Extending GUITAR
New Test Generation Methods & Models of Interaction

ICSE 2013 Tutorial

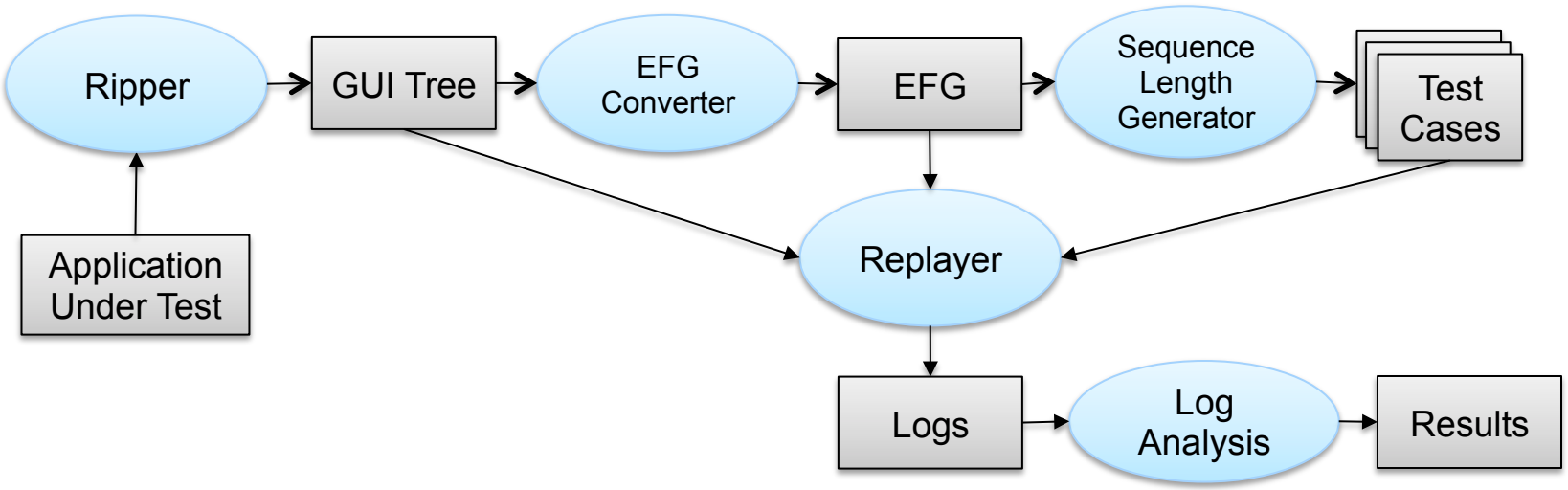
Automated Testing of GUI Applications
Models, Tools and Controlling Flakiness

Atif M. Memon and Myra B. Cohen

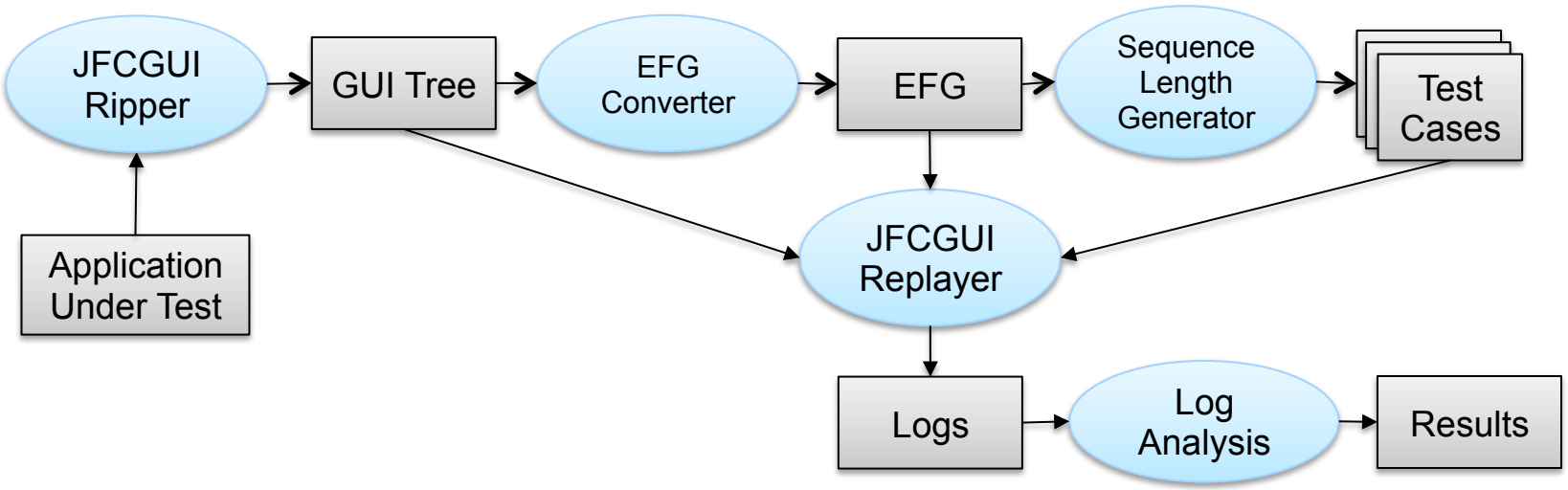
Topics

- GUITAR not just a tool
  - GUI Testing frAmewoRk
  - Extensible framework for GUI testing
- How GUITAR is extensible (Today's Focus)
  - Workflows
  - Widgets and Actions
  - Platforms
    - New platform
  - Algorithms and Tools
    - New test case generator
  - Models
    - New state-machine model
- Code architecture

