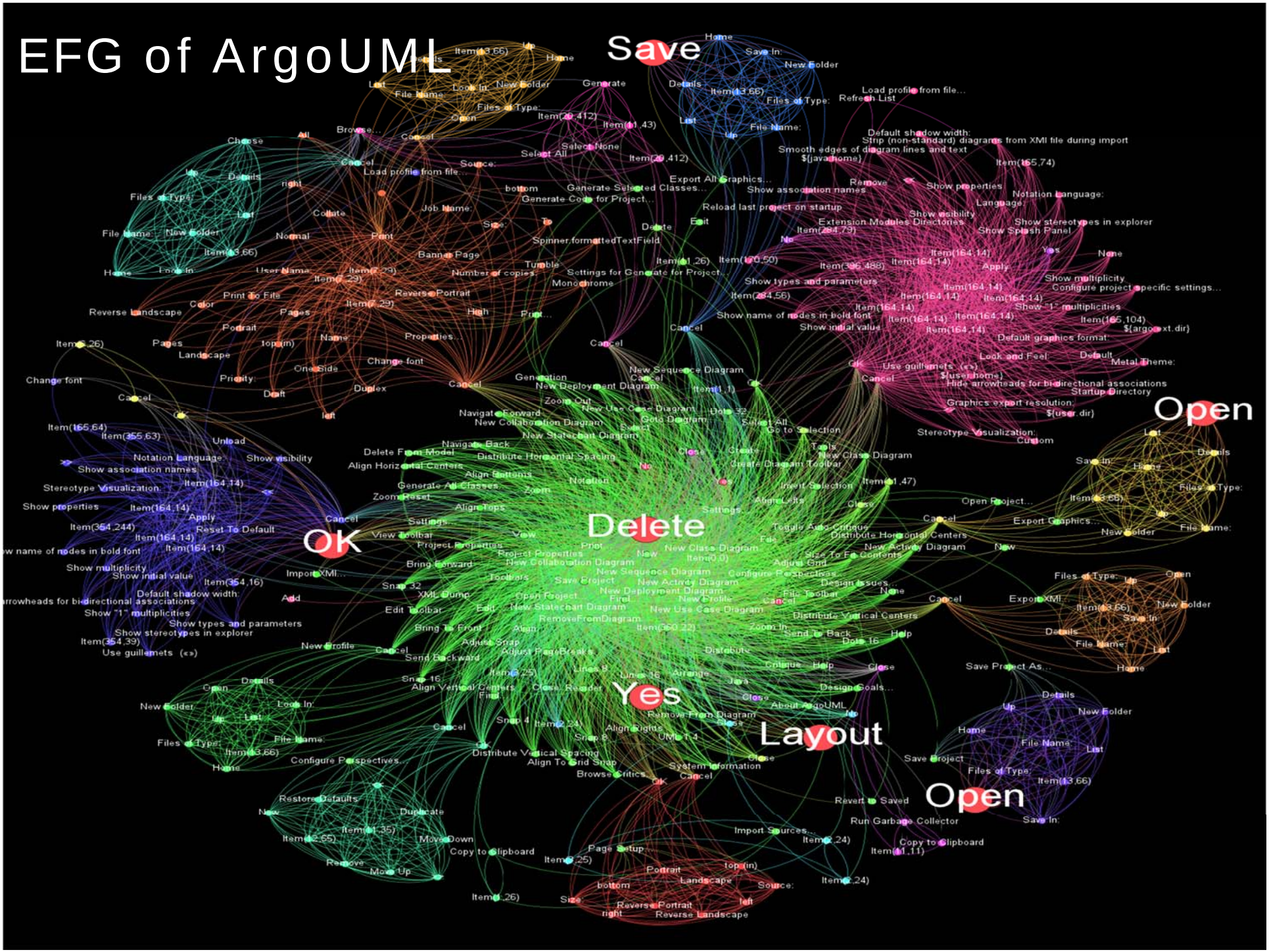
Definition: Event e_x follows e_y iff e_x can be performed immediately after e_y.

Event-flow Graph



Definition: Event-flow graph is a 4-tuple <*V, E, B, I*>

- *V* is the set of vertices, representing events,
- *E* is the set of directed edges, showing the follows relationship,
- *B* is the subset of events first available (shown in red),
- *I* is the subset of events that invoke other windows (dotted lines).

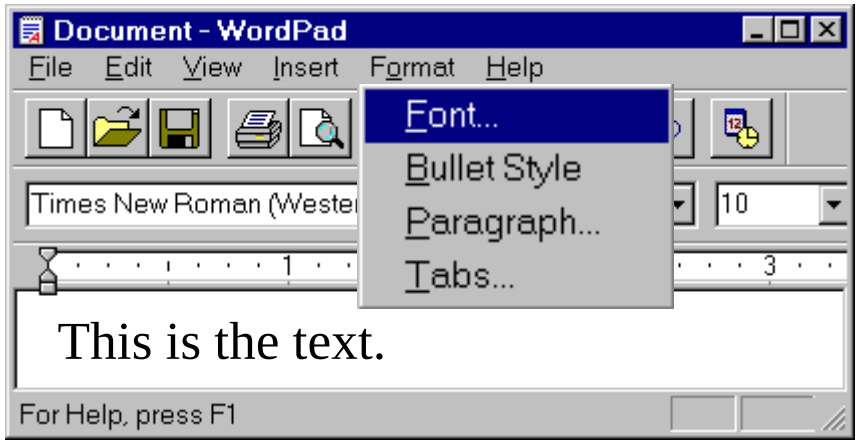

EFG of ArgoUML
Save
Open
Delete
OK
Yes
Layout
Open

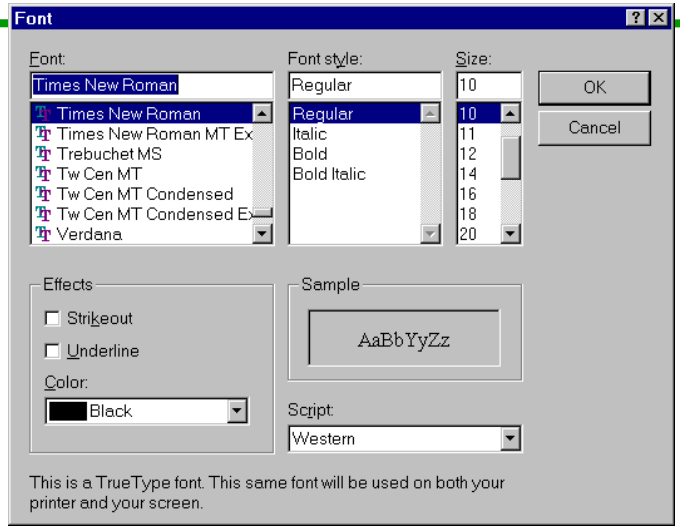
GUI Test Case

- *legal event sequence*
  - $e_1;e_2;e_3;\ldots;e_n$ is a legal event sequence
    - if (e_i, e_{i+1}) is an edge in an event-flow graph
    - or e_i invokes component C_x and e_{i+1} is an event in C_x
- A GUI test case is a triple
  - $(S_0, e_1;e_2;e_3;\ldots;e_n, S_1; S_2;\ldots;S_n)$
    - S_0 is a GUI state, and
    - $e_1;e_2;e_3;\ldots;e_n$ is a legal event sequence
    - $S_i = e_i(S_{i-1})$, $1 <= i <= n$

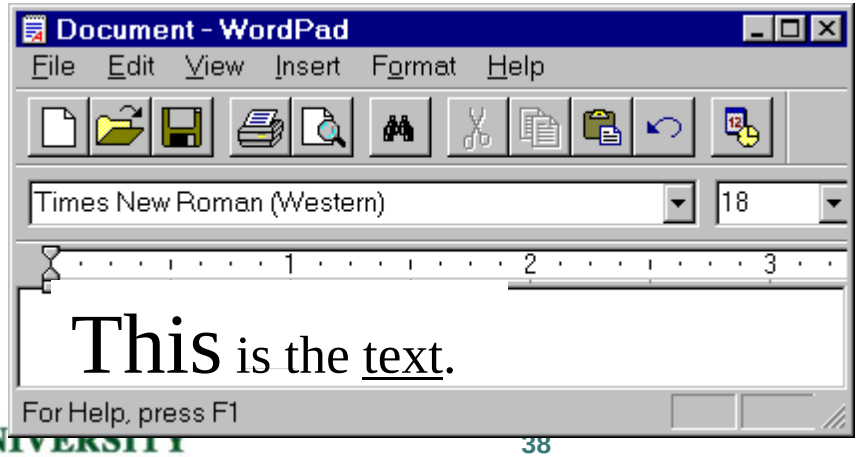
A Test Case for WordPad

s_0



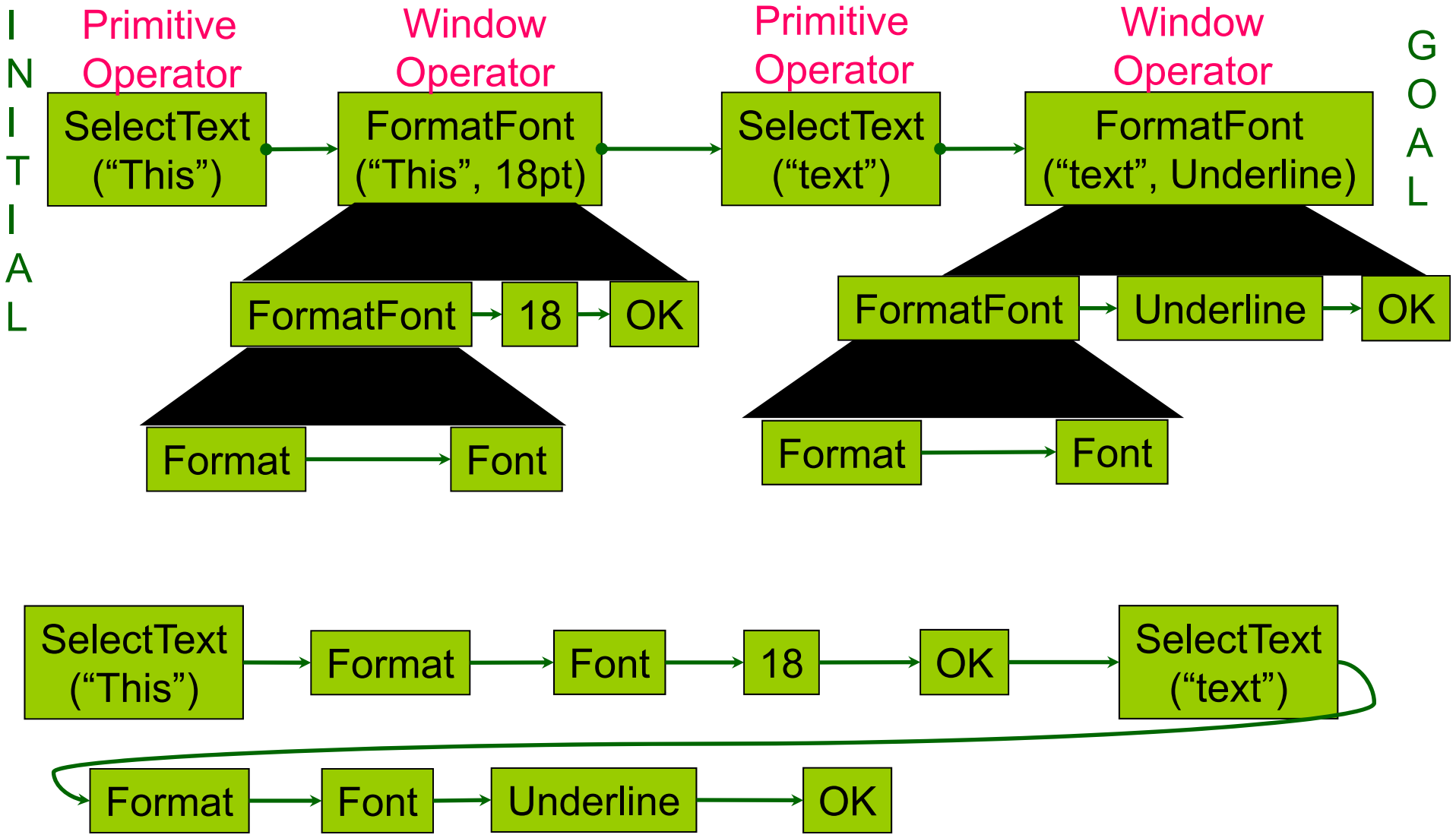


SelectText ("This") → Format → Font → 18 → OK → SelectText ("text") → Format → Font → Underline → OK



Expected State Sequence in Test Oracles

Test Case with Abstract Operators

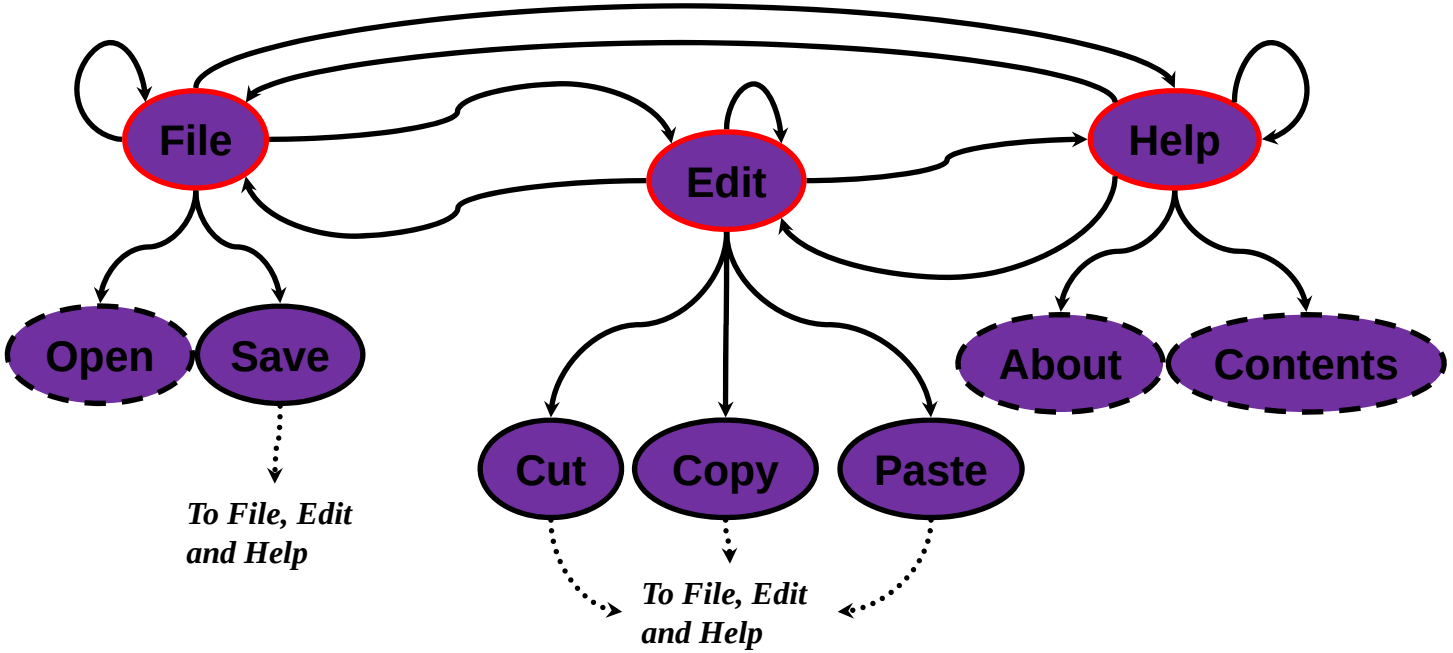


Coverage Criteria

- Intuitively
  - Each windows is a unit of testing
  - Test events within each window
    - Intra-window coverage criteria
  - Test events across components
    - Inter-window coverage criteria

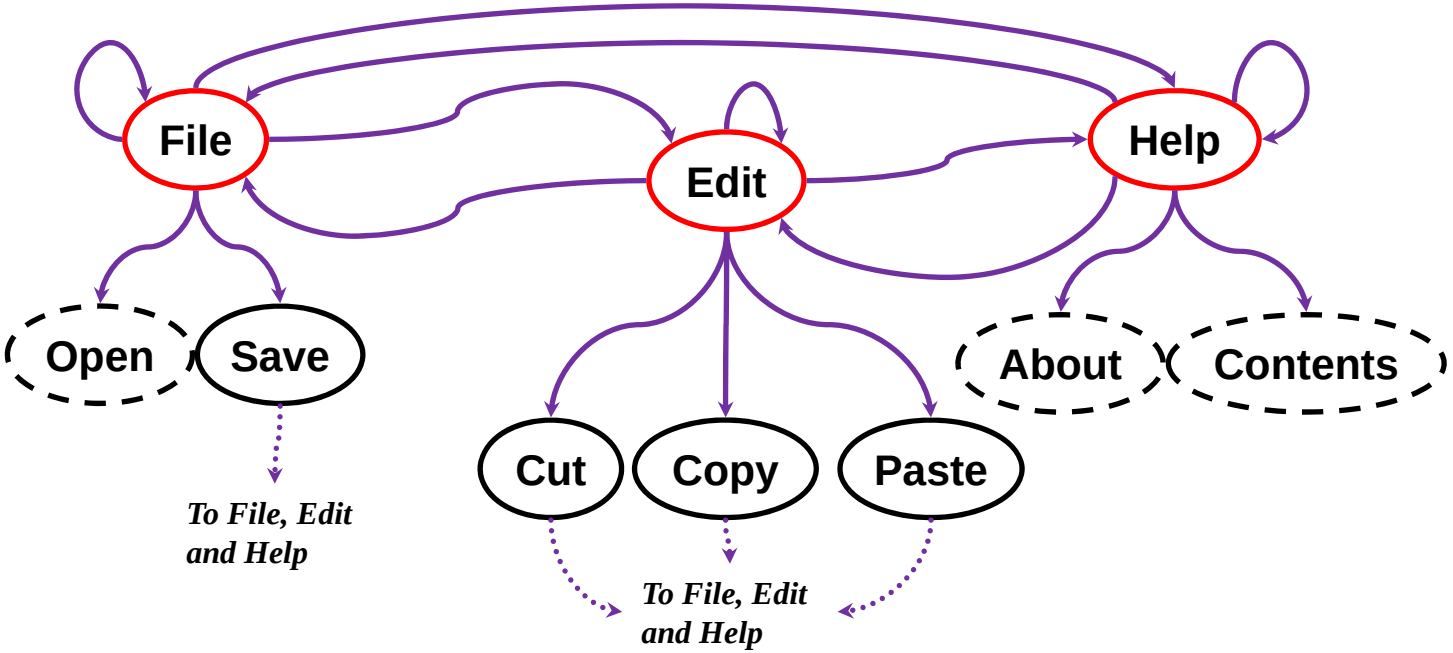
Intra-Window Coverage

- Event coverage
  - Cover every node in the event-flow graph
  - Each event in the window should be performed at least once.



