ADAutomation: An Activity Diagram Based Automated GUI Testing Framework for Smartphone Applications

Ang Li, Zishan Qin, Mingsong Chen* and Jing Liu

Shanghai Key Lab of Trustworthy Computing

East China Normal University, China

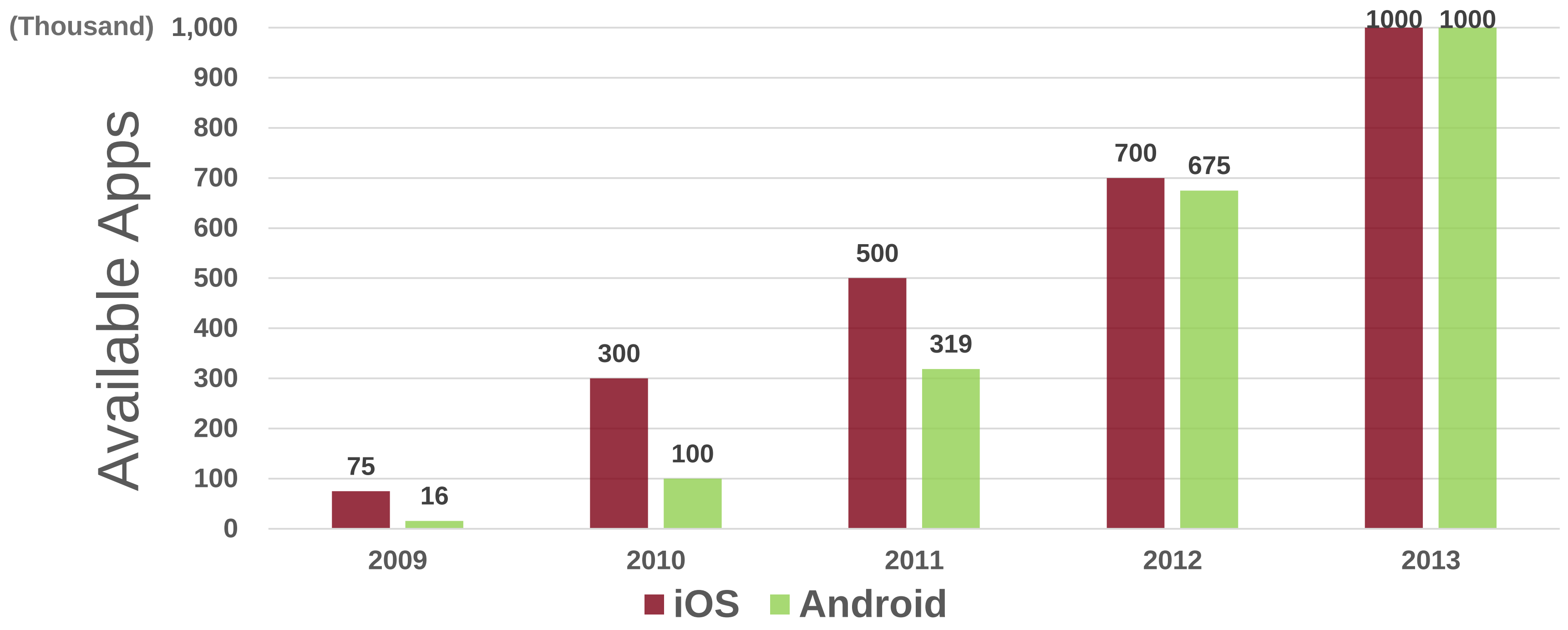
June 30, 2014

Outline

Outline

- Introduction
- Behavior Modeling Using Activity Diagrams
  - Graph–based Notations
  - User Behavior Modeling & Extraction
- Our Automated GUI Testing Framework
  - Test Script Library Generation
  - GUI Testing & Error Diagnosis
- Experiments
- Conclusion