GUITAR's "Default" Workflow



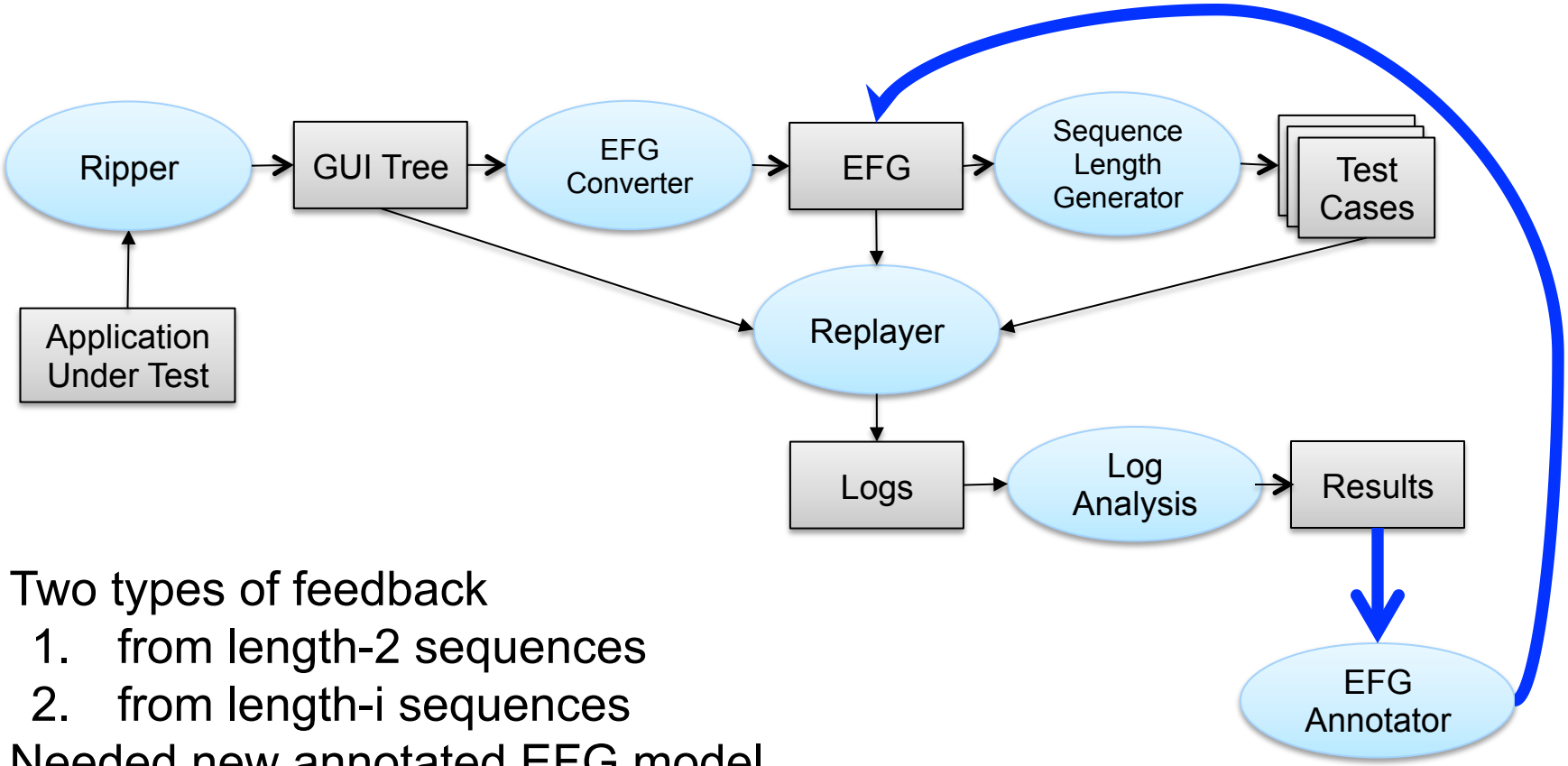
GUITAR's "JFC"+"Default" Workflow



Also Swing, Android, iOS, UNO, Web

Platform-specific Extensions

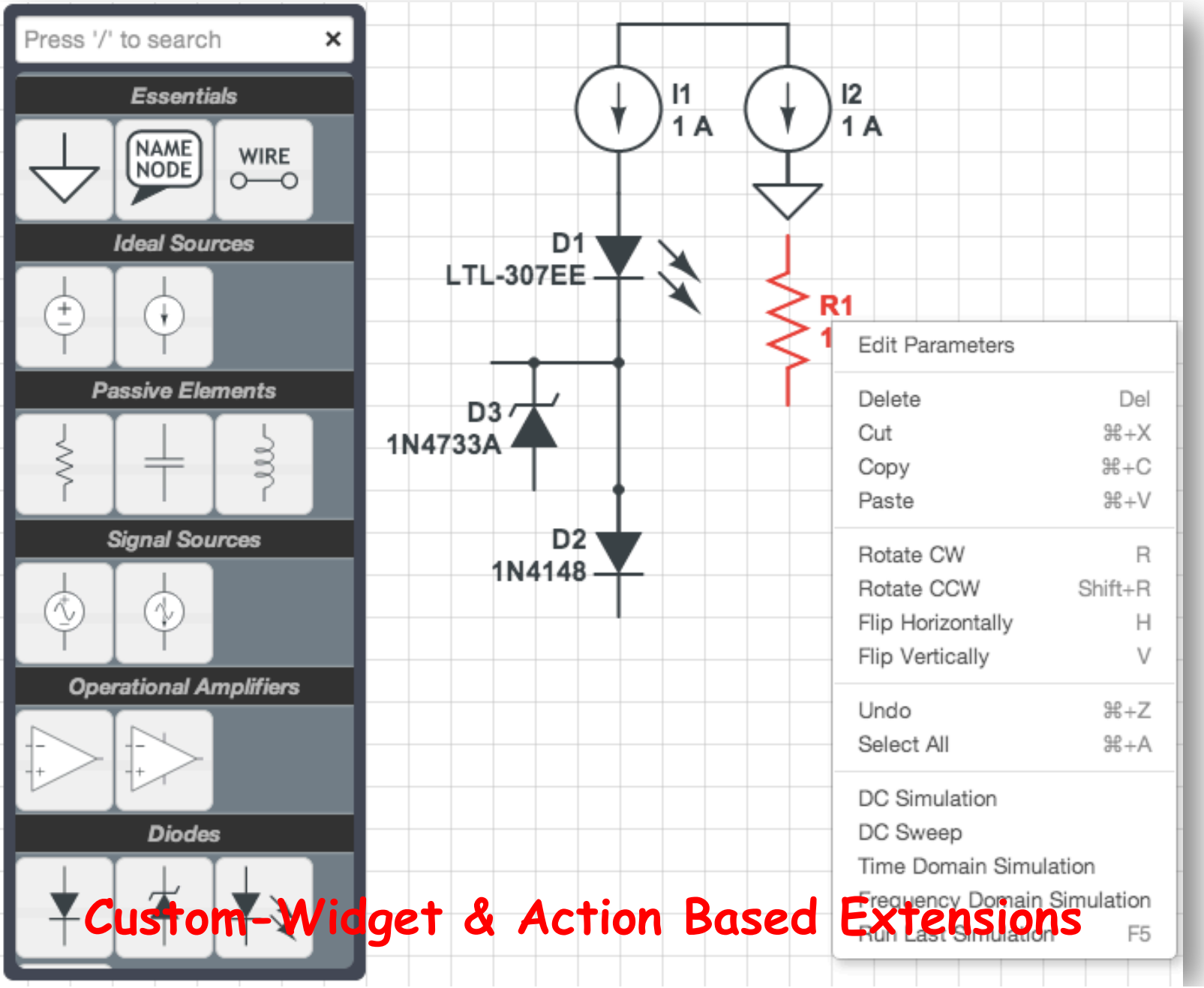
GUITAR's "Feedback-Based" Workflow



- Two types of feedback
  1. from length-2 sequences
  2. from length-i sequences
- Needed new annotated EFG model
- Needed 2 sequence generators
  1. "Default" based on EFG
  2. "New" based on annotations