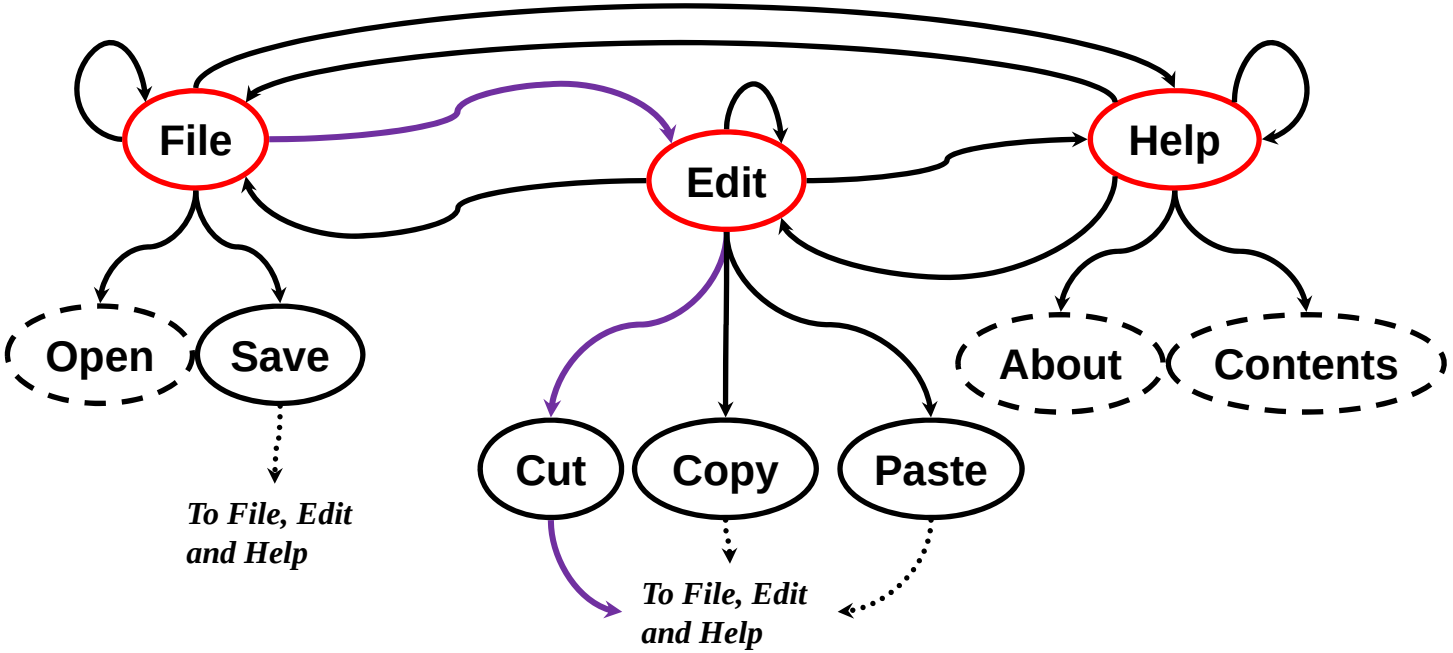
Intra-Window Coverage

- Event-interaction coverage
  - Cover every edge in the event-flow graph
  - After an event v has been performed, all events that can follow v should be executed at least once.



Intra-Window Coverage

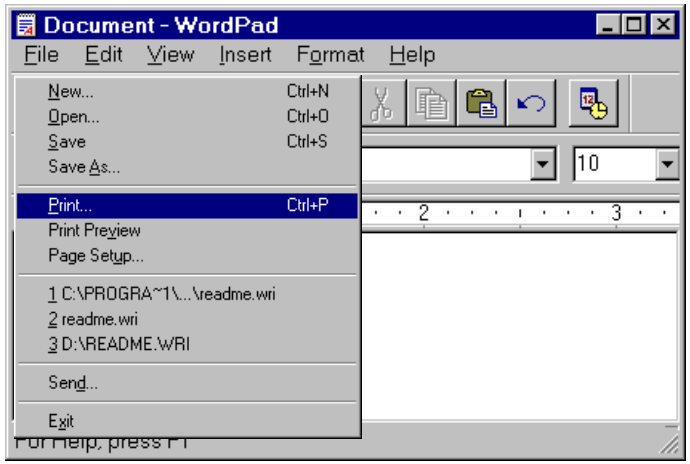
- Length-n event sequence coverage
  - Cover sub-paths of length n in the event-flow graph



Inter-Window Coverage

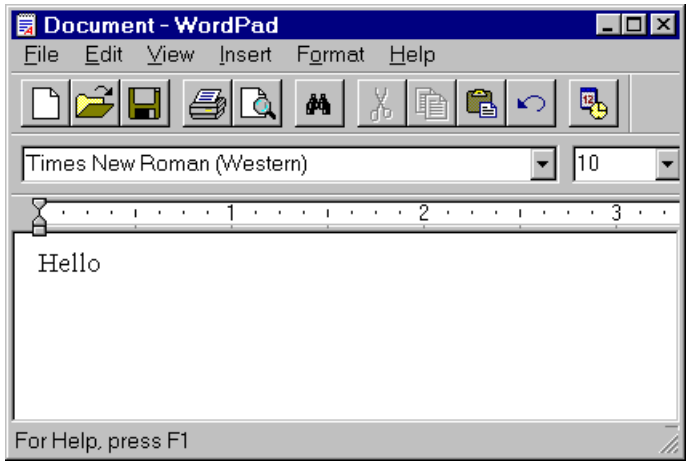
- Concern the event flow between windows

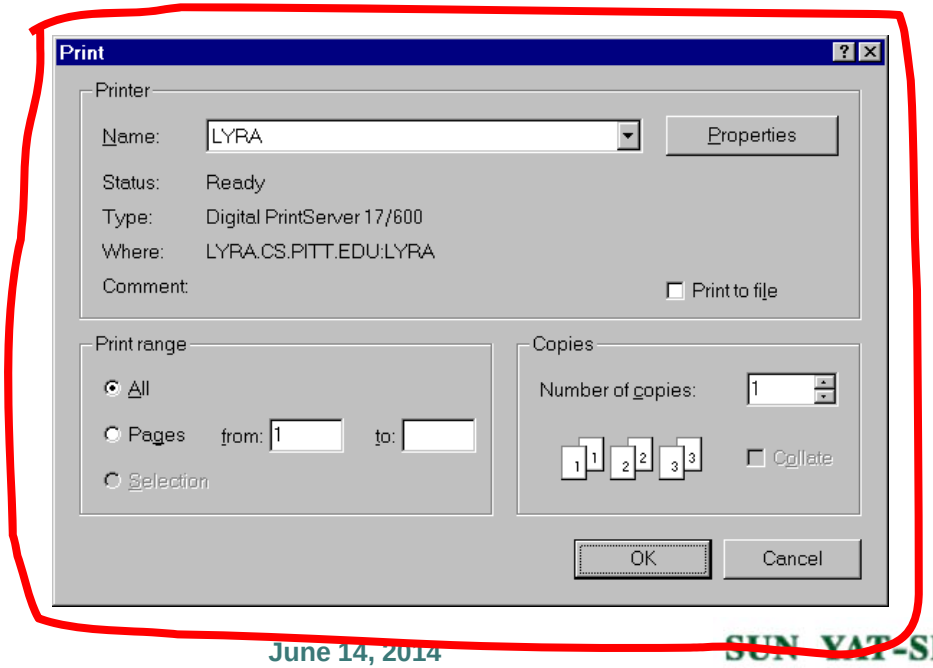
Modal Windows in GUIs

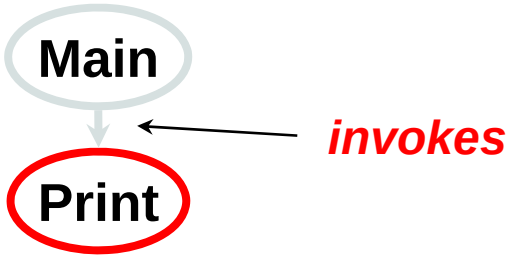


Main

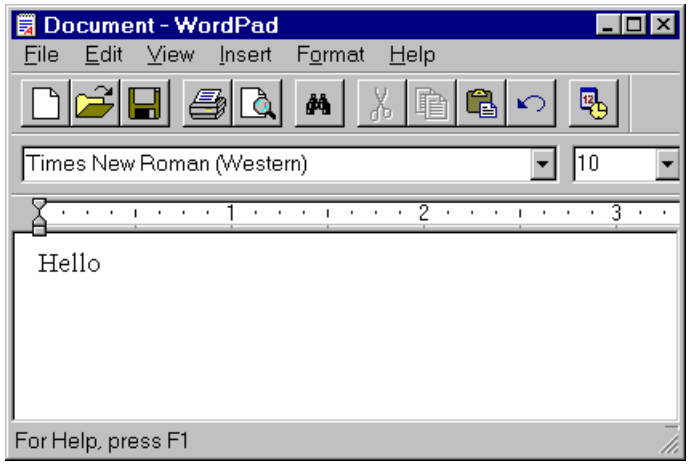
Modal Windows in GUIs

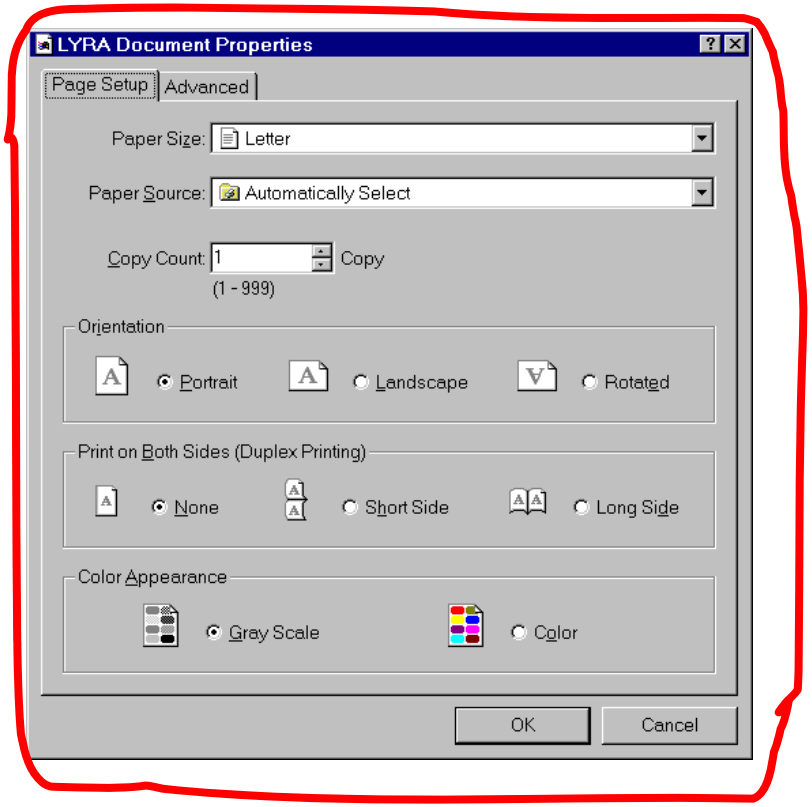


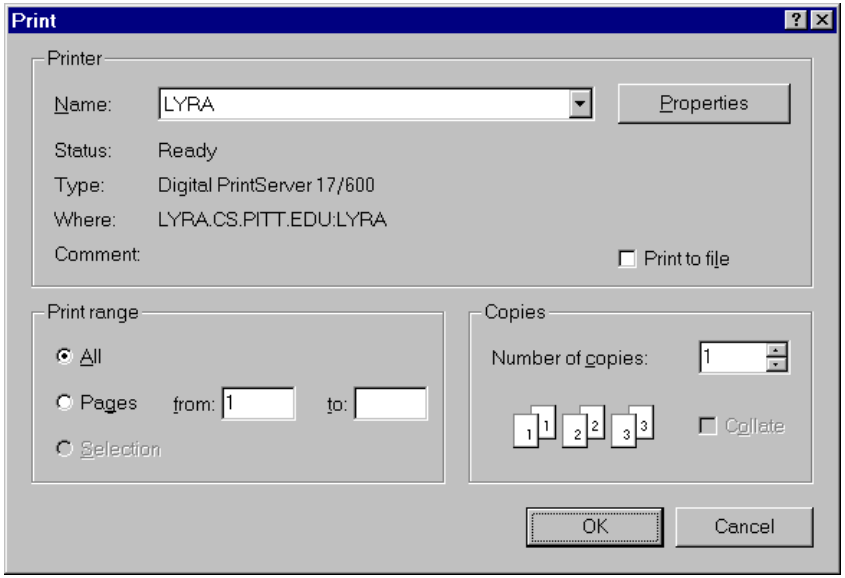


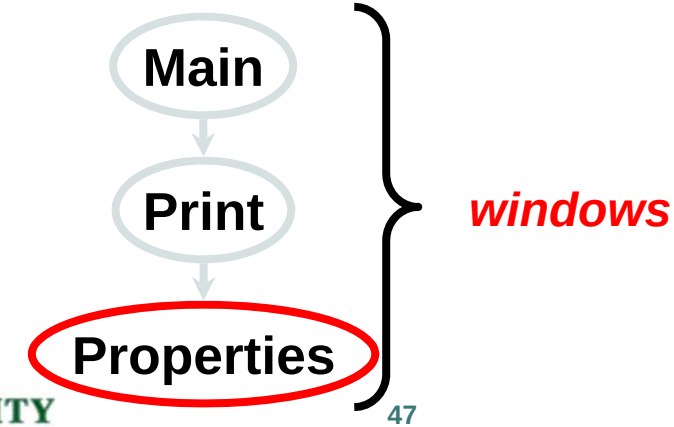


Modal Windows in GUIs

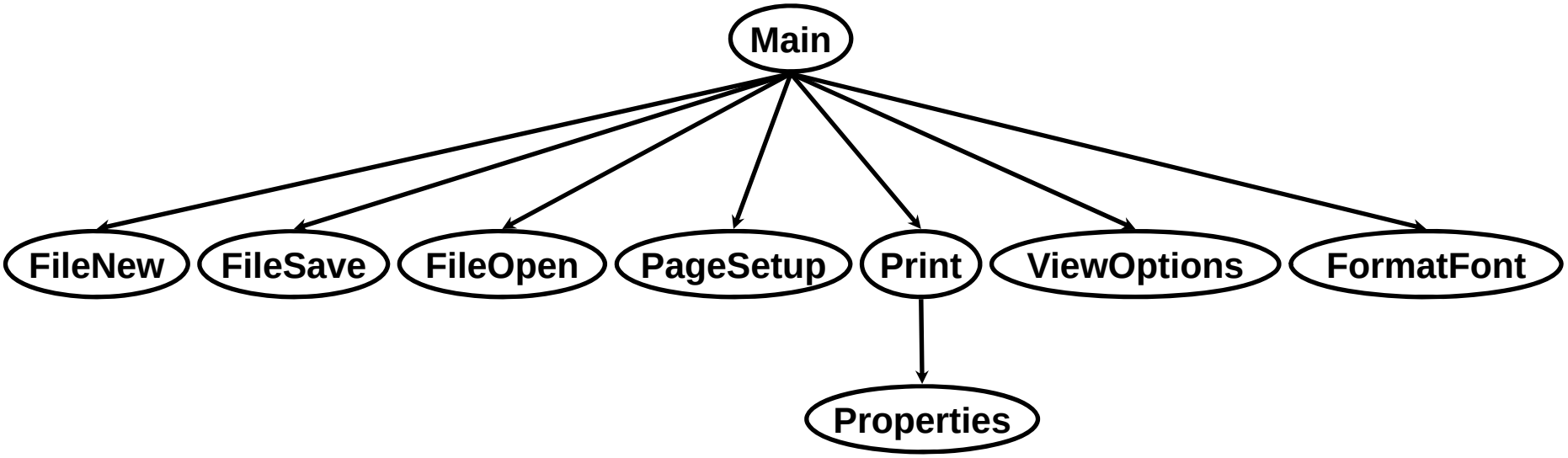








Integration Tree



Definition: Integration tree is a triple <*N*, *R*, *B*>

- *N* is the set of windows in the GUI
- *R* ϵ *N* is a designated window called the ***Main*** window
- *B* is the set of directed edges showing the invokes relation between windows, i.e., (C_x, C_y) ϵ *B* iff C_x invokes C_y.

Inter-Window Coverage

- Invocation coverage
  - Invoke each window
  - Cover each restricted-focus event
- Invocation-termination coverage
  - Invoke each window and terminate it
  - Cover restricted-focus event followed by a termination event
- Inter-window length-n coverage
  - Cover sequences that start with an event in one window and end with an event in another window.

Coverage Hierarchy

Length-*n* event-sequence *n* > 2

⇓

Event-interaction

⇓

Event

⇓

Invocation

Inter-window
Length-n Event-sequence n > 2

⇓

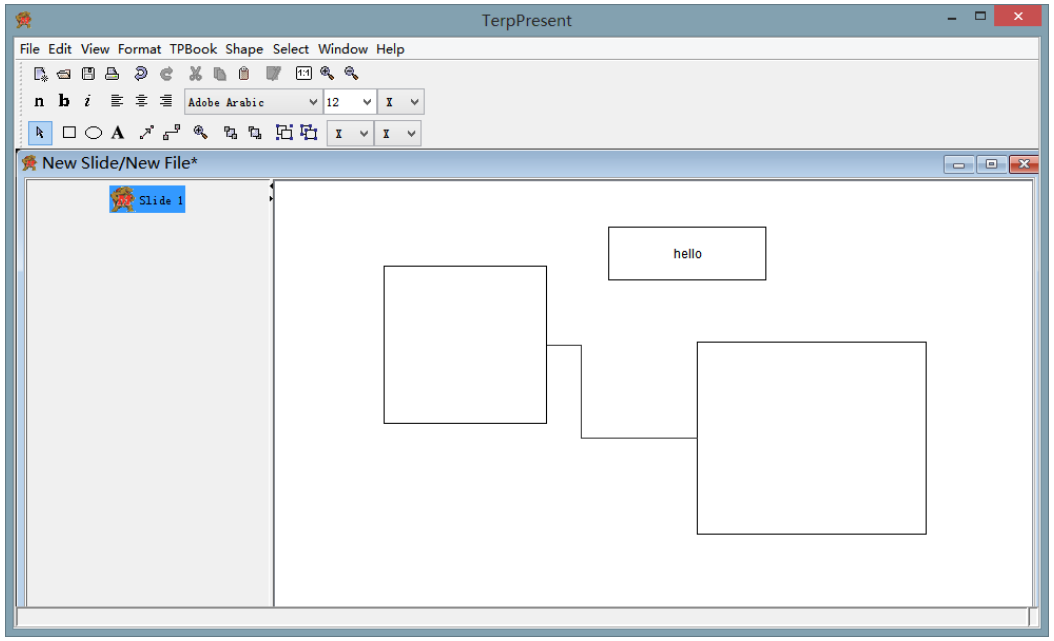
Invocation-termination

Design adequate GUI test cases

- How to design test suite that achieve event-based test adequacy for a GUI program?
  - **Step 1**: enumerate all windows of the program and build the window hierarchy.
  - **Step 2**: for each window, identify the events that can occur and describe them with operators.
  - **Step 3**: design test cases to cover every representative events.
  - **Step 4:** identify the “follow” relation between events and build the event flow graph.
  - **Step 5**: design test cases to cover intra-window event interactions.
  - **Step 6**: design test cases to cover inter-window event interactions.