Introduction



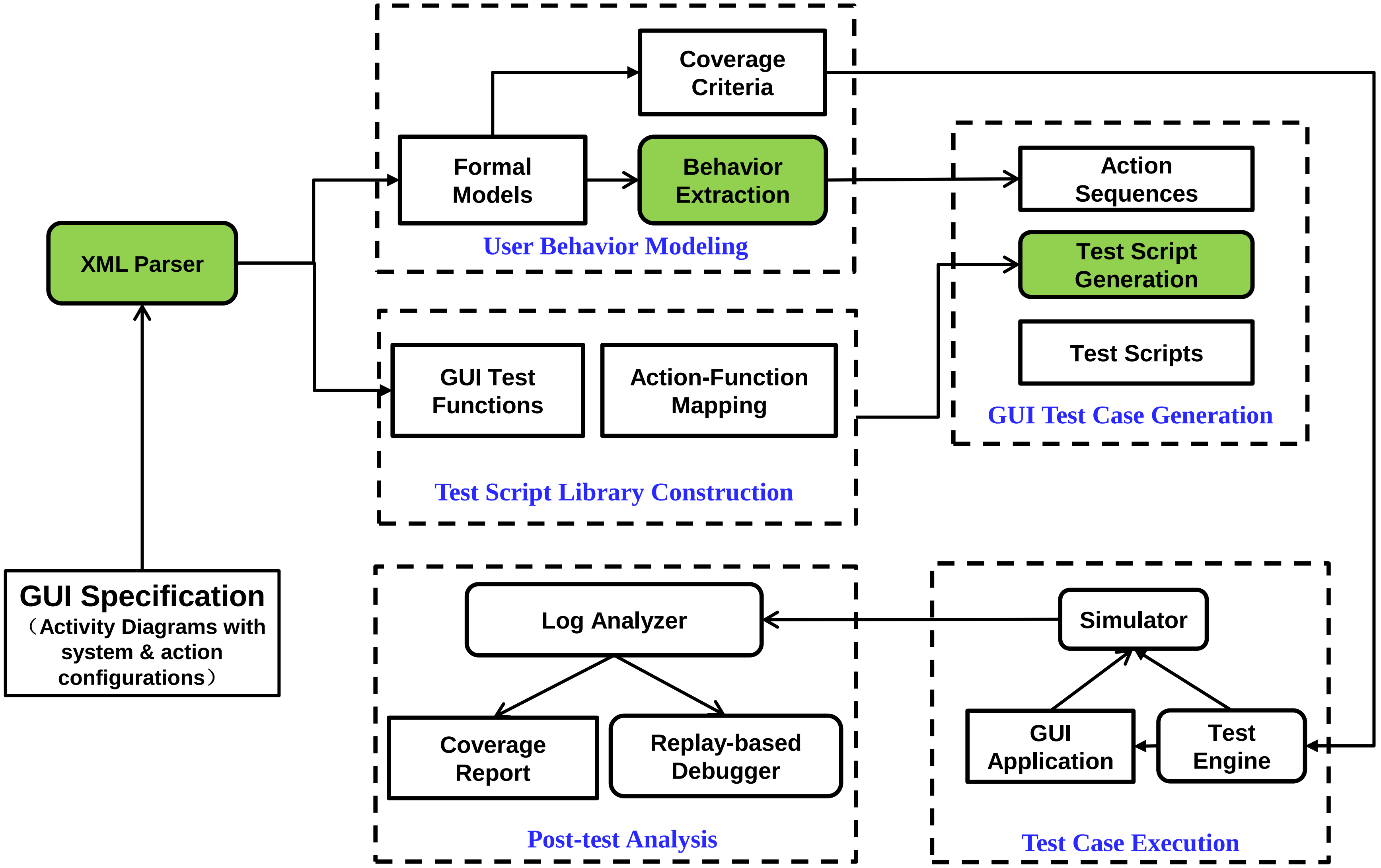
Total number of both iOS and Android APP Downloads : **>100 billion**

- ❑ More and more people rely on smartphone apps to manage their daily life (bills, shopping, surfing, emails, etc.).
- ❑ GUI testing is becoming a major bottleneck in overall testing (up to 40% of time and resources).
- ❑ Time-to-market pressure requires automated GUI testing techniques.