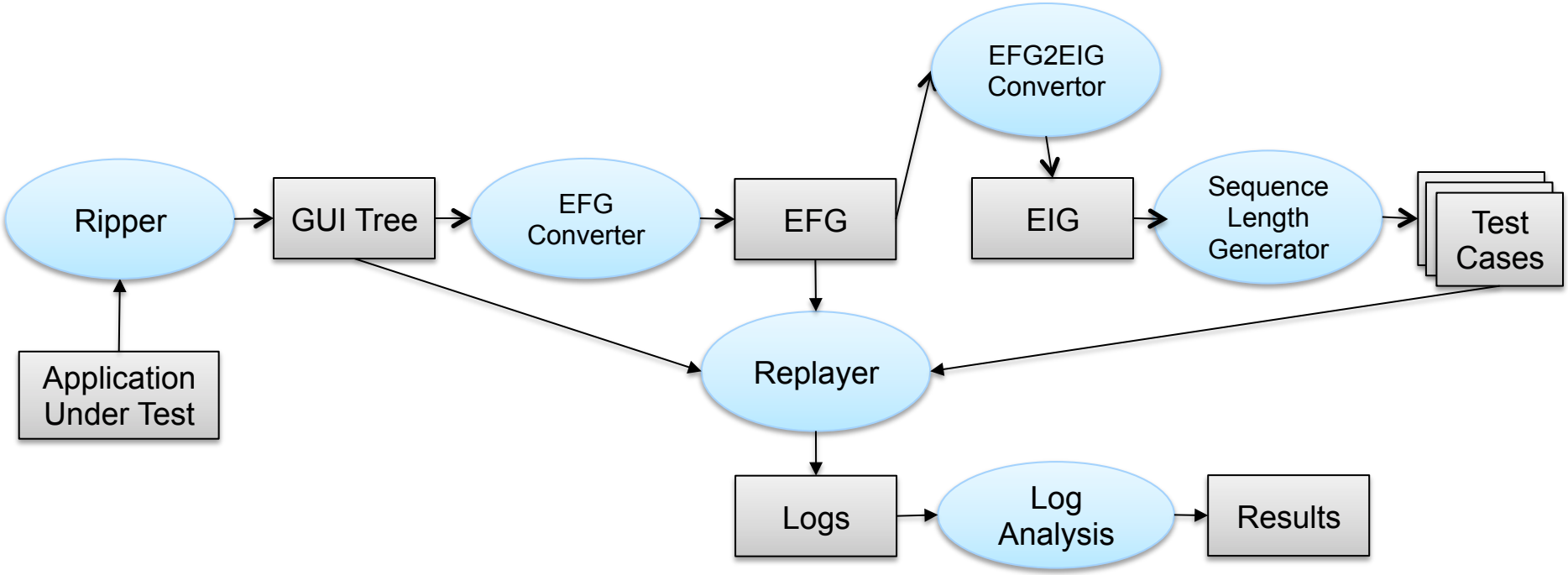
Workflow-Based Extensions

Custom Widgets & Actions



Custom-Widget & Action Based Extensions

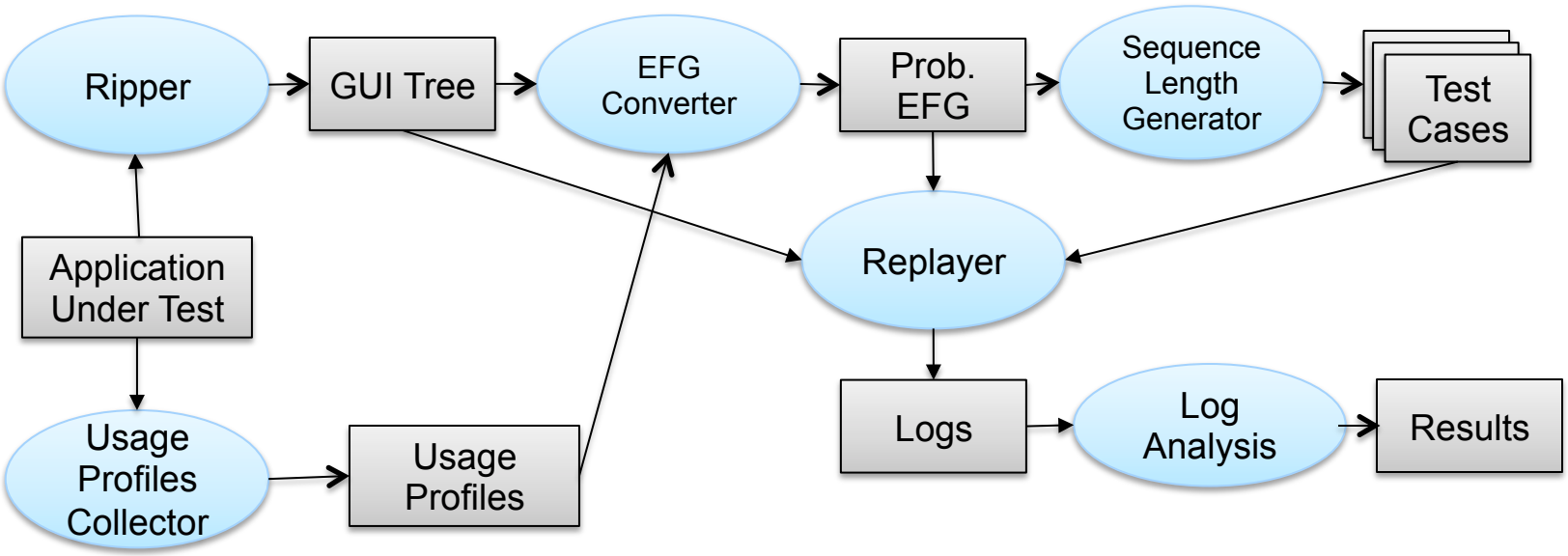
Event Interaction Graph



Needed a new EIG model and generator

Model Extensions

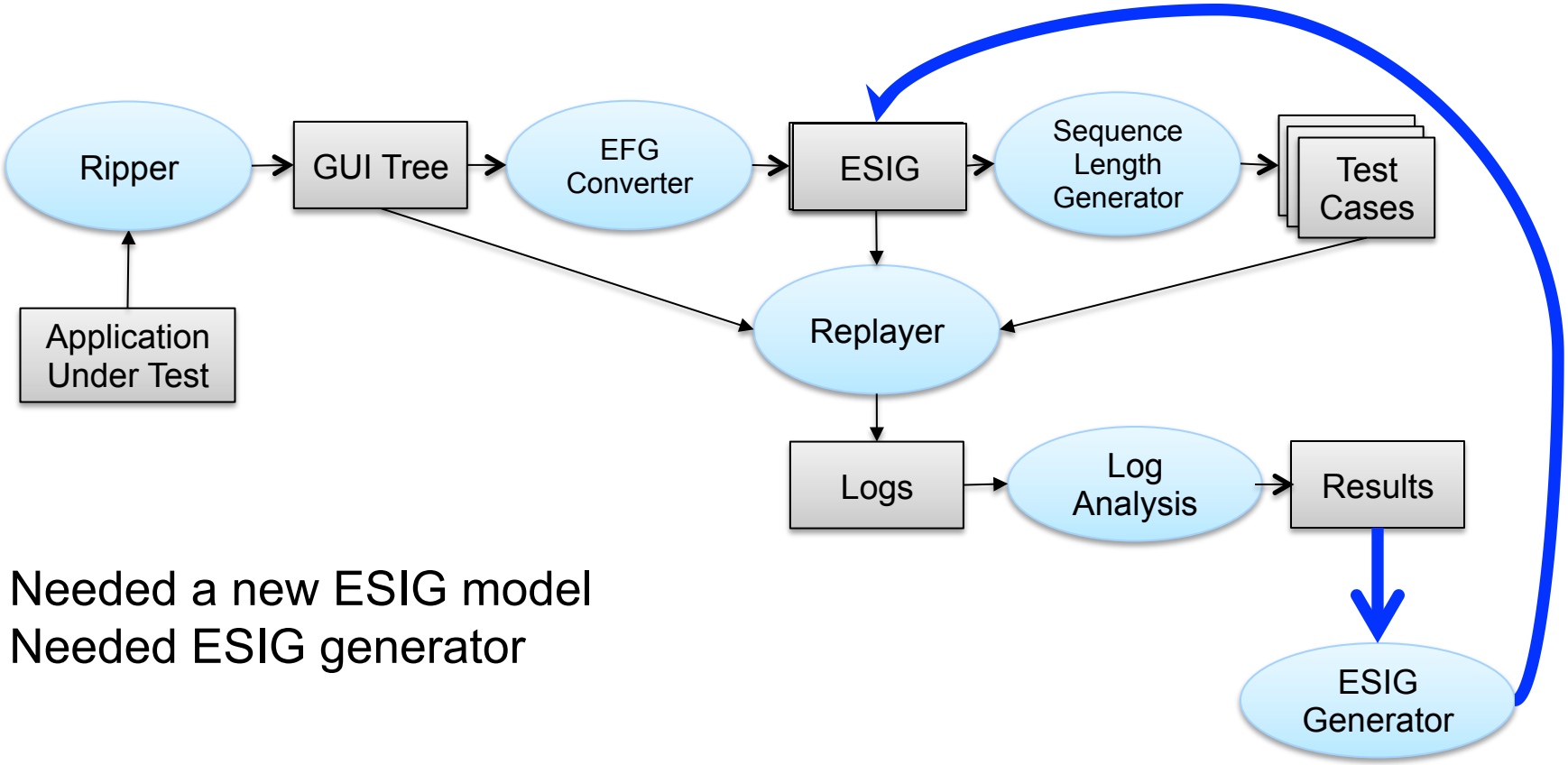
Probabilistic EFG



Needed a new Prob. EFG model and generator

Workflow-Based + Model-Based Extensions