Case Study: TerpPresent

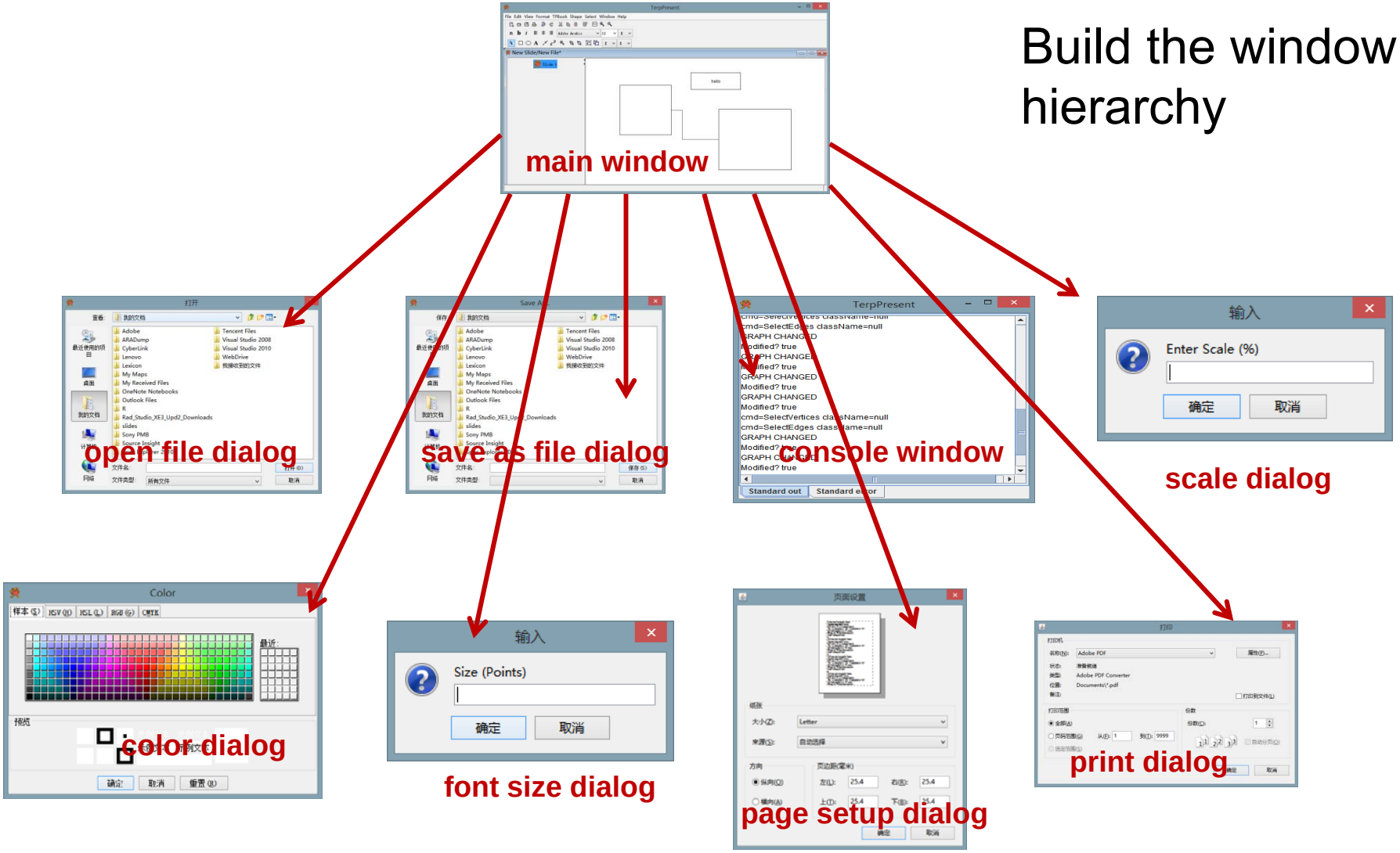
- Similar to Microsoft PowerPoint
  - Drawing figures to create slides



What are the adequate test cases for this GUI program?

Step 1: Enumerating Windows

Build the window hierarchy

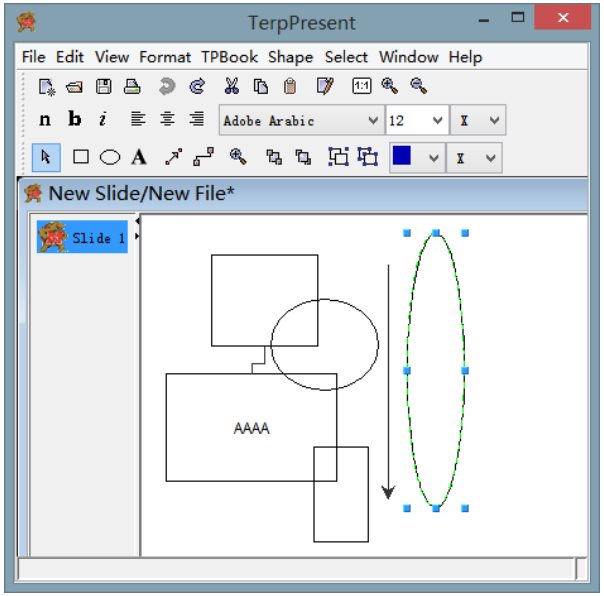


Step 2: Identify Operators

- **Event**: interaction operation that can occur to a GUI window.
  - Example: click the OK button, select a rectangle shape, copy a shape, open file, …
- An **operator** describe a set of events. It consists of:
  - **Parameters**: describe variants of the operator:
    - e.g. “select-shape” operator has a parameter “selected-shape-type”.
  - **Choices of parameters**: describe the equivalence classes of the parameter
    - e.g. “selected-shape-type” can have choices “rectangle”, “line” …
  - **Precondition**: describe under what conditions the operator can be triggered.
  - **Effects**: what happens after this operator occurs.
- An operator describe a set of representative events, each of which is corresponding to one combination of choices.
  - This is exactly the idea of category-partition testing.

Example 1: *select-multiple-shapes*

- Example: the *left-click-select-single-shape* operator for the main window in TerpPresent.



Operator: click-select-single-shape
Parameters:
selected shape type:
rectangle.
cycle.
line connector.
text box.
group.
Precondition: the active window contains at least one shape.
Effects: the shape is selected.

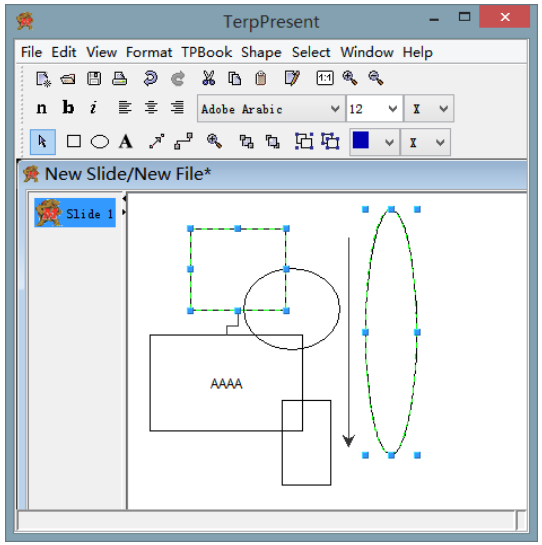
This operator generates 5 representative events.

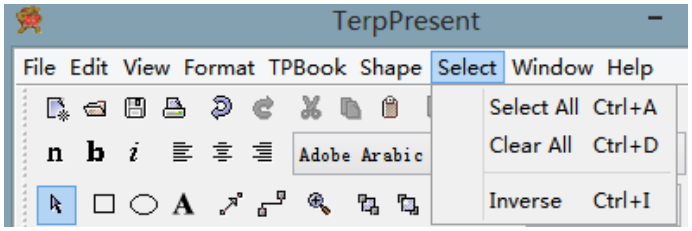
All the six events can only occur when the active window contains at least one shape to select

The occurrence of these events establish this effects, which can be required by other events as precondition.

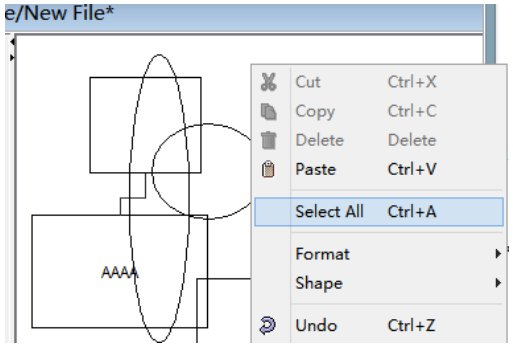
Example 2: *select-multiple-shapes*

- Example: the *select-multiple-shapes* operator for the main window in TerpPresent.





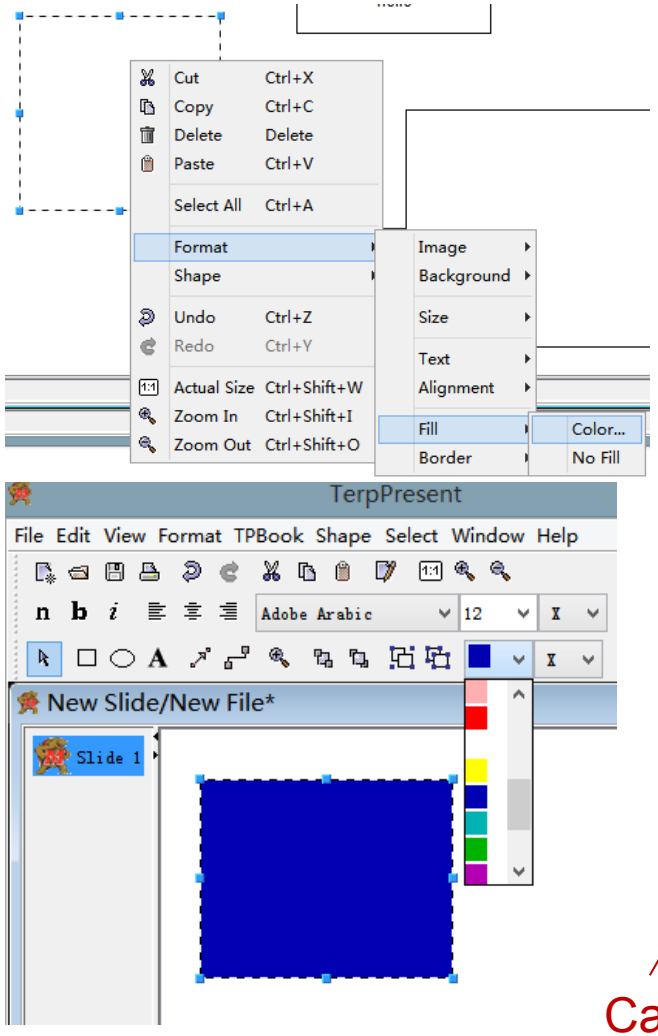
Totally 12 events for this operator, such as selected shape count: 999, selection method: shift + left-click



Operator: select-multiple-shapes
Parameters:
selected shape count:
two.
more than two.
999.
selection method:
selection box
shift + left-click
Menu item “Select all”
Menu item “Inverse”
Precondition: the active window contains sufficient shapes.
Effects: the shapes are selected.

Example 3: *set-fill-color*

- Example: the *set-fill-color* operator for the main window in TerpPresent



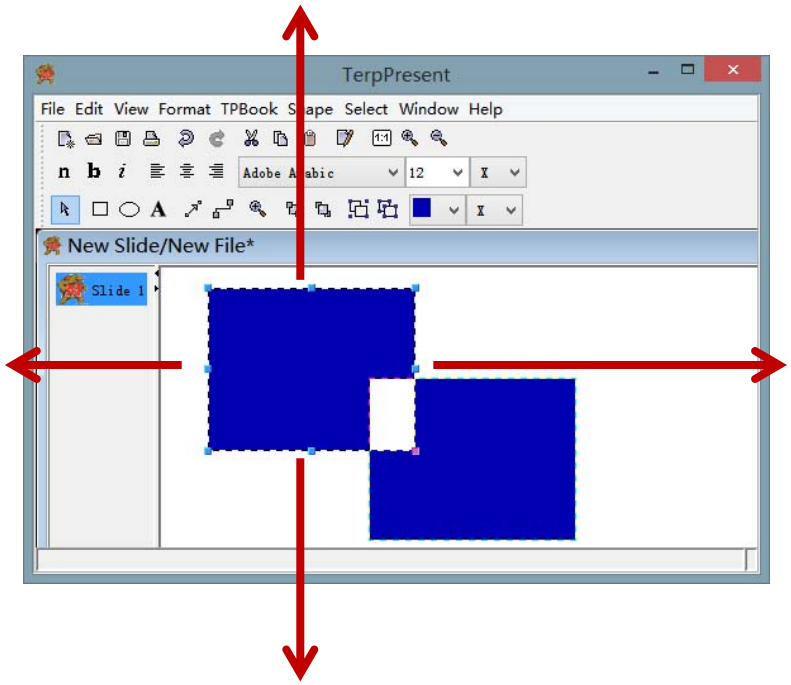
Operator: set-fill-color
Parameters:
selected shape count:
more than one. [NES]
one. [NES] [ONE]
empty.
the sole selected shape type:
rectangle. [if ONE]
cycle. [if ONE]
color selection method:
dialog.
toolbar preset. [PRESET]
color:
preset. [if PRESET]
in RGB format. [if not PRESET]
Precondition: none
Effects:
[if NES] change the color of the selected shape [else] do nothing

Can use constraints in a way similar to TSL.
Might use the TSL tool to help generating all events too.

Example 4: *move-shape*

- Example: the *move-shape* operator for the main window in TerpPresent

Move out of the upper window border will result in the shape missing from the slide (a bug?)

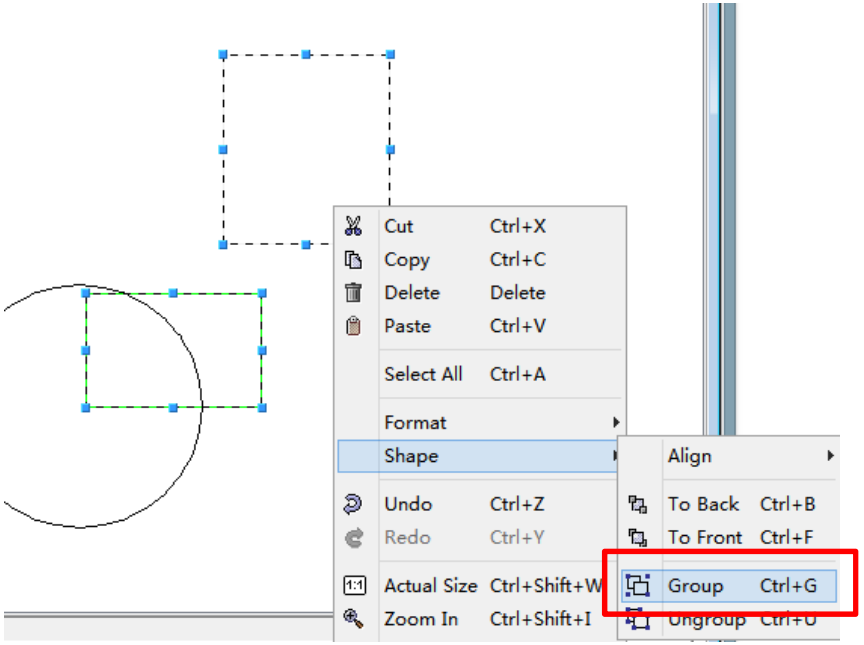


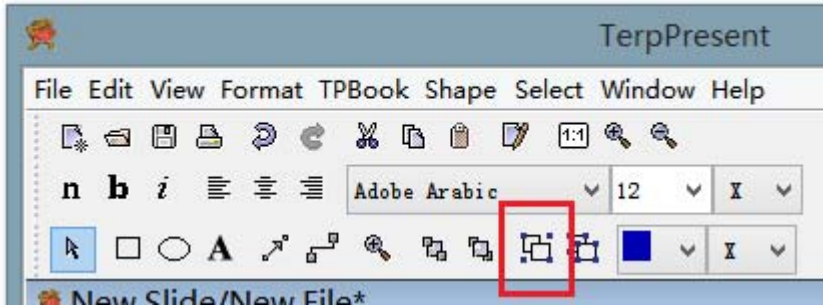
Move out of the bottom window border will automatically extend the slide height.

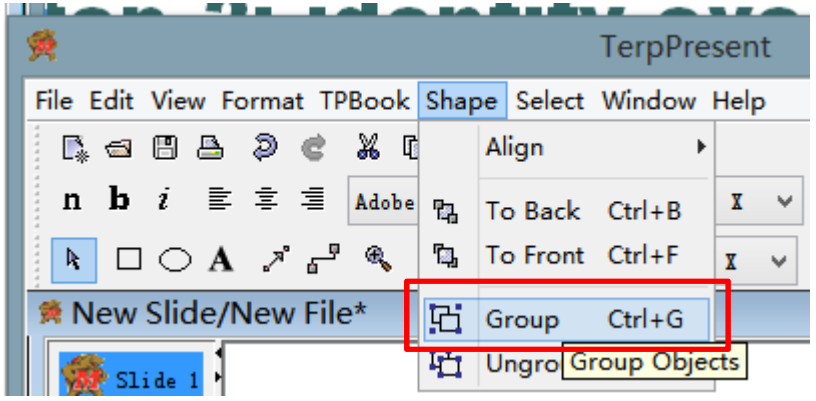
Operator: move-shape
Parameters:
selected shapes:
more than one.
one.
move method:
drag-and-drop.
arrow keys.
move-to position:
all selected shapes inside window border.
some out of the upper window border.
some out of the left window border.
some out of the right window border.
some out of the bottom window border.
Precondition: the selected shapes are non-empty
Effects:
The selected shapes are moved to the new position.