Summary of Related Work

- ❑ Capture & Replay / Script Driven Testing
  - ✓ Flexible for debug purpose (tools: Robotium, UIAutomation, etc.)
  - ✗ Require expert-knowledge to detect errors, need human intervention
- ❑ Random Testing
  - ✓ Can be fully automated (tools: Monkey, UI-Auto-Monkey, etc.)
  - ✗ Coverage convergence cannot be guaranteed. The coverage may take quite long time to go from 90% to 99%.
- ❑ Model-Based Approaches (e.g., FSM, Event-Flow)
  - ✓ Can be automated with less test cases.
  - ✗ Focus on the internal logic rather than user behaviors
- ❑ Few of existing testing approaches make use of user behavior information.
  - ◆ How to model user behaviors accurately?
  - ◆ How to check the inconsistency between GUI specifications and GUI implementations?

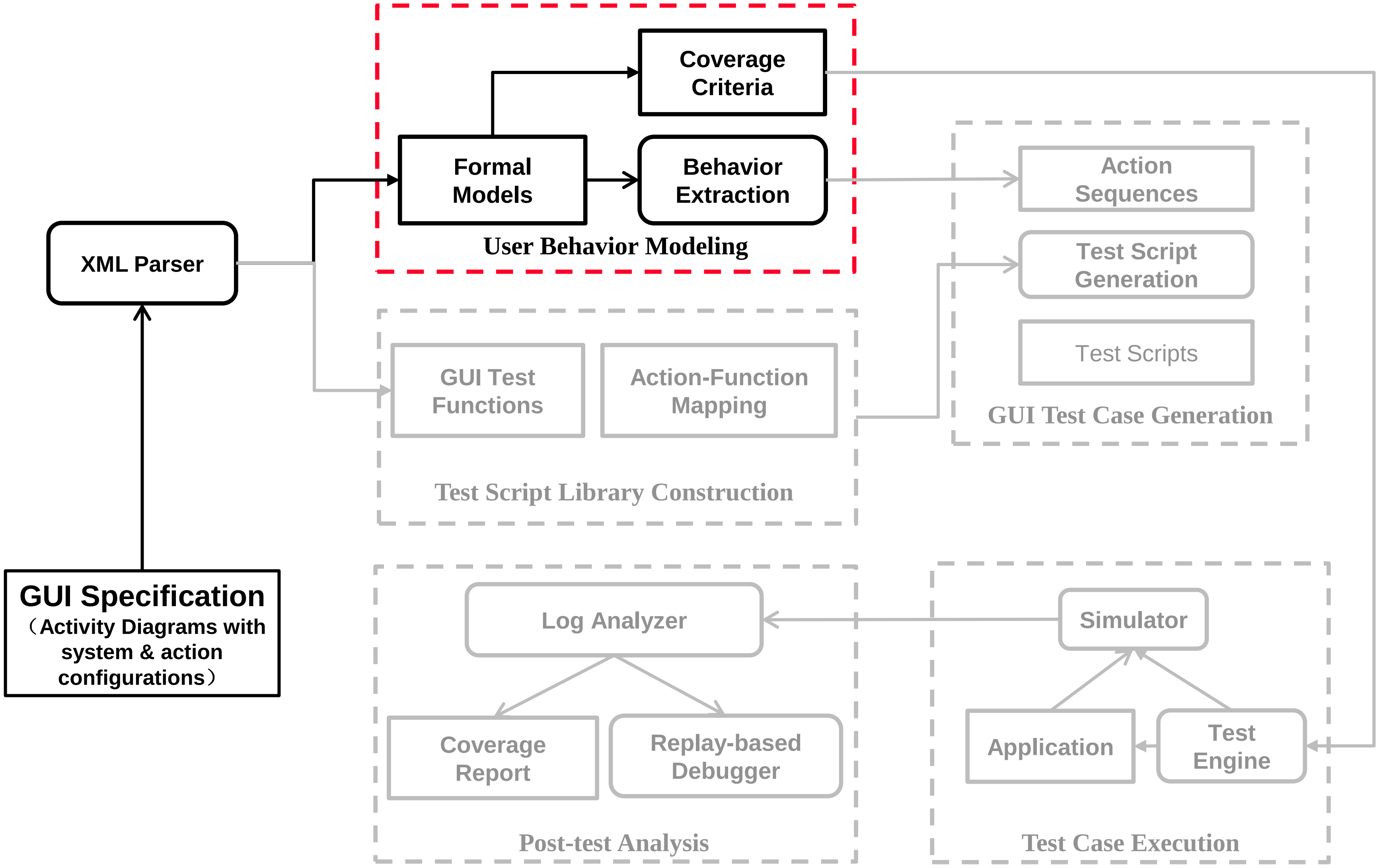
Overview of Our ADAutomation Framework



Outline

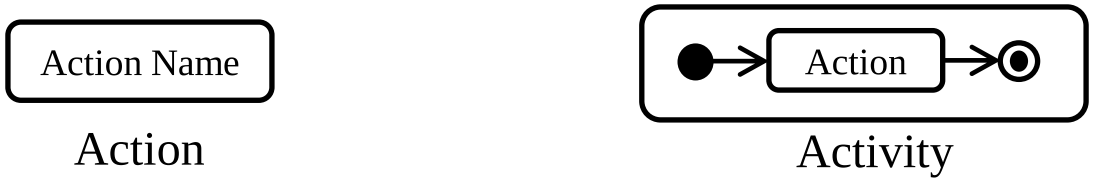
- Introduction
- Behavior Modeling Using Activity Diagrams
  - Graph–based Notations
  - User Behavior Modeling & Extraction
- Our Automated GUI Testing Framework
  - Test Script Library Generation
  - GUI Testing & Error Diagnosis
- Experiments
- Conclusion

User Behavior Modeling

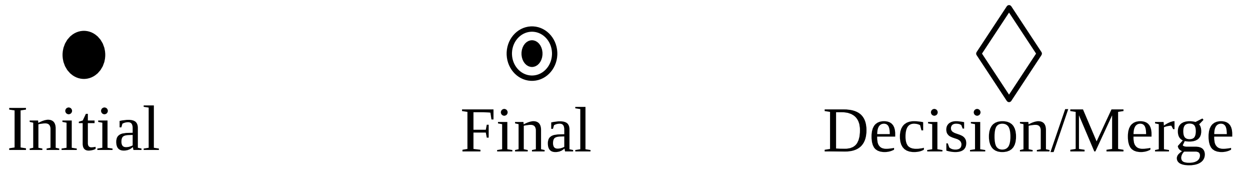


Graph-Based Notations

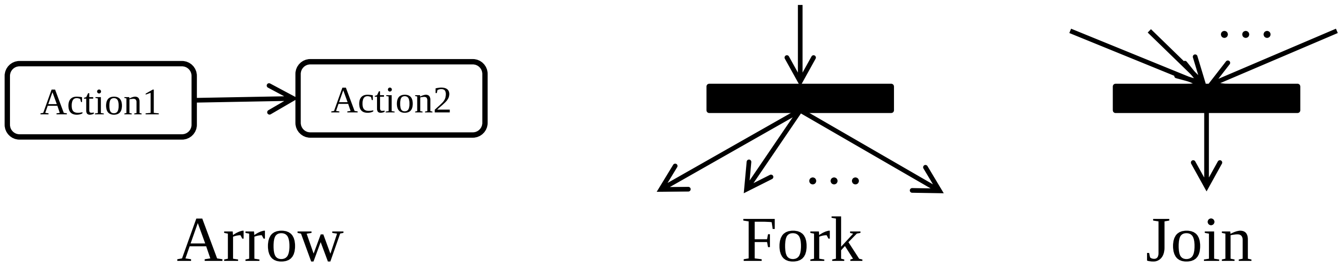
- UML Activity Diagram Notations
  - *Actions* (i.e., external operations) and *activities (i.e., GUI views, which* indicate a collection of correlated actions)



  - Control nodes and flows (indicating the execution order of actions)
    - Control nodes