Event Semantic Interaction Graph



- Needed a new ESIG model
- Needed ESIG generator

Workflow-Based + Model-Based Extensions

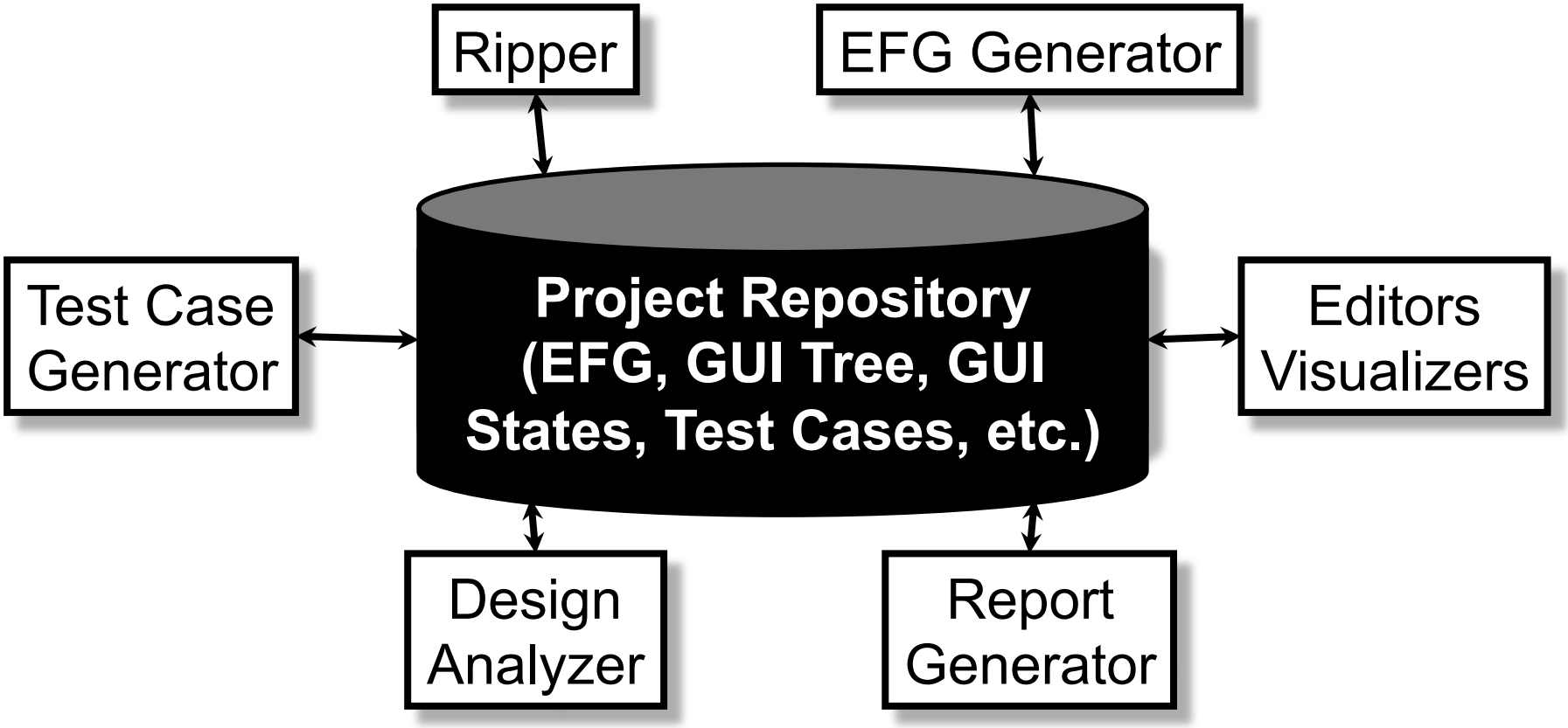
How is GUITAR Extensible?

- Built for Extensibility
  - New models can be plugged in
  - New tools can be added
  - Clear separation between algorithms/models and implementation platform
  - Completely flexible workflow definitions
  - No native GUITAR widgets or interactions

1. What Makes this Possible?

- First Design Decision
  - Definition of First-Class Components
- GUITAR Core
  - Model Core
    - All artifacts defined here
  - Execution Core
    - Ripper and replayer algorithms defined here
  - Analysis Core
    - Pre- and post-execution algorithms defined here
- Basis for all models and tools
  - All models are artifacts
  - Tools are generic programs that use/create artifacts