Quiz 1: Identifying Operators

- Write the description for the *group-shapes* operator
  - This operator groups multiple shapes together into a single shape.
  - Can be triggered by right-click menu, main menu, toolbar button, or ctrl+g as follows:







Quiz 1: Identifying Operators

- Write the description for the *group-shapes* operator

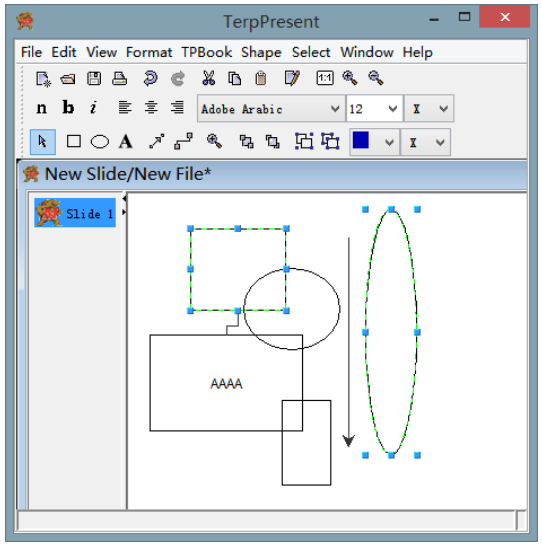
Operator: group-shapes
Parameters:
    selected shape count:
        one.
        more than one.
        999.
    whether a grouped shape is selected:
        yes.   # What is the expected behavior?
        no.
    trigger method:
        toolbar button.
        right-click menu item.
        main-menu item.
        ctrl-g.
Precondition: the selected shapes are non-empty
Effects: The selected shapes are grouped into one single shape.

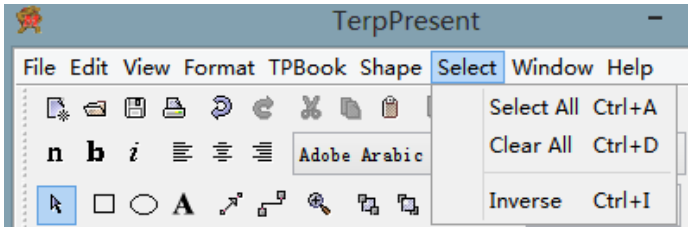
Decisions to Make

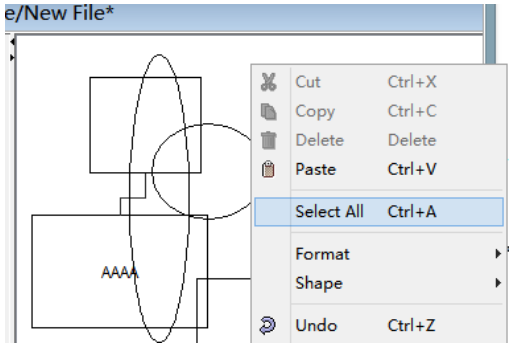
- When identifying operators, we need to make several decisions:
  - How to partition each parameter?
  - Parameterization or not?
  - Abstraction or not?
  - Aggregation or not?

Decisions to Make (1)

- How to partition parameters? What equivalence classes shall be added?
- Example: the *select-multiple-shapes* operator we define previously.







Operator: select-multiple-shapes
Parameters:
selected shapes:
two.
More than two.
999.
selection method:
selection box
shift + left-click
Menu item “Select all”
Menu item “Inverse”
Precondition: the active window contains sufficient shapes.
Effects: the shapes are selected.

Is “more than two” really necessary? Shall we add more choices?

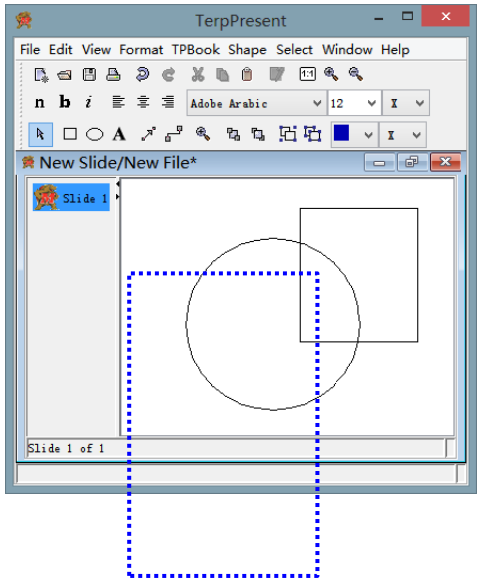
Shall we further partition these two choices?

Further Partitioning

- The choice *selection box* can be further partitioned:

Operator: select-multiple-shapes
Parameters:

……
selection method:
selection box. [MSE]
……
whether part of the selection box is outside the window:
yes. [if MSE]
no. [if MSE]
how the selection box overlaps with shapes:
overlaps with at least one shape. [if MSE]
does not overlap with any shape. [if MSE]



part of the selection box is outside the window, and the selection box overlaps with both two shapes

Is this necessary? It depends on your experience on programs of this kind: whether the additional choices require special treatment in the implementation.

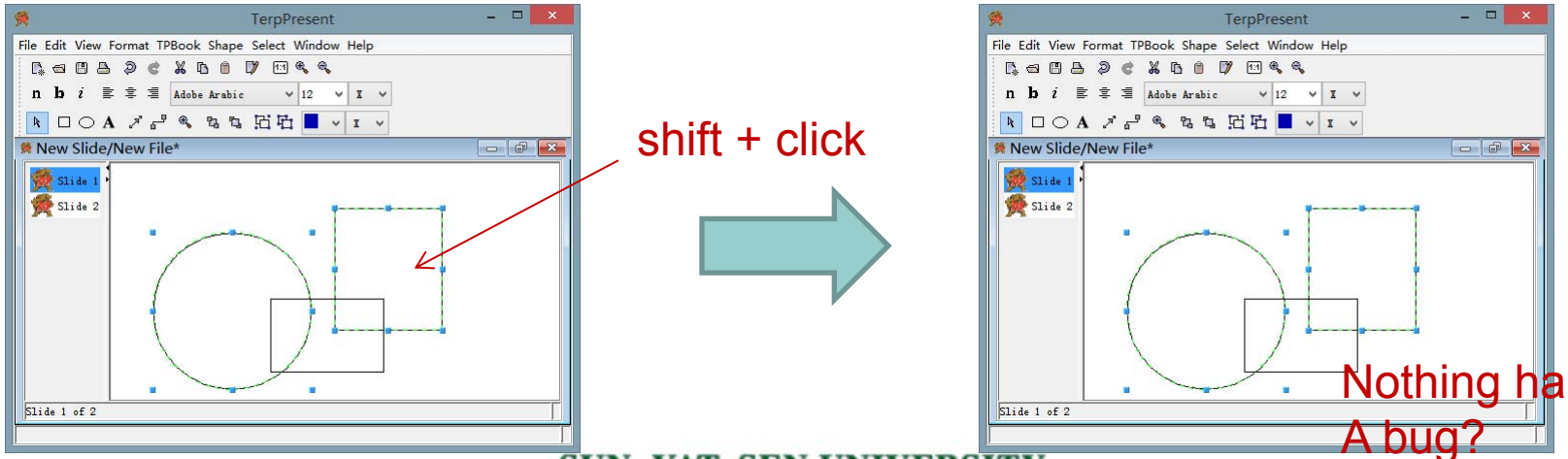
In this case we prefer further partitioning

Further Partitioning

- Similarly, the choice "*shift* + *left-click*" can be further partitioned:

Operator: select-multiple-shapes
Parameters:

……
selection method:
shift + left-click. [SHIFT_CLICK]
……
whether a selected shape has been shift + clicked again:
yes. [if SHIFT_CLICK]
no. [if SHIFT_CLICK] #expect unselect this shape

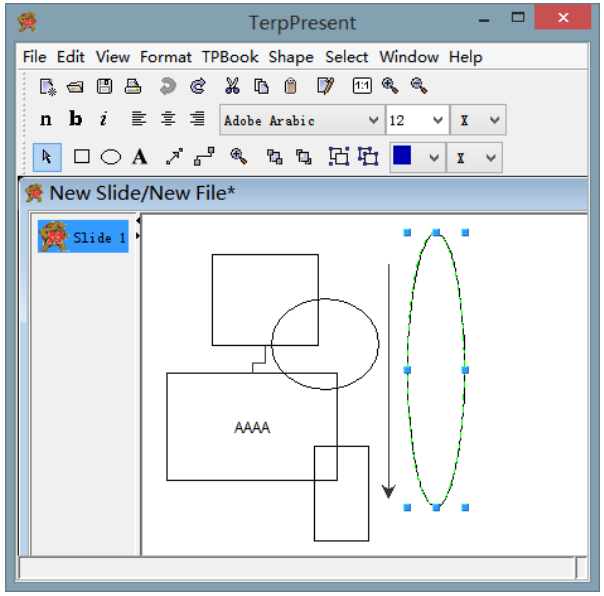


Decisions to Make (1)

- General rules for parition:
  - Adding additional choices if we know from experience they require special treatment in the implementation.
  - Also, choices for boundary cases are usually necessary.

Decisions to Make (2)

- Parameterization or not
  - Example: the *left-click-select-single-shape* operator we define previously.



Operator: left-click-select-single-shape
Parameters:
selected shape type:
rectangle.
cycle.
line connector.
text box.
group.
Precondition: the active window contains at least one shape.
Effects: the shape is selected.

Another two ways to select one single shape: right click or selection box

Shall we create additional operators, or model the different selection ways as a parameter?

Decisions to Make (2)

- Parameterization or not?

Operator: right-click-select-single-shape
Parameters:
    selected shape type:
        rectangle.
        cycle.
        line connector.
        text box.
        group.
Precondition: the active window contains at least one shape.
Effects: the shape is selected.

Operator: selection-box-select-single-shape
Parameters:
    selected shape type:
        rectangle.
        cycle.
        line connector.
        text box.
        group.
Precondition: the active window contains at least one shape.
Effects: the shape is selected.

vs.

Operator: select-single-shape
Parameters:
    selection method:
        left-click.
        right-click.
        selection-box
    selected shape type:
        rectangle.
        cycle.
        line connector.
        text box.
        group.
Precondition: the active window contains at least one shape.
Effects: the shape is selected.

Either way, the event set is the same.

The difference is in the ease of understanding and maintenance.

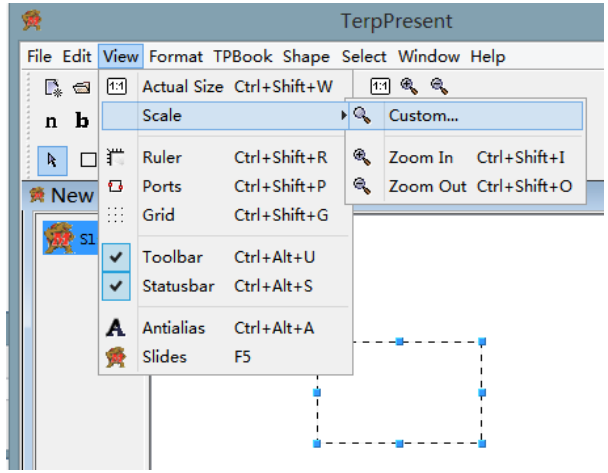
In this case, we prefer parameterization.

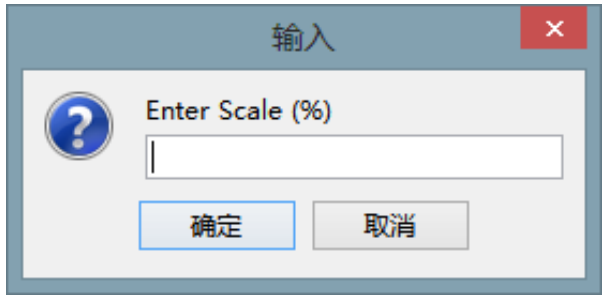
Decisions to Make (2)

- Parameterization or not? General rules:
  - If two operators have similar parameters, preconditions, and effects, consider joining them into one operator, with one or more additional parameters separating them. However, …
  - If such a parameterization results in complex constraints, don't do it.

Decisions to Make (3)

- Abstraction or not?
  - Example: the *show-custom-scale* operator
    - This operation open a dialog to set the scale.





Operator: show-custom-scale
Parameters:
Precondition: None.
Effects: Open the scale input dialog.

Not abstracted
Pros: simpler operator
Cons: require inter-window event interaction coverage to cover the operation.

vs.

Operator: show-custom-scale
Parameters:
scale value:
0%
1%
1%~99%
100%
MAX
Precondition: None.
Effects: the display scale is changed to the desired value .

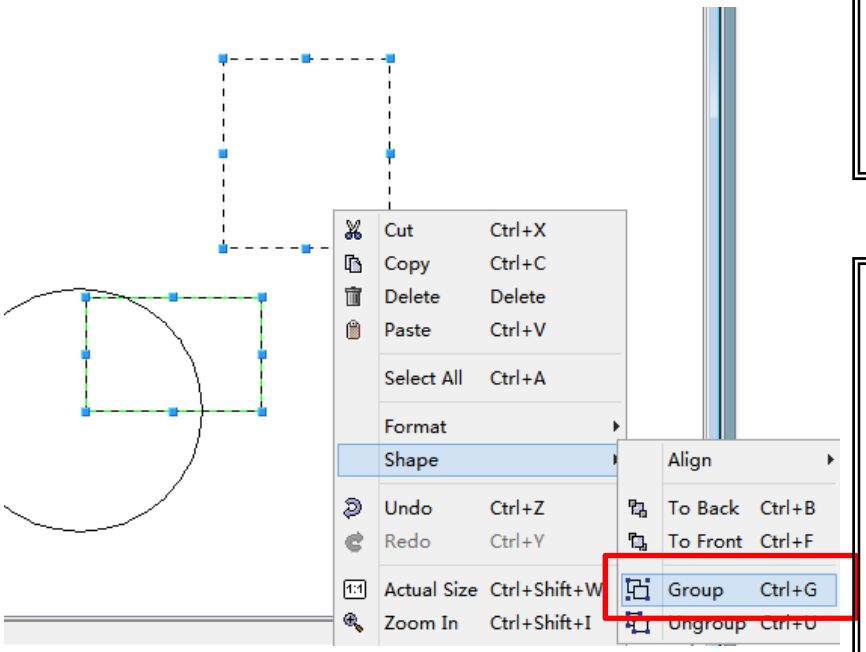
Abstracted
Pros: event-coverage can already cover this operation.
Cons: redundant with the operators for the scale input dialog. result in complex operators.

Decisions to Make (3)

- Abstraction or not? General rules:
  - If you do not plan to achieve inter-window event interaction coverage (which can be costly), always abstract the dialog into an operator if it is possible.
  - If the dialog is simple, prefer abstraction.

Decisions to Make (4)

- Aggregation or not?
  - Example: the quiz we just did.
  - Another way to model operator



In this case, we prefer aggregation.

Operator: open-right-click-menu
Parameters:
Precondition: None.
Effects: open the right-click menu.

Operator: open-sub-menu-shape
Parameters:
Precondition: the right-click menu is open
Effects: open the “Shape” sub-menu.

Operator: select-menu-item-group
Parameters:
selected shape count:
one.
more than one.
999.
whether a grouped shape is selected:
yes.
no.
Precondition: the “Shape” sub-menu is open, and the selected shapes are not empty
Effects: The selected shapes are grouped into one single shape

Decisions to Make (4)

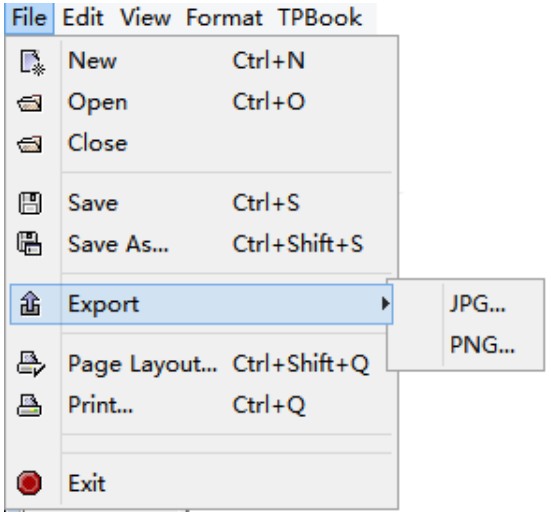
- Aggregation or not? General rules:
  - If some events are unlikely to trigger errors (such as open menu/sub-menu), or the program does not handle them at all (such as pressing keys in the edit control), then we can aggregate them into the events that follow them.

Identifying Full Operator Set

- Not a trivial task even for a small GUI program
  - Not surprising, because GUI programs are indeed complex.
  - TerpPresent contains 1861 methods with totally 22100 lines of code.
  - GUI games can be much more complex:

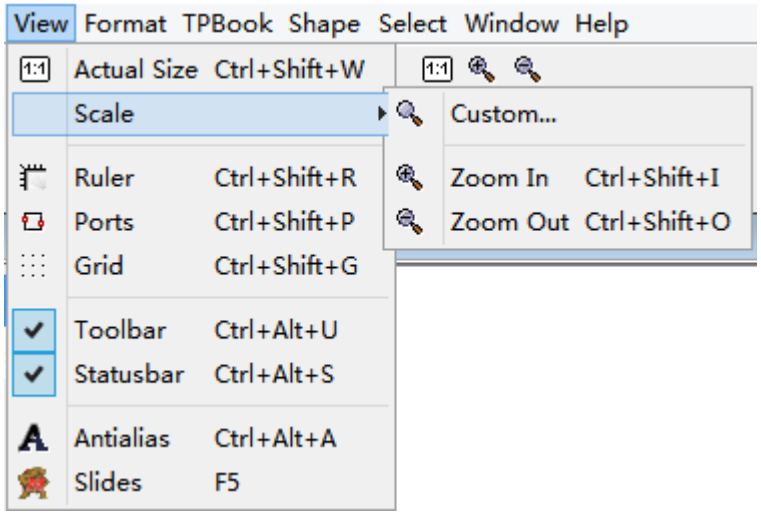
- A systematic way to discover all operators:
  - Explore all menu items (main or right-click menu).
  - Explore all buttons.
  - Explore the manual and requirement documents.
  - Use a GUIRipper (automatic, but not reliable).

Explore All Menu Items



- create-new-file
- open-file
- save-file
- save-as
- export-as-jpg
- export-as-png
- page-layout
- print

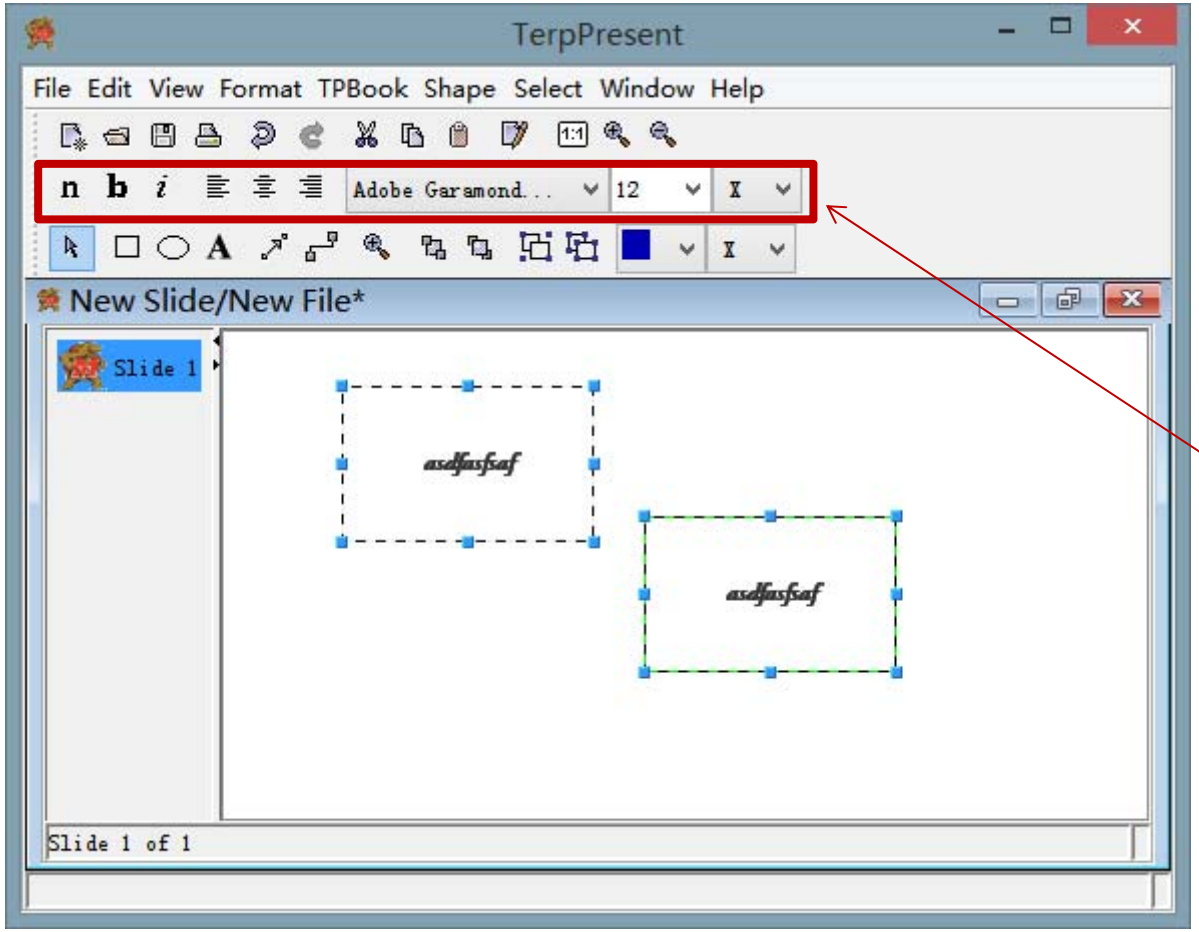
- edit-shape-caption
- undo
- redo
- cut
- copy
- paste
- delete



- show-actual-size
- show-custom-scale
- zoom-in
- zoom-out
- show-ruler
- show-ports
- show-grid
- show-toolbar
- show-status-bar
- enable-antialias
- show-slides

.....

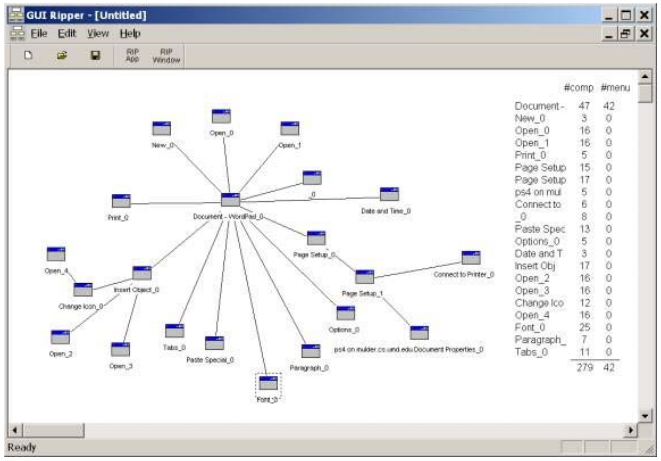
Explore All Buttons



- set-font-style
- set-font-justified
- set-font-type
- set-font-size
- set-font-color

GUIRipper

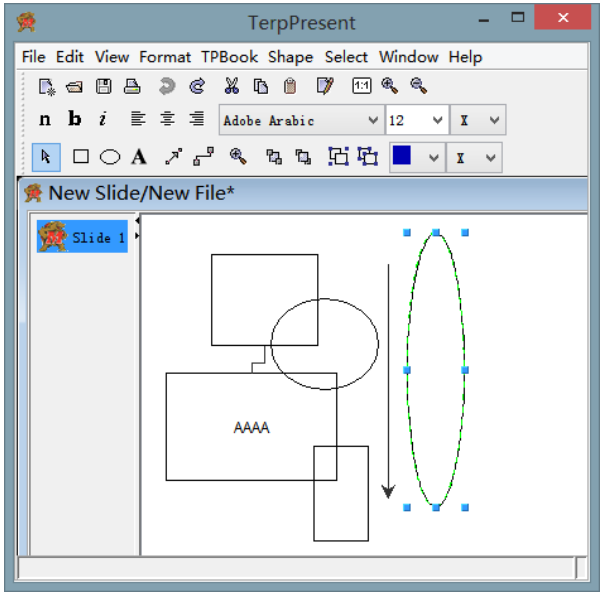
- A GUIRipper automatically discovers the window hierarchy and events of a GUI program by interacting with the GUI objects in the program.
- Example: GUITAR (http://guitar.sourceforge.net/)



- Personal experience: they are not very useful.
  - Cannot discover preconditions and effects.
  - Not smart enough to do abstraction, aggregation, and parameterization.
  - Only recognize limited types of GUI objects. e.g. cannot detect shapes in TerpPresent.

Step 3: Cover Representative Events

- Generate representative events from the set of operators.
  - Design test cases to cover each of them.
- For each event,
  - First satisfies its precondition.
  - Then satisfies its choices
  - Check the expected results.



Five representative events

The inserted shape shall be rectangle

Create a new slide and insert one shape

Check the effect

Operator: click-select-single-shape
Parameters:
selected shape type:
rectangle.
cycle.
line connector.
text box.
group.
Precondition: the active window contains at least one shape.
Effects: the shape is selected.

Step 4: identify the "follow" relation

- Consider each pair of operators (A, B)
  - Whether event of A can occurs after event of B?
    - always, never, conditional
  - How many different ways can A follow B?
    - "representative interactions".
- It is helpful to draw such a table:

Can col follows row?	move-shapes	undo	redo
move-shapes	always	always	never
undo	conditional shape selected after undo	never (only allow one undo)	always
redo	conditional shape selected after redo	always	never (only allow one redo)

Another Set of Operators

Can col follows row?	left-click-select-shape	copy	cut	paste	delete
left-click-select-shape	always • select same shape • different shapes	always	always	always • replacement • paste-as-new	always
copy	always • select copied shape • different shapes	always	always	always	always
cut	always	never	never	always	never
paste	always • select pasted shape • different shapes	always	always	always	always
delete	conditional has >1 shapes	never	never	conditional clipboard no-empty	never

- The final table can be very big.
  - Consider separate it into several parts.

Step 5: Cover Intra-Window Interaction

- Typically, the strength of interaction is limited to two.
- Intra-window two-way event interaction coverage:
  - Cover the three possible “follows” relation: always, never, and conditional.
- **Case 1**: A always follow B:
  - Satisfy the precondition of A.
  - Create one test case for each way of interaction
  - Example: (left-click-select-shape, left-click-select-shape)
    - test case 1: create a new slide -> insert a rectangle shape -> left-click this rectangle -> left-click this rectangle again
    - test case 2: create a new slide -> insert two rectangle shape -> left-click one rectangle -> left-click the other rectangle

Can col entry follows row entry	left-click-select-shape
left-click-select-shape	always • select same shape • different shapes

Step 5: Cover Intra-Window Interaction

- Typically, the strength of interaction is limited to two.
- Intra-window two-way event interaction coverage:
  - Cover the three possible “follows” relation: always, never, and conditional.
- **Case 1**: A always follow B:
  - Satisfy the precondition of A.
  - Create one test case for each way of interaction
  - Example: (left-click-select-shape, left-click-select-shape)
    - test case 1: create a new slide -> insert a rectangle shape -> left-click this rectangle -> left-click this rectangle again
    - test case 2: create a new slide -> insert two rectangle shape -> left-click one rectangle -> left-click the other rectangle
- **Case 2**: under a certain condition A can follow B:
  - Similar to the first case, additionally require satisfying this condition.
- **Case 3**: A can never follow B:
  - After the event of A occurs, check whether event of B is impossible to occur.

Can col entry follows row entry	undo
undo	never

Step 5: Cover Intra-Window Interactions

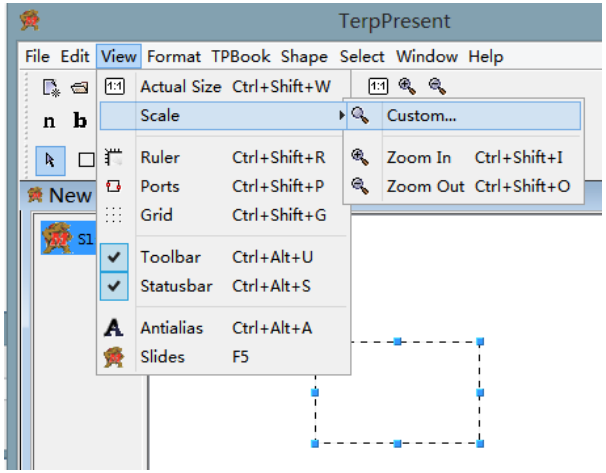
- Covering all *operator interaction* vs. covering all *event interaction*.
- Example: (left-click-select-shape, left-click-select-shape)
  - test case 1: create a new slide -> insert a rectangle shape -> left-click this rectangle -> left-click this rectangle again
  - test case 2: create a new slide -> insert two rectangle shape -> left-click one rectangle -> left-click the other rectangle
- These two test cases only cover two-way operator interaction. In order to cover two-way event interaction, we need 5+5*5 = 30 test cases, as the left-click-select-shape operator can generate 5 events differentiated by the type of selected shape.
  - Even for this small program, covering all event-interaction is too much. Operator-interaction is usually a more practical choice.

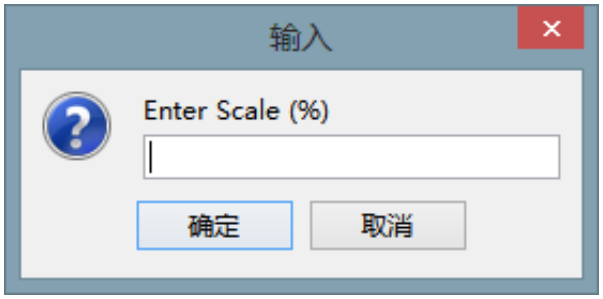
Step 6: Cover Inter-Window Interactions

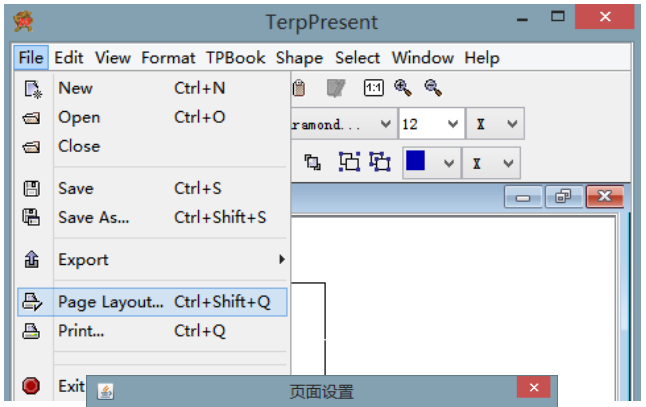
- Inter-window length-n coverage
  - Cover sequences that start with an event in one window and end with an event in another window.
- Typically, n=3
  - One event in the first window, followed by the invocation event, followed by one event in the second window
- Two typical types of invocation:
  - Open a modal dialog
  - Switch to another MDI window

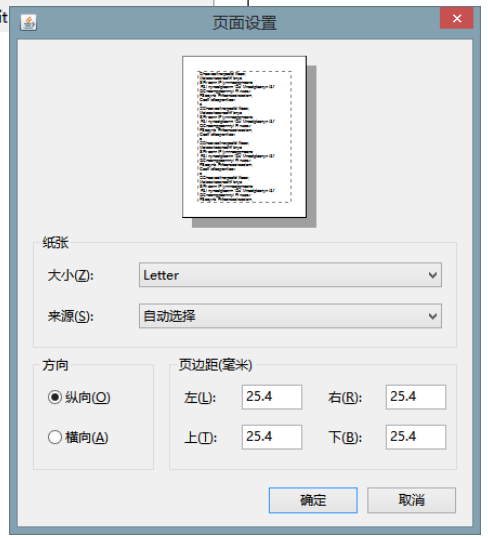
Modal Dialog

- Examples:





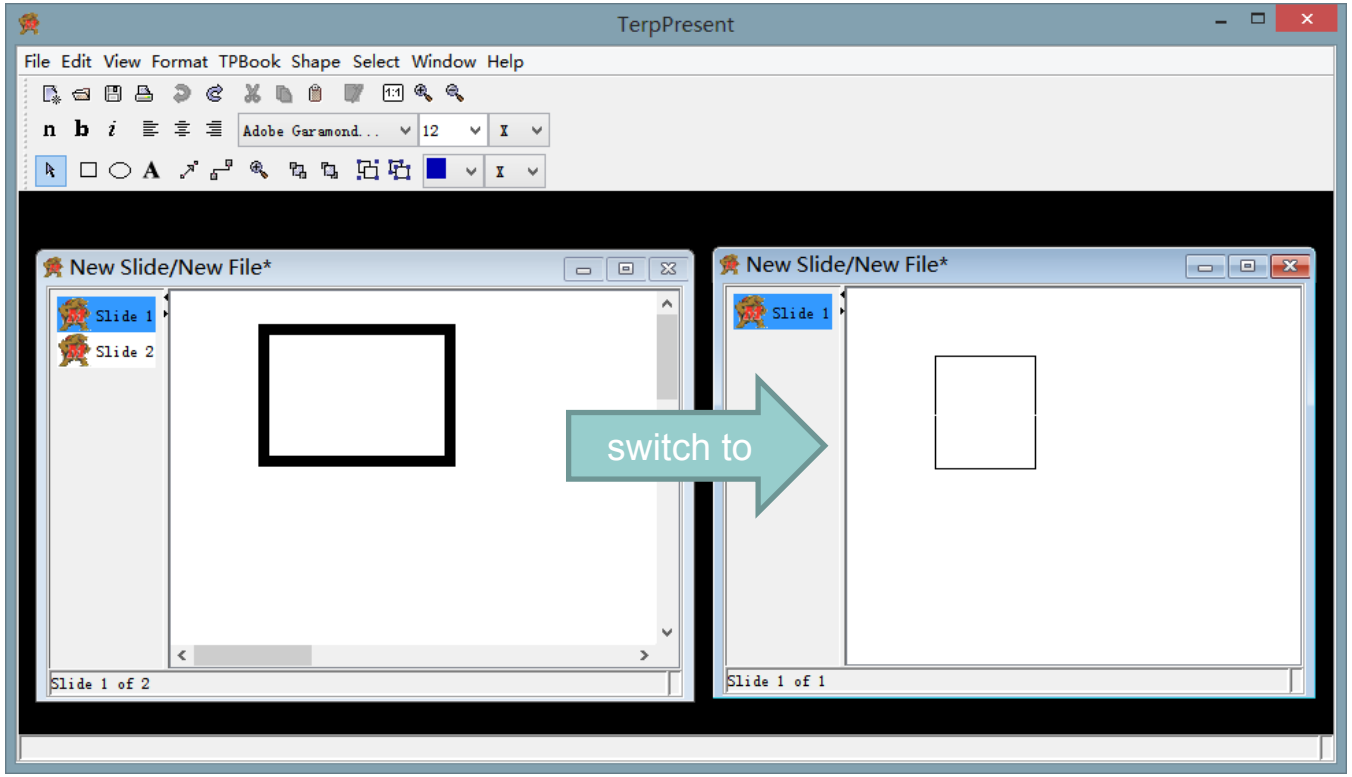




- In this case, we prefer abstraction instead of inter-window event interaction coverage

MDI Window

- Examples of length-3 event interaction:
  - copy -> switch window -> paste
  - delete -> switch window -> undo
  - undo -> switch window -> redo
- Draw a table similar to that in step 4 and design test cases.



Review: Event-based Coverage

- Key concepts:
  - Event, Operator, Parameter, Choice, Interaction
- How to design test suite that achieve event-based test adequacy for a GUI program?
  - **Step 1**: enumerate all windows of the program and build the window hierarchy.
  - **Step 2**: for each window, identify the events that can occur and describe them with operators.
  - **Step 3**: design test cases to cover every representative events.
  - **Step 4:** identify the “follow” relation between events and build the event flow graph.
  - **Step 5**: design test cases to cover intra-window event interactions.
  - **Step 6**: design test cases to cover inter-window event interactions.