Repository Model



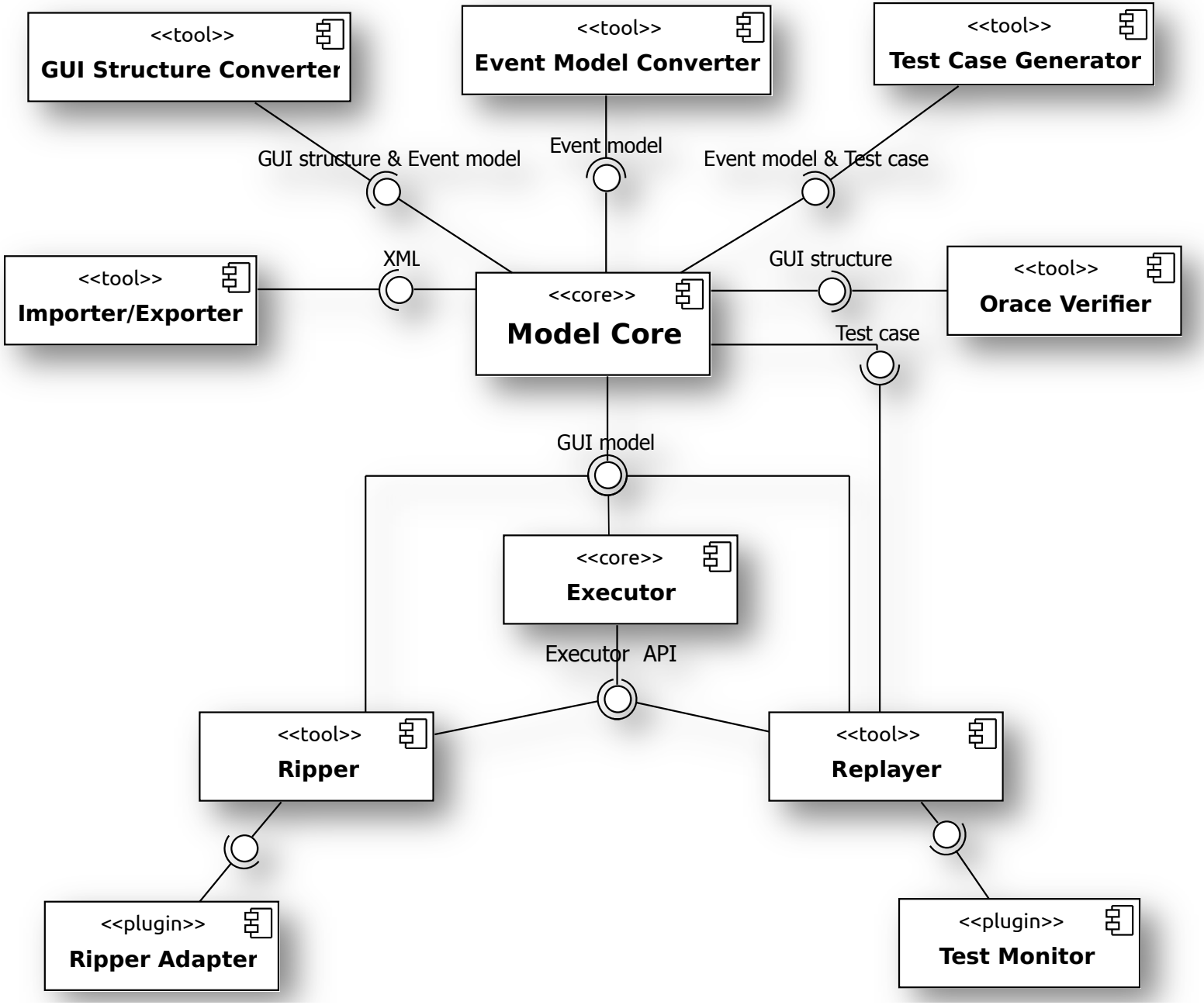
2. What Makes this Possible?

- Second Design Decision
  - Platform Independence
- Algorithms
  - Ripper traversal, GUI Tree creator, Model (EFG) generator, test-case generator
  - All implemented in Java
- Platform specific-extensions
  - Core defines API for platform-specific information
    - "*give me the set of all widgets*"
    - "*give me the set of all windows*"
    - Perform an action (click on a button)
    - Launch the application under test
  - Extensions implement API

3. What Makes this Possible?

- Third Design Decision
  - Generic ***Object*** of Interaction
- Execution Core interacts only with ***Object***
  - All “widgets” instances of ***Object***
  - Each widget
    - Defines own mode of interaction
      - E.g., onClick()
    - Defines own “state getter”
      - E.g., getState()
  - Holds for native platform widgets too
    - Java Swing
    - iOS
    - Definitions are trivial

High-Level Architecture



What this Means in Practice

- Lets create a new model
  - State-machine model
- Lets support a new platform
  - iOS
- Lets create a new test case generator
  - Based on program slicing