Level-3 Coverage Criteria

- **Event–based Coverage （基于事件的覆盖）**
  - **GUI program = a set of events**

- **Model-based Coverage（基于状态的覆盖）**
  - **GUI program = a set of features**

Features of a GUI program

- A **feature** is an end-effect that the user of a GUI wants to achieve.
- **Examples:**
  - File-save
  - Copy-paste
  - Shape-alignment
  - Font-setting
- Features are described in the requirement and manual.
  - “user-stories” in SCUM

Scenario

- A **scenario** is sequence of events carried out by a user to use a desired feature.
  - Example: in order to use a copy-paste feature, the user first select a shape, then copy it, then paste it to another place.
- A feature can have multiple (possibly infinite) scenarios
- A scenario x might overlap with another scenario y, or be fully contained by y, or simply share GUI objects with y.

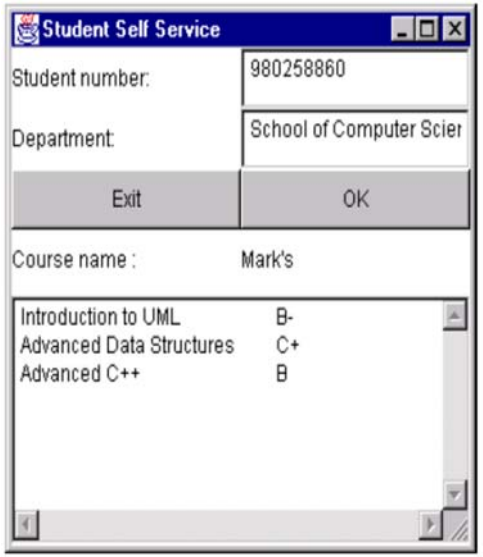
State Machine Model of GUI

- Features are first identified by reading the documents.
  - Each feature involves multiple scenarios

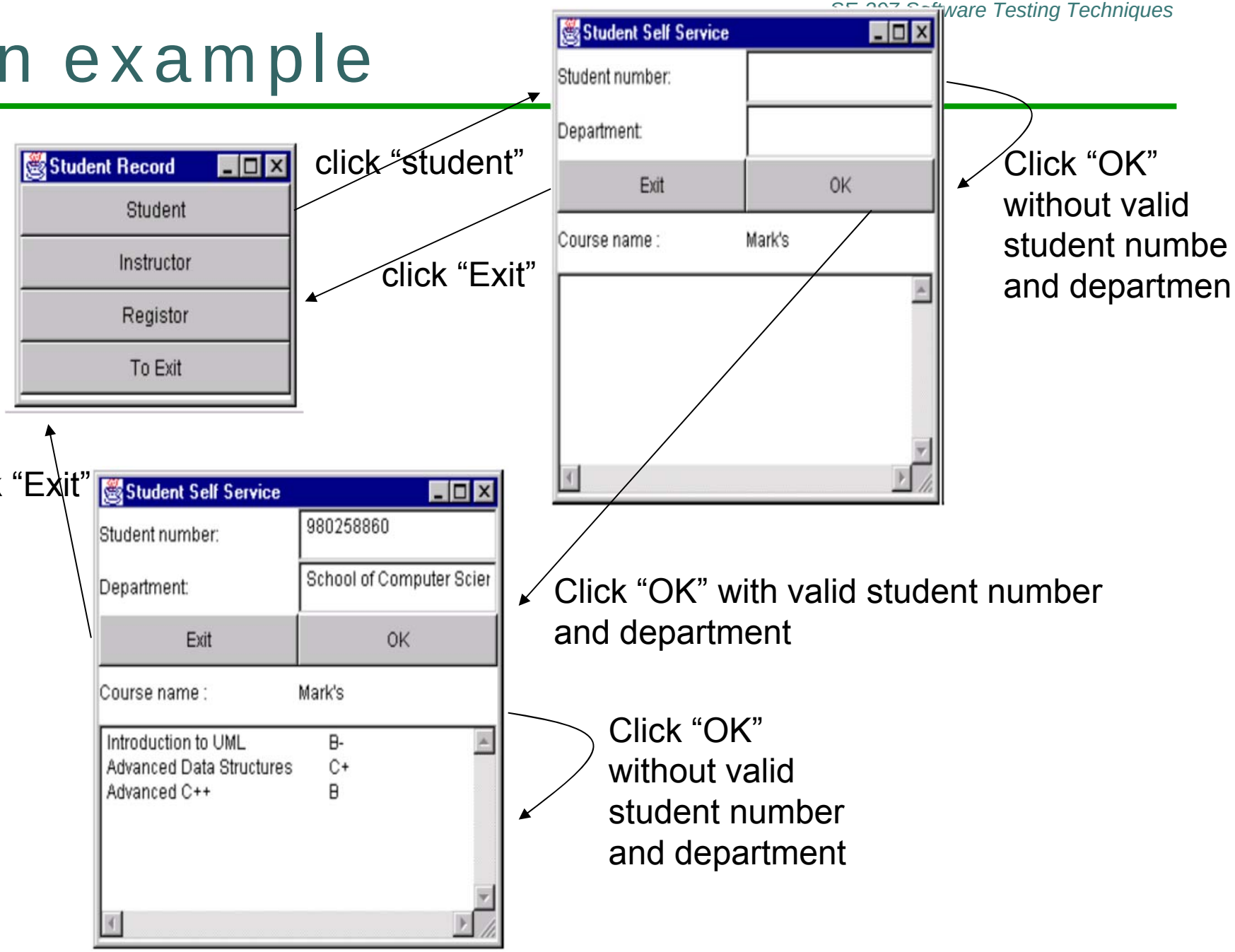
- A finite state machine is constructed for each feature to capture all scenarios of this feature.
  - **States** are intermediate states of the GUI windows
  - **Transitions** are events between states that users trigger.

An example

- The “query student score” features of a student management system
- Scenarios:
  - click “Student” -> Enter valid student ID and department name-> click “OK” -> click “Exit”
  - click “Student” -> Enter invalid student ID and department name-> click “OK” -> click “Exit”
  - click “Student” -> Enter valid student ID and department name-> click “OK” -> Enter another valid student ID and department name-> click “OK” -> click “Exit”
  - …



An example


Student Record
Student
Instructor
Registor
To Exit
click "student"
click "Exit"
Student Self Service
Student number:
Department:
Exit
OK
Course name :
Mark's
Click "OK" without valid student numbe and departmen
click "Exit"
Student Self Service
Student number:
980258860
Department:
School of Computer Scier
Exit
OK
Course name :
Mark's
Introduction to UML B-
Advanced Data Structures C+
Advanced C++ B
Click "OK" with valid student number and department
Click "OK" without valid student number and department

Notation for finite state machine

We will use ***UML State Diagrams***

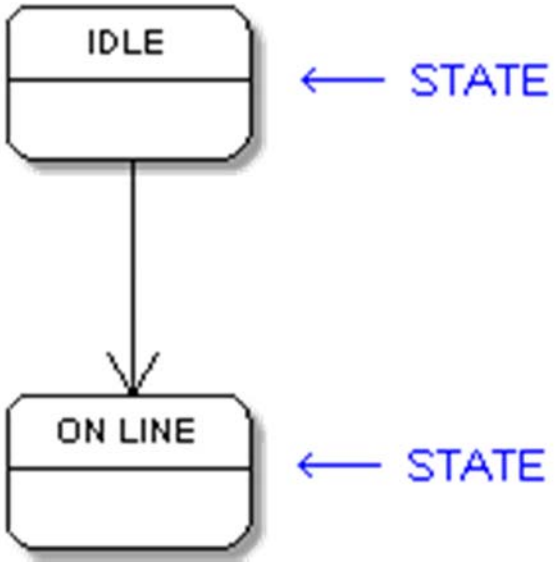
- The following is a brief summary of the notation and behaviour of state charts.

- For a full presentation of the UML state chart notation, see the UML 2.0 specification, available at:

www.omg.org/technology/documents/modeling_spec_catalog.htm

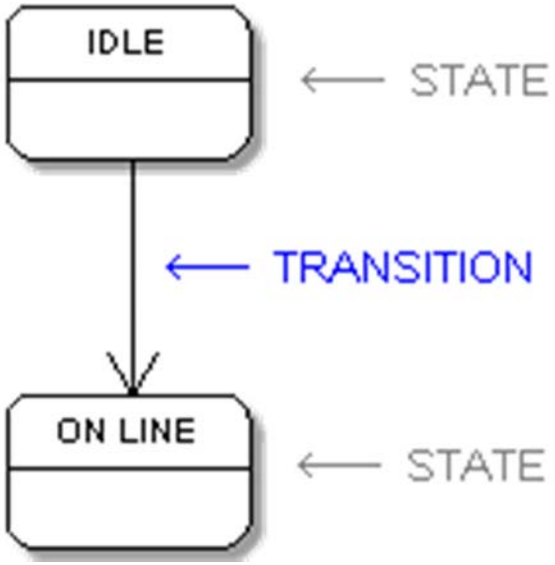
UML state notation

Graphically, UML shows states as boxes with rounded corners



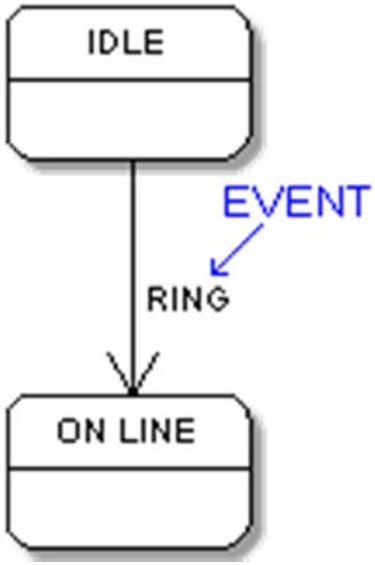
UML transitions notation

Graphically, UML shows states as boxes with rounded corners and transitions as arrows lines.



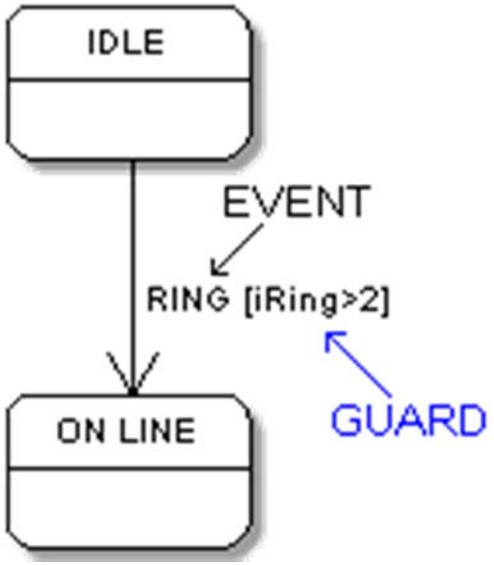
UML transitions notation

Graphically, UML shows states as boxes with rounded corners and transitions as arrows lines. The transitions are labelled with the *event* that causes the transition.



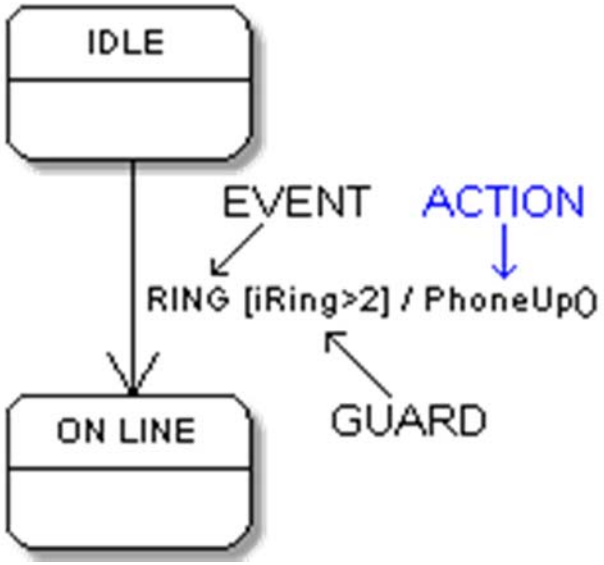
UML transitions notation

Graphically, UML shows states as boxes with rounded corners and transitions as arrows lines. The transitions are labeled with the *event* that causes the transition. A condition called *guard can be indicated* in square brackets…



UML transitions notation

Graphically, UML shows states as boxes with rounded corners and transitions as arrows lines. The transitions are labelled with the *event* that causes the transition. A condition called *guard can be indicated* in square brackets followed by the *action(s)* that will be taken upon transition.



UML states detailed

A state can be divided in 2 areas: the upper for its name

UML states detailed

A state can be divided in 2 areas: the upper for its name and the lower for its actions. Actions are divided in:

- entry actions: executed entering the state

UML states detailed

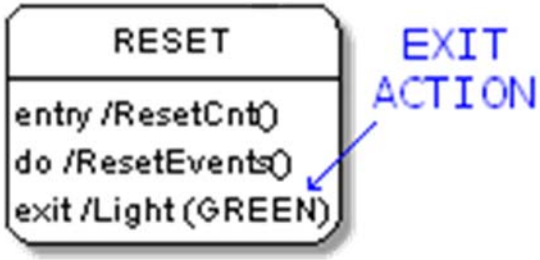
A state can be divided in 2 areas: the upper for its name and the lower for its actions. Actions are divided in:

- entry actions: executed entering the state
- do actions: executed inside the state

UML states detailed

A state can be divided in 2 areas: the upper for its name and the lower for its actions. Actions are divided in:

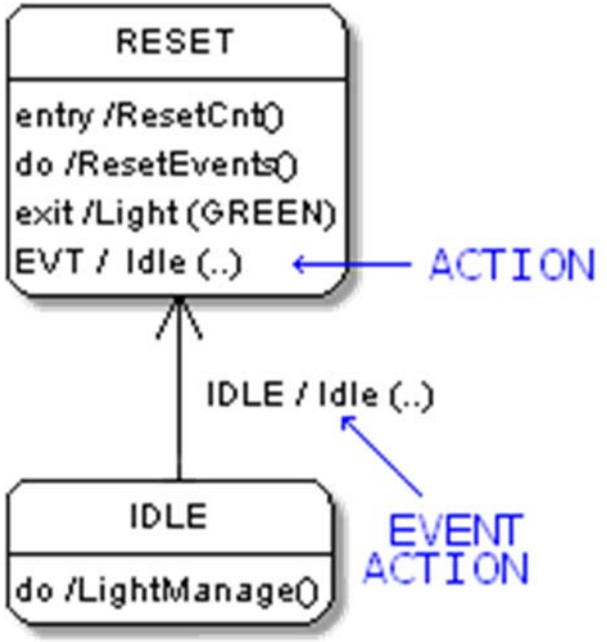
- entry actions: executed entering the state
- do actions: executed inside the state
- exit actions: executed leaving the state



UML states detailed

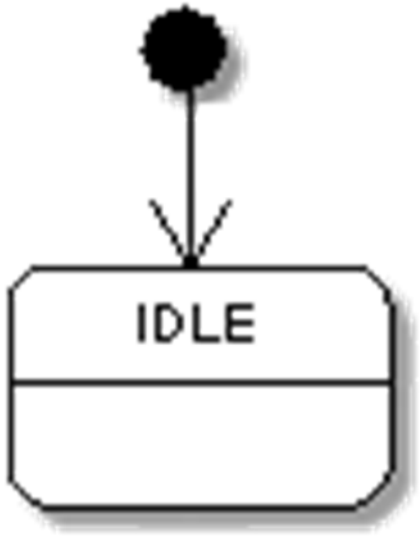
A state can be divided in 2 areas: the upper for its name and the lower for its actions. Actions are divided in:

- entry actions:
  executed entering the state
- do actions:
  executed inside the state
- exit actions:
  executed leaving the state
- event actions:
  executed due to an event
  (specified inside a transition)



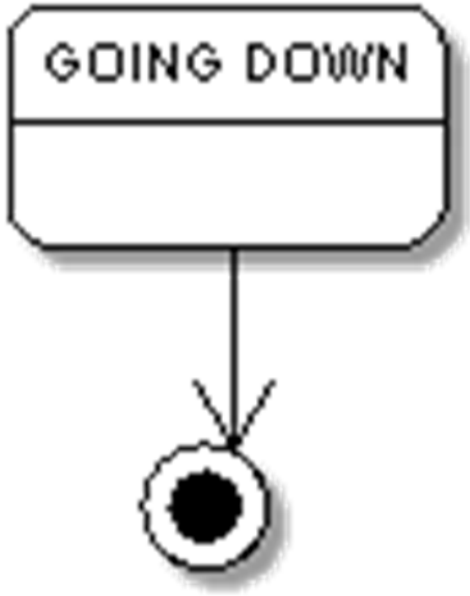
UML pseudo states

- The UML notation for state carts introduces new symbols: a pseudo state to mark the initial state



UML pseudo states

and a pseudo state to mark the final state.



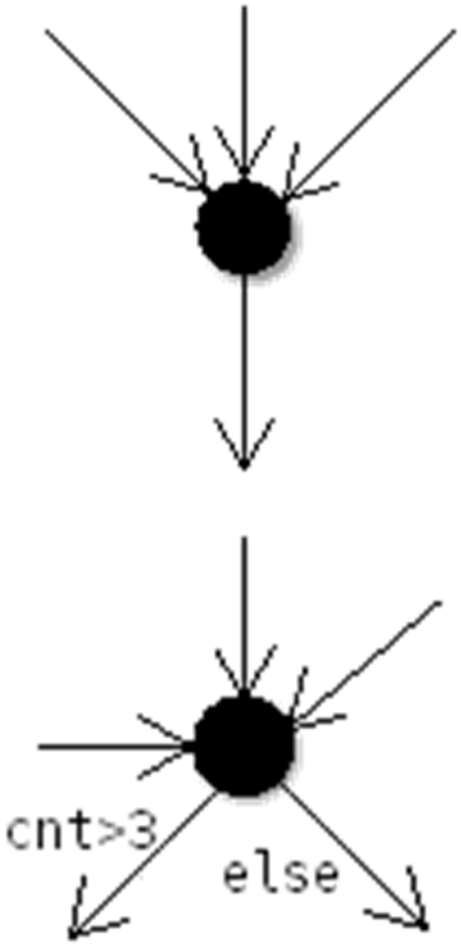
UML pseudo states

Either connects to the states by a transition that may be completed with the notation seen before with event, guard and action fields.

UML junction point

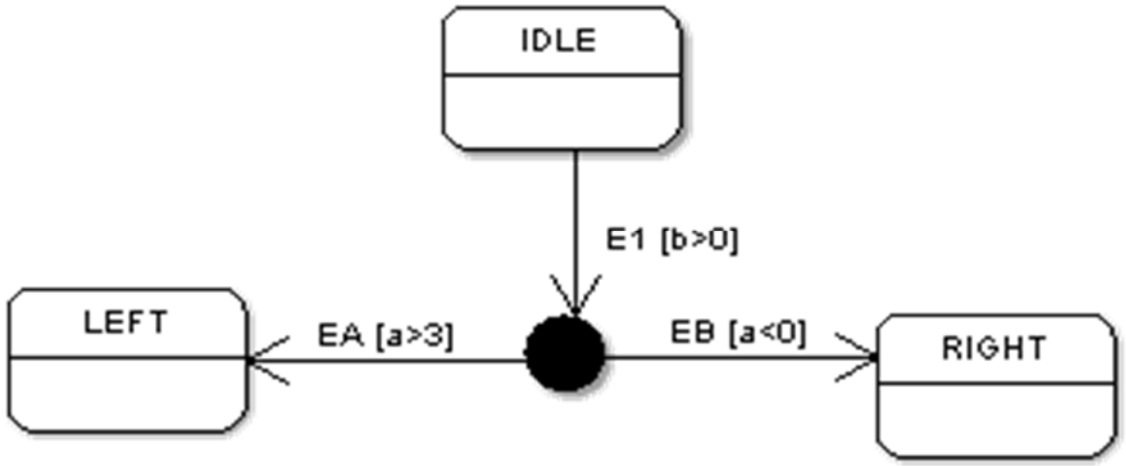
To simplify the graphical design of the state charts UML notation introduces **junction points** and **choice points**.

- Junction points are used to merge and split several transition paths in a state chart diagram because they accept several input and output transitions.



UML junction point

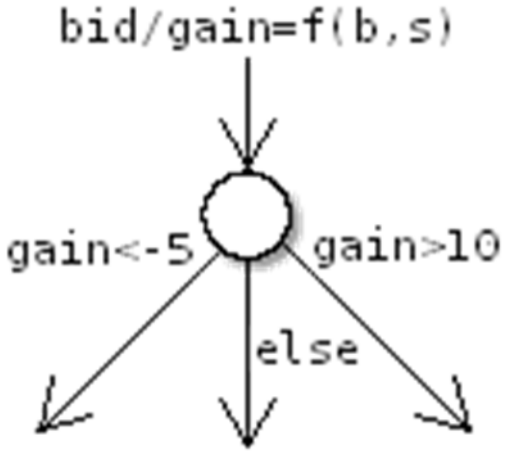
NOTE: it is not mandatory that a junction point has transitions to provide every possible conditions to change state.



UML choice point

A choice point implements a dynamic choice based on the event on the entering transition.

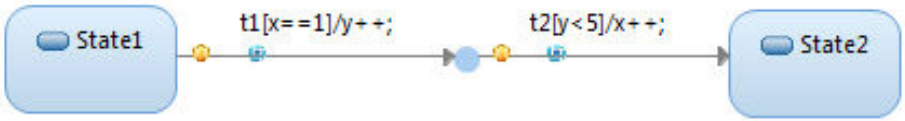
- Always has 1 entering transition and 2 or more outgoing transitions
- The outgoing transactions must cover all possible conditions (we will see why)



Difference between junction and choice

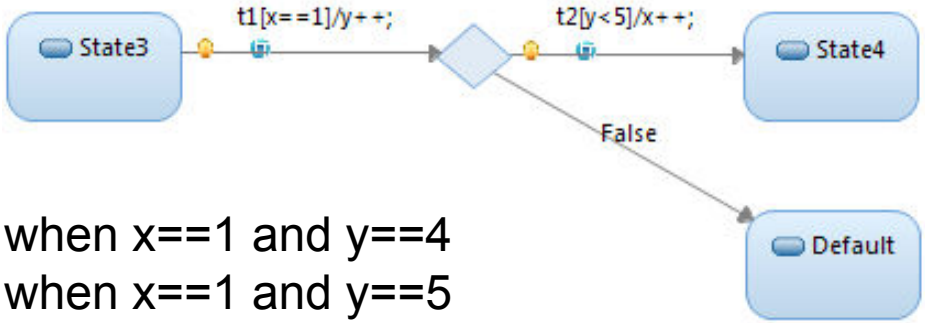
Junction points are static conditional branches, while choice points are dynamic conditional branches.

Junction point



Will fire when x==1 and y==4
Will not fire when x==1 and y==5

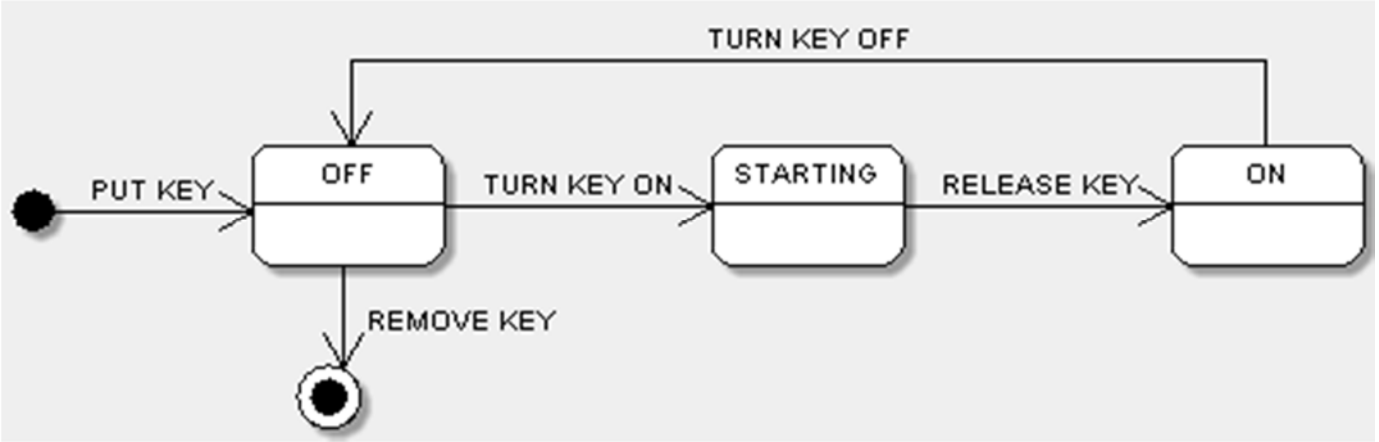
choice point



Will fire when x==1 and y==4
Will fire when x==1 and y==5

State chart example

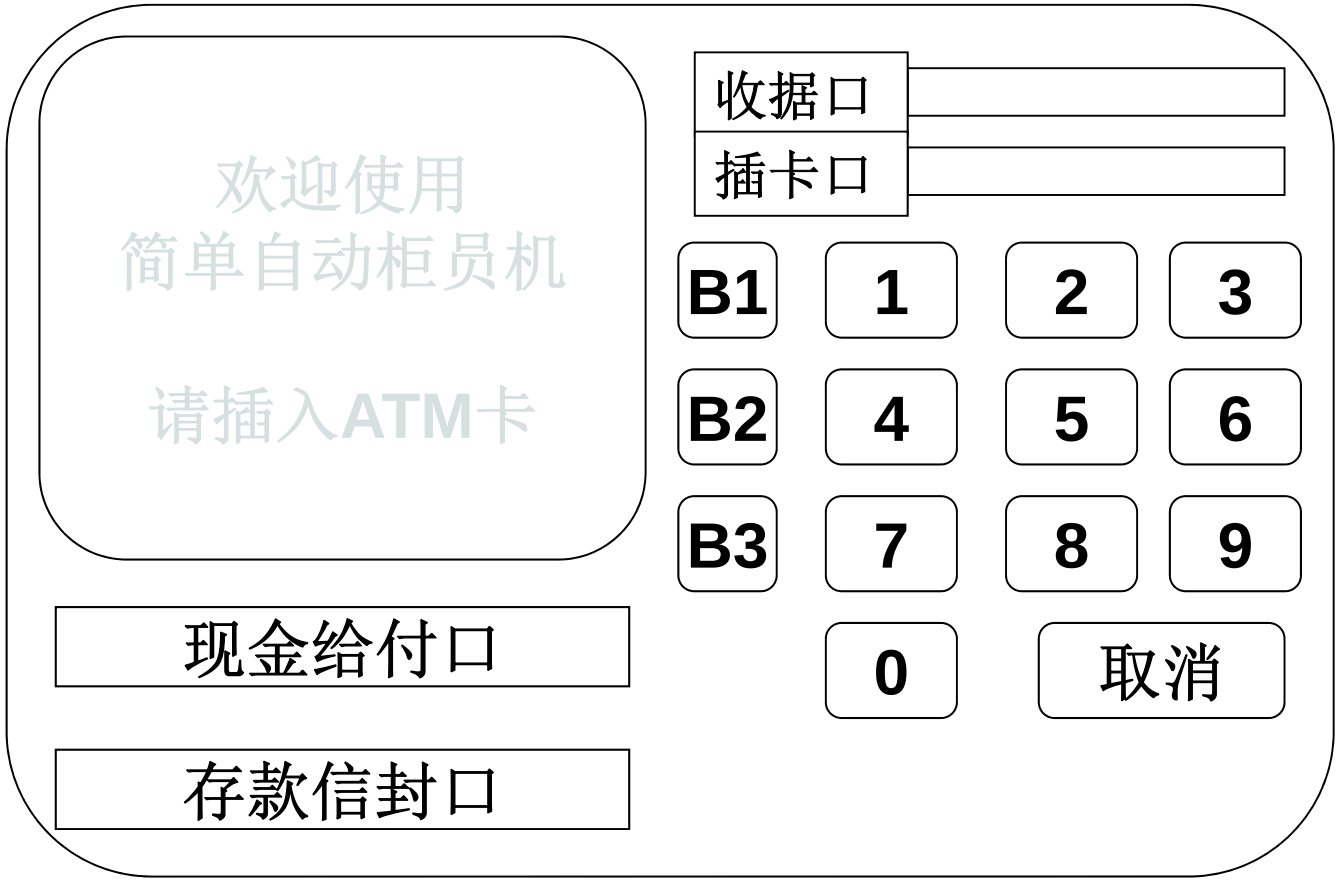
- This state chart describes the procedure to start an engine using a minimum notation:



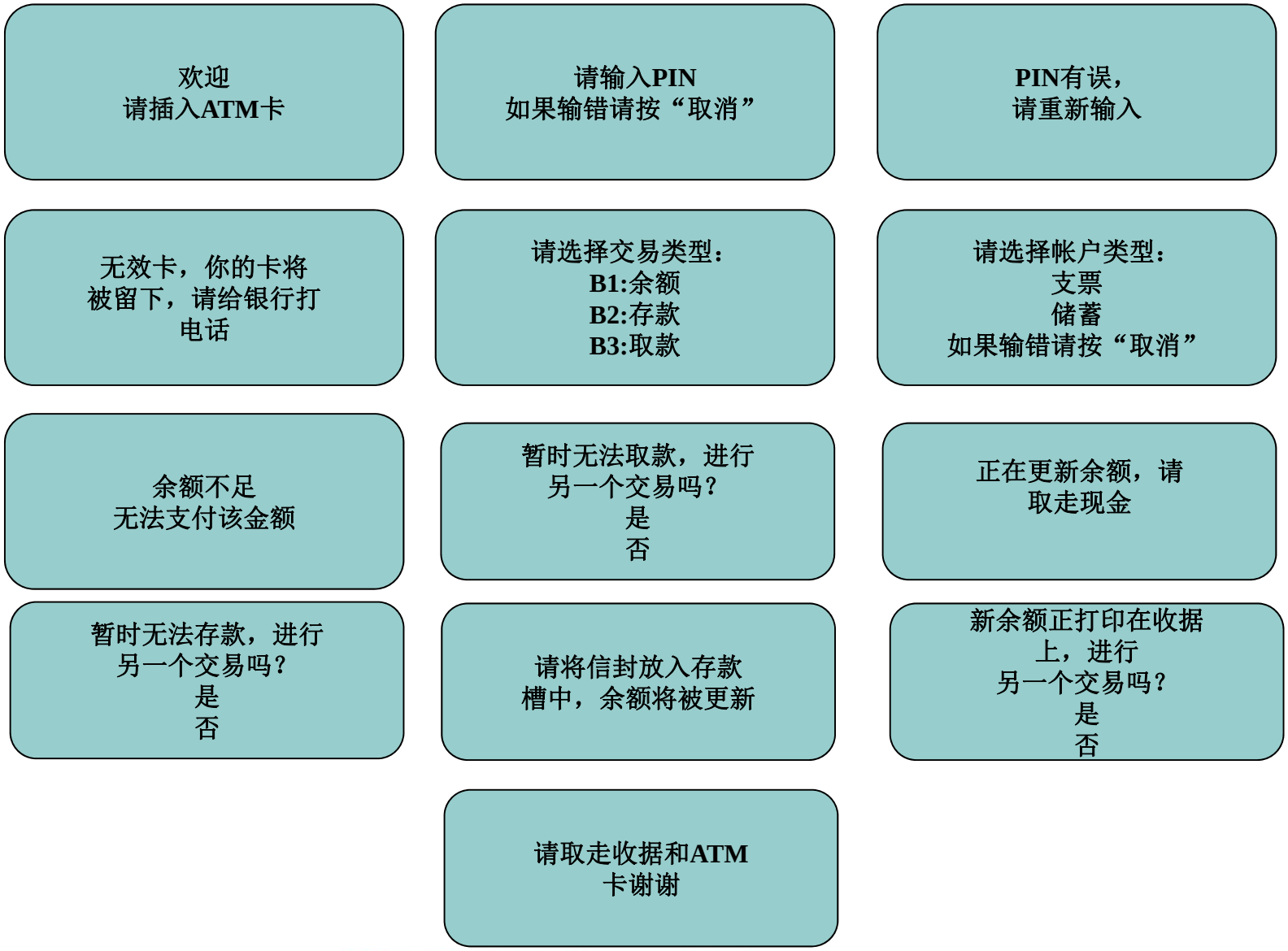
The power of UML notation is due to its flexibility, it may be used at different level of detail depending on the needs.

Putting this to Practice

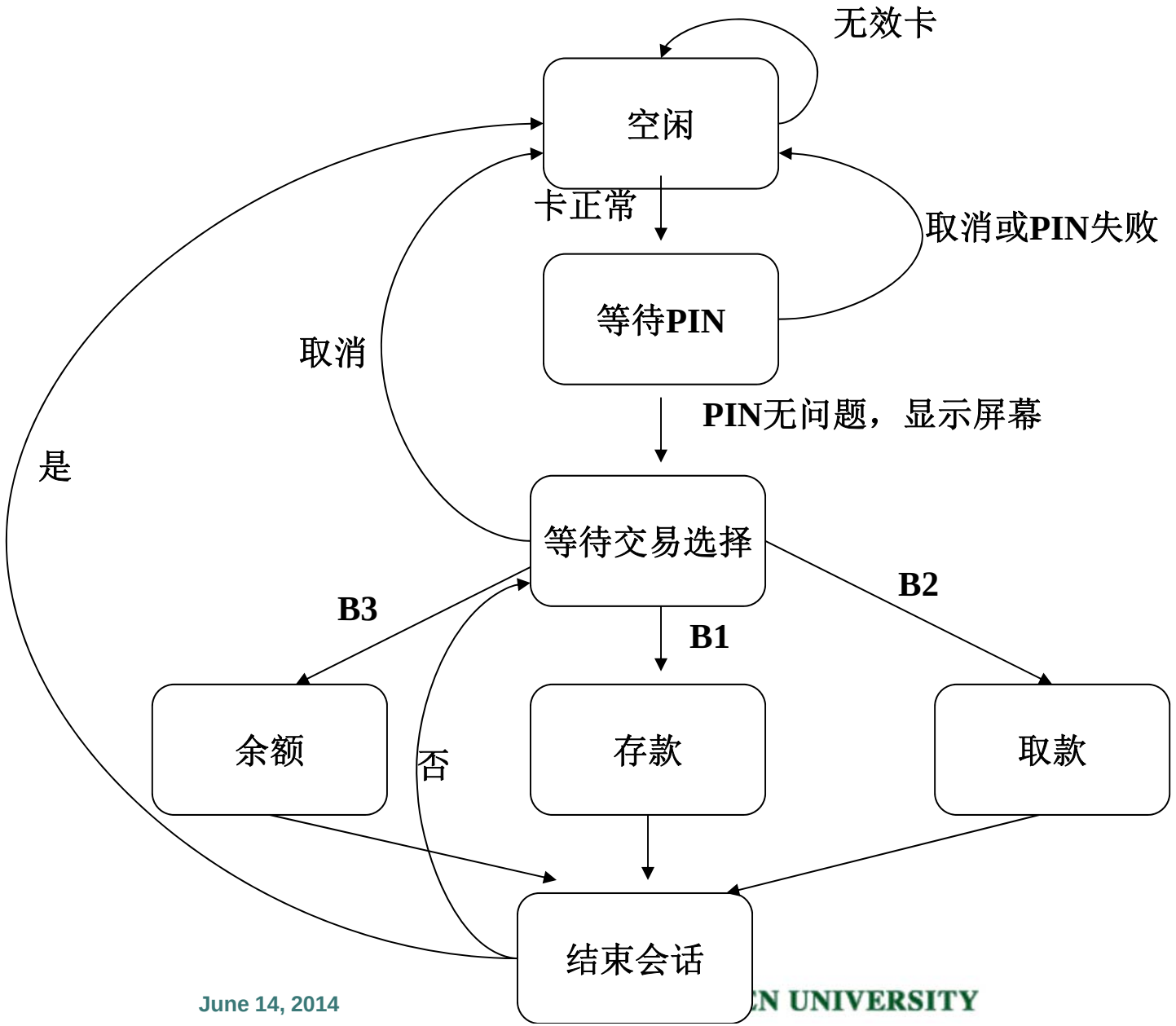
- Consider this GUI of an ATM machine



States


欢迎
请插入ATM卡
请输入PIN
如果输错请按“取消”
PIN有误，
请重新输入
无效卡，你的卡将
被留下，请给银行打
电话
请选择交易类型：
B1:余额
B2:存款
B3:取款
请选择帐户类型：
支票
储蓄
如果输错请按“取消”
余额不足
无法支付该金额
暂时无法取款，进行
另一个交易吗？
是
否
正在更新余额，请
取走现金
暂时无法存款，进行
另一个交易吗？
是
否
请将信封放入存款
槽中，余额将被更新
新余额正打印在收据
上，进行
另一个交易吗？
是
否
请取走收据和ATM
卡谢谢

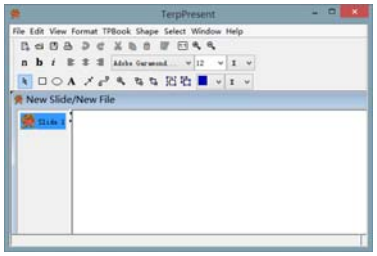
A Simplified State Chart



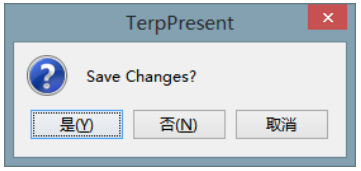
A more complicated example

- The “save file” feature of TerpPresent.
- Scenarios:
  - Open a file, modify it, close the window, select “yes” in “Save Changes?” dialog
  - Open a file, modify it, close the window, select “no” in “Save Changes?” dialog
  - Open a file, modify it, click “SaveAs…” menu item, choose an existing file name in “SaveAs” dialog, click “Save”, click “yes” in “Overwrite?” dialog
  - Open a file, modify it, click “SaveAs…” menu item, choose an existing file name in “SaveAs” dialog, click “Save”, click “no” in “Overwrite?” dialog, click “cancel” in “SaveAs” dialog
  - …….