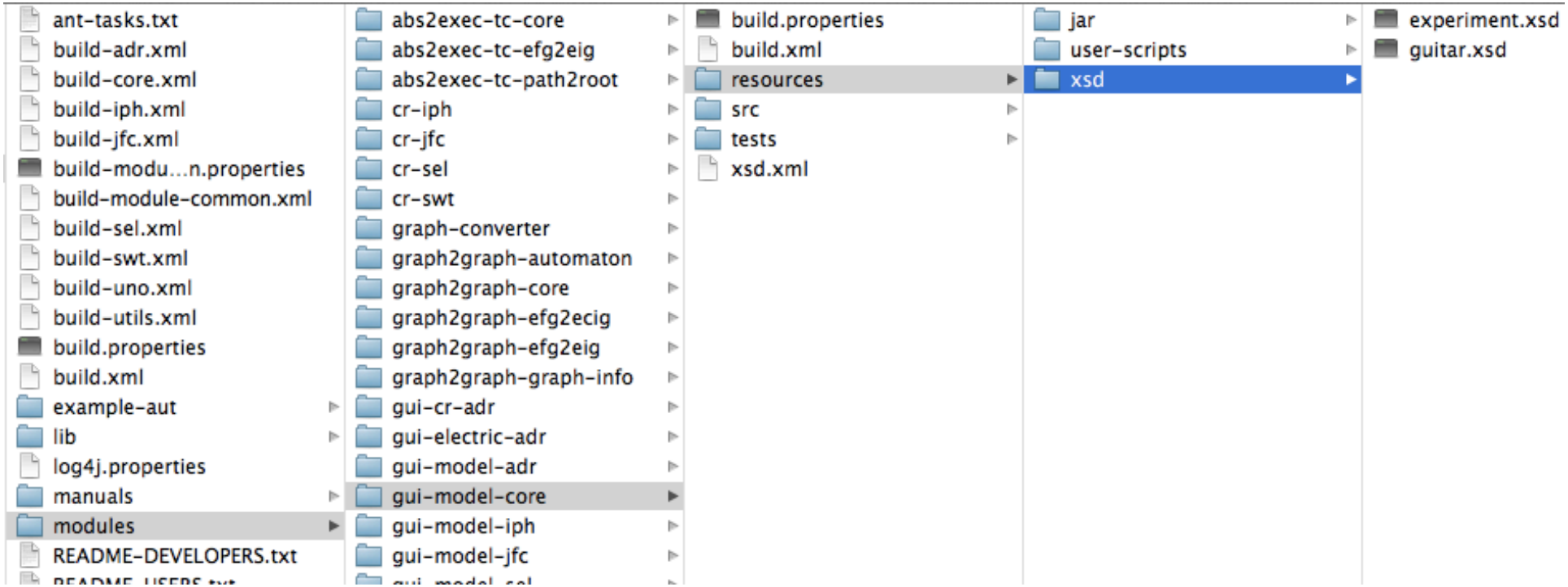
Defining New State Machine Model

- Ripper creates state-machine instead of EFG
- "state" defined as "*set of available events*"
  - We'll call it an "event machine model"
- XML for all artifacts
- XML Schema

Definition of *EventMachine*

```
<xs:element name="EventMachine">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="EventStateSet" type="EventStateSetType"/>
                <xs:element name="TransitionSet" type="TransitionSetType"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="EventStateSetType">
        <xs:sequence>
            <xs:element maxOccurs="unbounded" minOccurs="0" name="EventState"
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="EventStateType">
        <xs:sequence>
            <xs:element name="EventStateId" type="xs:string"/>
            <xs:element name="Initial" type="xs:boolean"/>
            <xs:element name="EventSet" type="EventSetType"/>

        </xs:sequence>
    </xs:complexType>
```

New Definition Added to GUITAR src/



- Add to guitar.xsd
- GUITAR compiles
  - Creates Java objects
- Import/Export
- Syntax checking

New Platform

- Need to Implement platform-independent Executor API
  - API is contract between platform-specific GUI Automation components and high-level platform-independent parts of Model Core
  - Executor API interfaces with all other GUITAR components
    - Once Executor API is implemented, platform-specific components of Executor communicate with rest of GUITAR in platform-independent way

Executor API's Four Interfaces

- GApplication
  - Represents GUI application
    - E.g., initialize applications, start and terminate GUI, access window handlers
- GWindow
  - Represents GUI window
    - E.g., access window properties
- GComponent
  - Represents GUI component (e.g., a widget)
    - E.g., access component properties
- GEvent
  - Represents event type such as left-click, right-click, and text entry
- GEvent paired with the GComponent represents a specific GUI event on a GUI component (e.g., a left-click on the OK button)

Peek into GApplication

```
public abstract class GApplication {

    private static ObjectFactory factory = new ObjectFactory();

    /**
     *
     * Get current GUI state of the application.
     *
     * <p>
     *
     * @return The <code> GUIStructure </code> of the current state
     * @see GUIStructure
     */
    public GUIStructure getCurrentState() {

        GUIStructure guiStructure = factory.createGUIStructure();
        Set<GWindow> lWindows = getAllWindow();
        List<GUIType> lGUIs = new ArrayList<GUIType>();
        for (GWindow gWindow : lWindows) {
            if (gWindow.isValid()) {
                GUIType dGUI = extractDeepGUI(gWindow);
                lGUIs.add(dGUI);
            }
        }
        guiStructure.setGUI(lGUIs);
        return guiStructure;
    }
    public abstract Set<GWindow> getAllWindow();
```

Swing GUITAR (Sitar)

- SitarApplication.java

```
@Override
public void connect() {
    try {

        // sleep because user said so
        if (initialWait > 0) {
            Thread.sleep(initialWait);
        }

        // sleep because we have to
        int sleepIncrement = 100;
        int totalSleepTime = 0;

        while ((guiDisplay = Display.findDisplay(guiThread)) == null) {
            GUITARLog.log.debug("GUI not ready yet");

            // wait forever if timeout == 0
            if (timeout != 0 && totalSleepTime > timeout) {
                GUITARLog.log.error("Timed out waiting for GUI to start");
                throw new ApplicationConnectException();
            }
            GUITARLog.log.debug("Waiting for GUI to initialize for: "
                    + sleepIncrement + "ms");
            Thread.sleep(sleepIncrement);
            totalSleepTime += sleepIncrement;
        }

        // make sure display is not only non-null, but ready
        Thread.sleep(sleepIncrement);
```

IphApplication.java

```
public class IphApplication extends GApplication {
    int

    publ
```

```
@Override
public void connect(String[] args) throws ApplicationConnectException {
    GUITARLog.log.debug("============================");
    GUITARLog.log.debug("Application Parameters: ");
    GUITARLog.log.debug("----------------------------");
    for (int i = 0; i < args.length; i++)
        GUITARLog.log.debug("\t" + args[i]);
    GUITARLog.log.debug("");

    IphCommServer.setUpIServerSocket();
    if (IphCommServer.waitForConnection()) {
        IphCommServer.request(IphCommServerConstants.INVOKE_MAIN_METHOD);
        if (IphCommServer.hear() == IphCommServerConstants.APP_LAUNCHED) {
            Sys
        }
    } else {
        System.
        System.
    }

    try {
        Thread.
    } catch (Ir
        // TODO
        GUITARL
    }
}
```

```
@Override
public Set<GWindow> getAllWindow() {
    if (allWindows == null) {
        allWindows = new HashSet<GWindow>();
    } else {
        return allWindows;
    }

    ArrayList<IphWindow> windows = new ArrayList<IphWindow>();
    IphCommServer.requestMainView(windows);
    allWindows.addAll(windows);
    for (IphWindow iWindow : windows) {
        allWindows.add(iWindow);
        System.out.println("Exist " + allWindows.size() + " windows!");
    }
    return allWindows;
}
```