Active MDI window: modified, unmodified, new, none

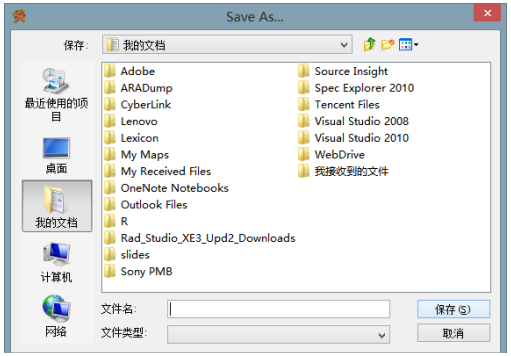


Save Changes? dialog

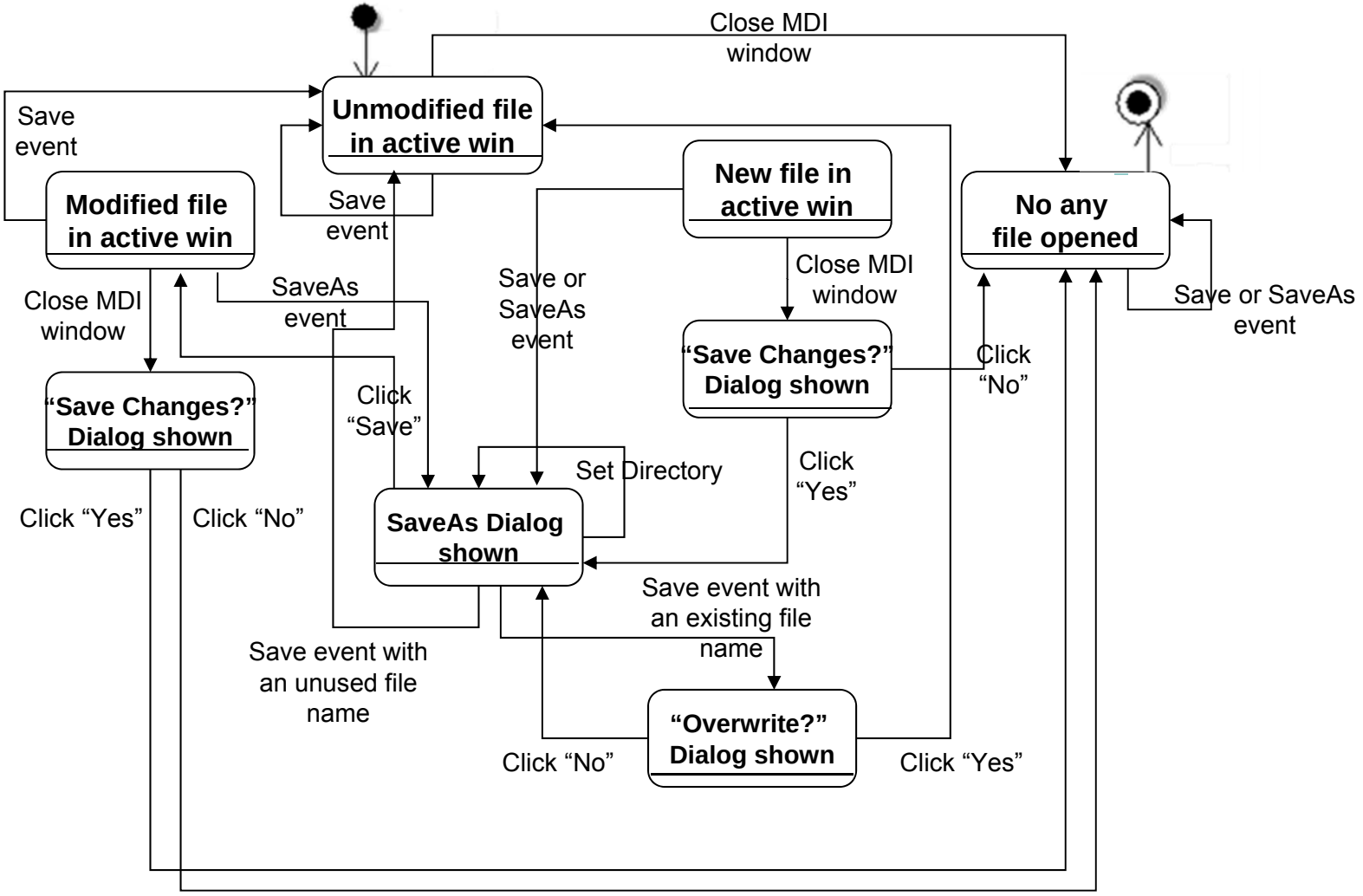


Overwrite? dialog

SaveAs dialog



UML state chart for this feature



State-based Coverage

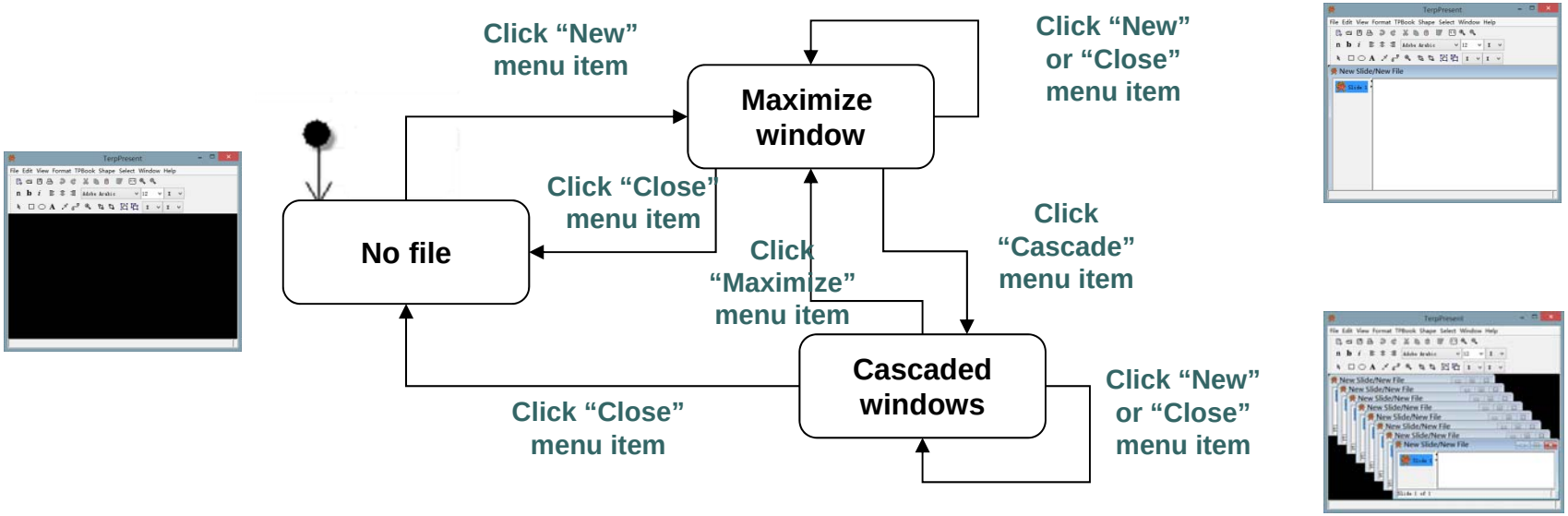
- Generate from the FSM a minimal set of event sequences that:
  - Cover every state
  - Cover every transition
- Scenarios are already event sequences. If we focus on features, why not directly use scenarios as test cases?
  - Why bother creating the FSM?
- Reasons:
  - Scenarios are informal. They don't analyze states and can miss transitions.
  - Test cases generated from FSM can be more efficient by covering many scenarios in one test case. (Scenarios can be unlimited)
- However, for simple features, it might be more efficient to use scenarios directly as test cases.

Typical GUI Errors

- For each test case, looking for the following common errors:
  - Incorrect functioning
  - Missing commands (e.g., GUI events)
  - Incorrect GUI screenshots/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 UI construction
  - ….

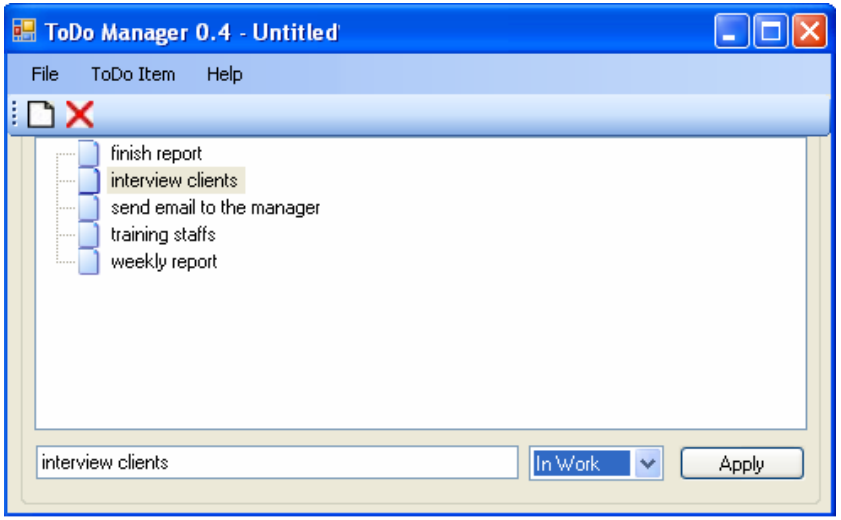
Quiz 1: TerpPresent

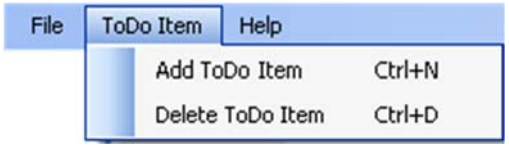
- Feature: “MDI windows arrangement”
- Scenarios
  - Click **New** under **File** menu, Click **Cascade** under **Window** menu, Click **Close** under **File** menu
  - Click **New** under **File** menu, Click **New** under **File** menu, Click **Cascade** under **Window** menu, Click **Close** under **File** menu, Click **Close** under **File** menu
  - Click **New** under **File** menu, Click **New** under **File** menu, Click **Cascade** under **Window** menu, Click **Maximize** under **Window** menu
  - …



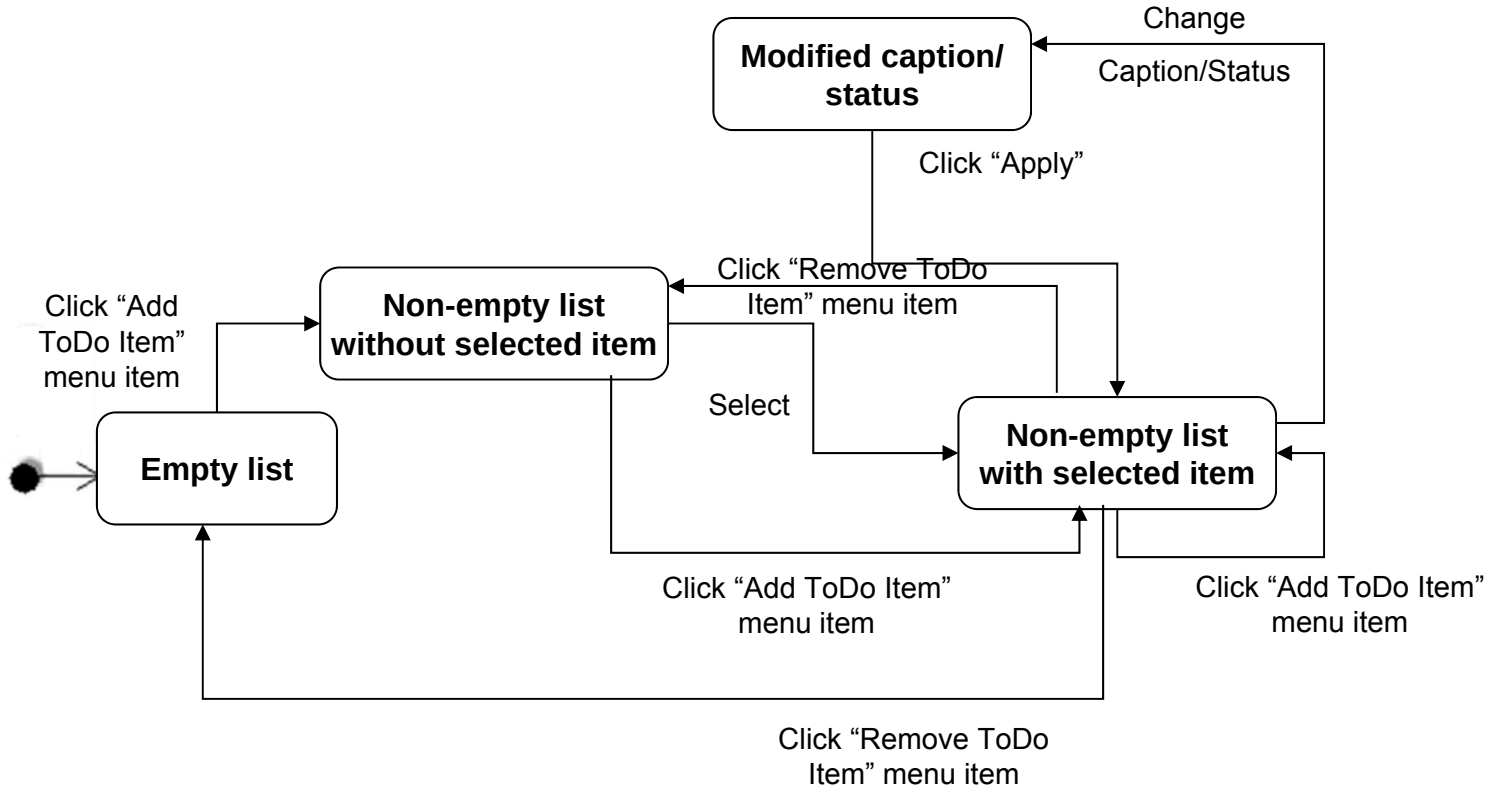
Quiz 2: Todo list

- The “ToDo list editing” feature of a ToDo Manager
- Scenarios:
  - Select an item, change the caption, change the status, click apply
  - Click “Add ToDo Item” menu item
  - Select an item, click “Delete ToDo Item” menu item

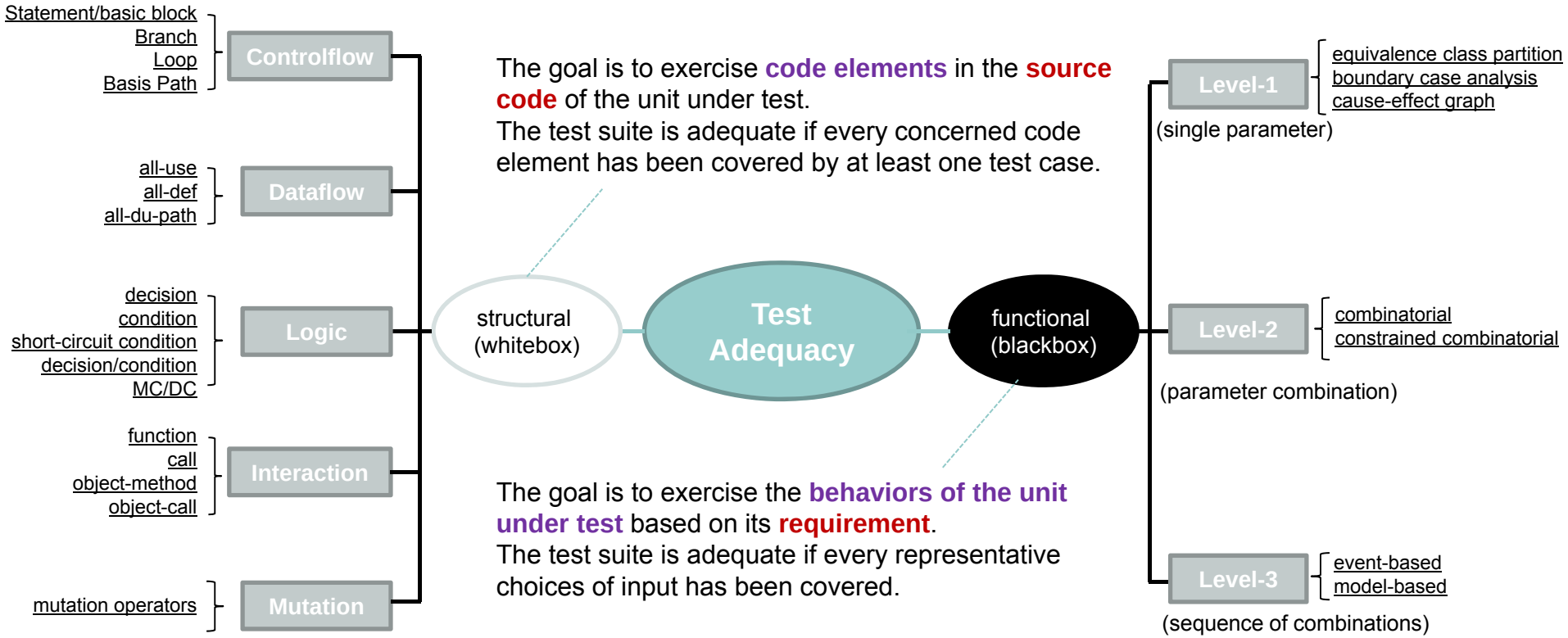




Quiz 2: ToDo list



Summary of the Last Five Lectures



Thank you!