Additional Tasks

- Also implement remaining interfaces
- Store new code in folder structure

- And you're done …
- To execute, simply specify new classname in classpath

- *iOS, Android, Swing GUITARs were implemented in CMSC435, undergraduate software engineering as class projects*

New Test-Case Generator

- Idea
  - Does data flow from one event handler to another?
  - If YES, then test these events together
- Realizing the idea
  - Use program slicing
  - Compute a forward slice from an event handler
  - If other event handlers are part of slice, then there is a possible data flow

Implementation

- Abstract Class
  - GTestCaseGeneratorPlugin

```
public abstract class GTestCaseGeneratorPlugin {

    /**
     *
     */
    f

    /**
     * Generate test cases
     *
     * <p>
     *
     * @param efg
     * @param outputDir
     * @param nMaxNumber
     * @param noDuplicateEvent
     * @param treatTerminalEventSpecially
     */
    abstract public void generate(EFG efg,
                                  String outputDir,
                                  int nMaxNumber,
                                  boolean noDuplicateEvent,
                                  boolean treatTerminalEventSpecially);

    f

    O
    L

    /**
     * Map of <event, all successor events>
     */
    Hashtable<EventType, Vector<EventType>> succs;
```

Existing Based on EFG

```
public class SequenceLengthCoverage extends GTestCaseGeneratorPlugin {
...
...
@Override
public void
generate(EFG efg,
         String outputDir,
         int nReqTestcases,
         boolean noDuplicateEvent,
         boolean treatTerminalEventSpecially) {

   new File(outputDir).mkdir();

   List<EventType> eventList = efg.getEvents().getEvent();
   List<EventType> interactionEventList = new ArrayList<EventType>();

   nGeneratedTestcases = 0;
   nSkippedDup         = 0;
   nSkippedPath        = 0;

   for (EventType event : eventList) {
      if (treatTerminalEventSpecially && isTerminalEvent(event)) {
         terminalEvents.add(event);
      }
      if (isSelectedEvent(event)) {
         interactionEventList.add(event);
      }
   }

   for (EventType e : interactionEventList) {
      LinkedList<EventType> initialList =
         new LinkedList<EventType>();
```

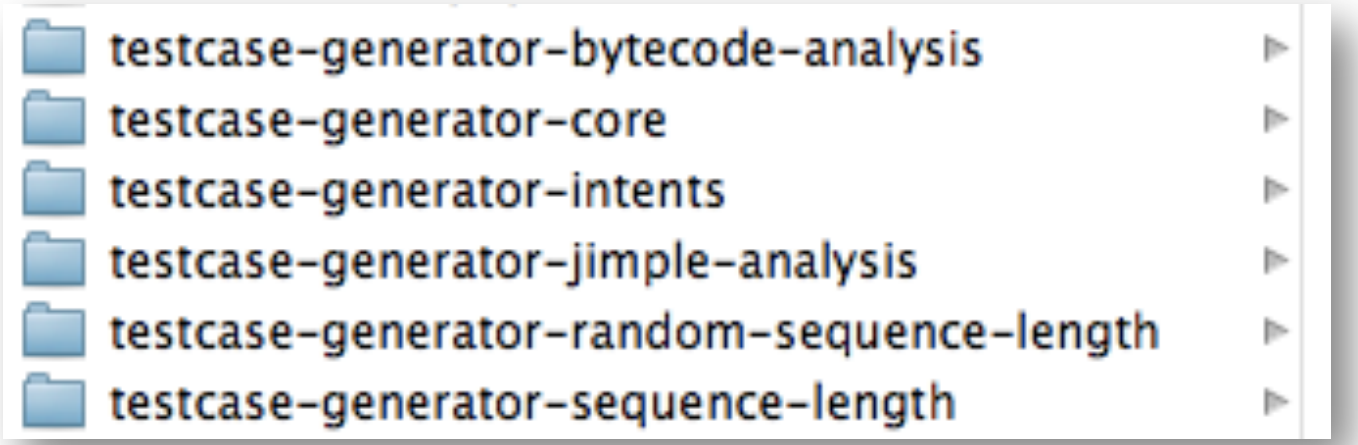
New Test-Case Generator

```
public class JimpleAnalysis extends GTestCaseGeneratorPlugin {

    @Override
    public TestCaseGeneratorConfiguration getConfiguration() {
        return new JimpleAnalysisConfiguration();
    }
```

```
    @Override
    public void generate(EFG efg, String outputDir, int nMaxNumber,
            boolean noDuplicateEvent, boolean treatTerminalEventSpecially) {
        ...
        ...
        // config body transformer
        PackManager.v().getPack("jtp")
                .add(new Transform("jtp.myTransform", bodyTransformer));

        // run body transformer
        soot.Main.main(new String[] { "-output-format", "n", "-pp", "-cp",
                cp.toString(), "-process-dir",
                JimpleAnalysisConfiguration.SCOPE });

        // run slicer
        CTSlicer slicer = new CTSlicer(bodyTransformer);
        slicer.run(events);

        // run sequence selector
        CTSequenceSelector selector = new CTSequenceSelector(slicer, events);
        selector.run(JimpleAnalysisConfiguration.LENGTH);
```

Additional Tasks

- Store new code in folder structure



- And you're done …
- To execute, simply specify new classname in classpath of test-case generator

- *Several new test-case generators were implemented as class projects in CMSC737, graduate class on software testing.*

Concluding Remarks

- GUI Testing frAmewoRk is not just a tool
- Very briefly, showed
  - Support for new platforms
  - How to incorporate new algorithms and tools
  - How to define and use new models
- Additional reading
  - Upcoming article***: GUITAR: An Innovative Tool for Automated Testing of GUI-Driven Software*** by Bao N. Nguyen, Bryan Robbins, Ishan Banerjee, and Atif Memon