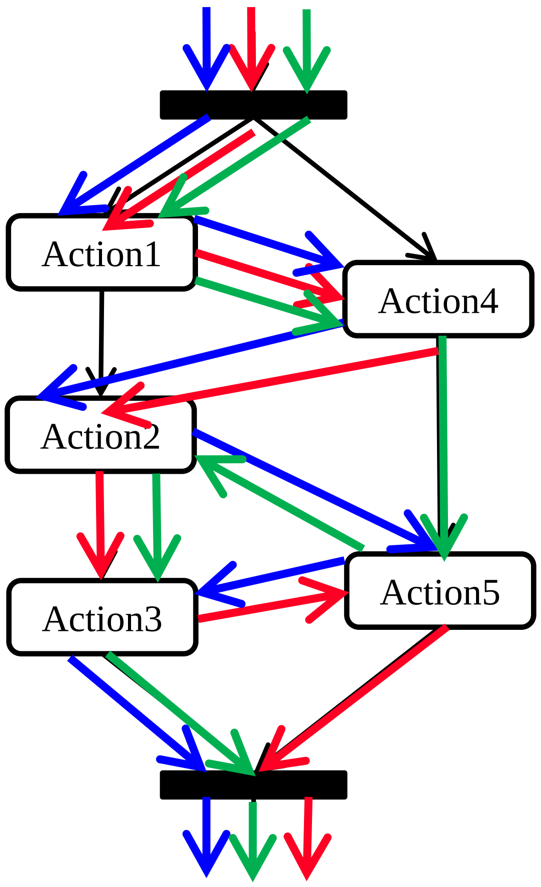
    - Control flow

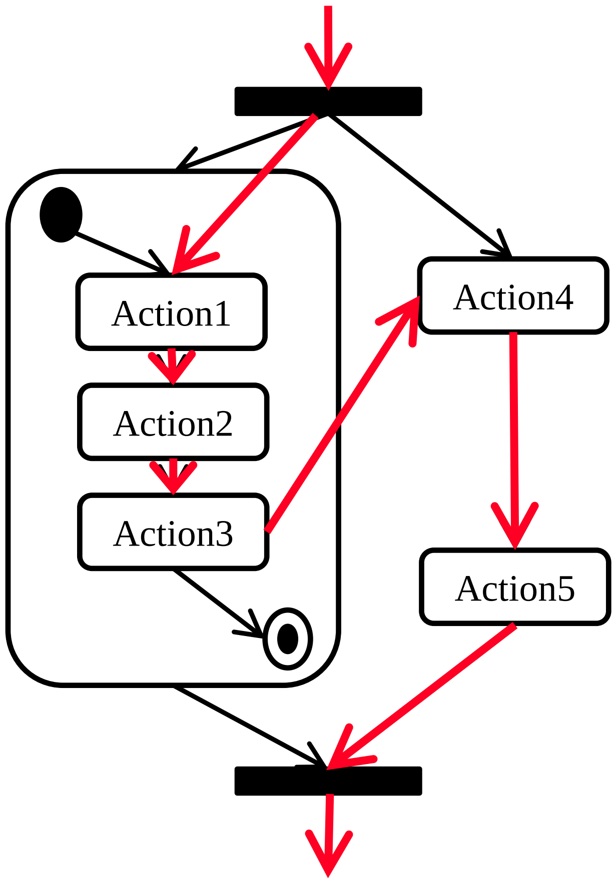


User Behavior Modeling

- As a semi-formal specification, UML activity diagrams cannot be automatically analyzed.
- We extend the relation between actions with a quasi-Petri-net semantics for GUI testing purpose.
  - Concurrent action execution → Interleaving of actions
  - Activities → GUI views
    - Actions in the current view cannot be preempted by actions from other views.
  - Activity depth → Action execution priority
    - Nested activities are indexed by their depth
    - At any time, only actions in the deepest activity can be fired.
- The behavior of an activity diagram can be represented by a sequence of actions (e.g., <Initial, a, b, c, …, Final>).

Fork/Join Operations & Activity Hierarchy





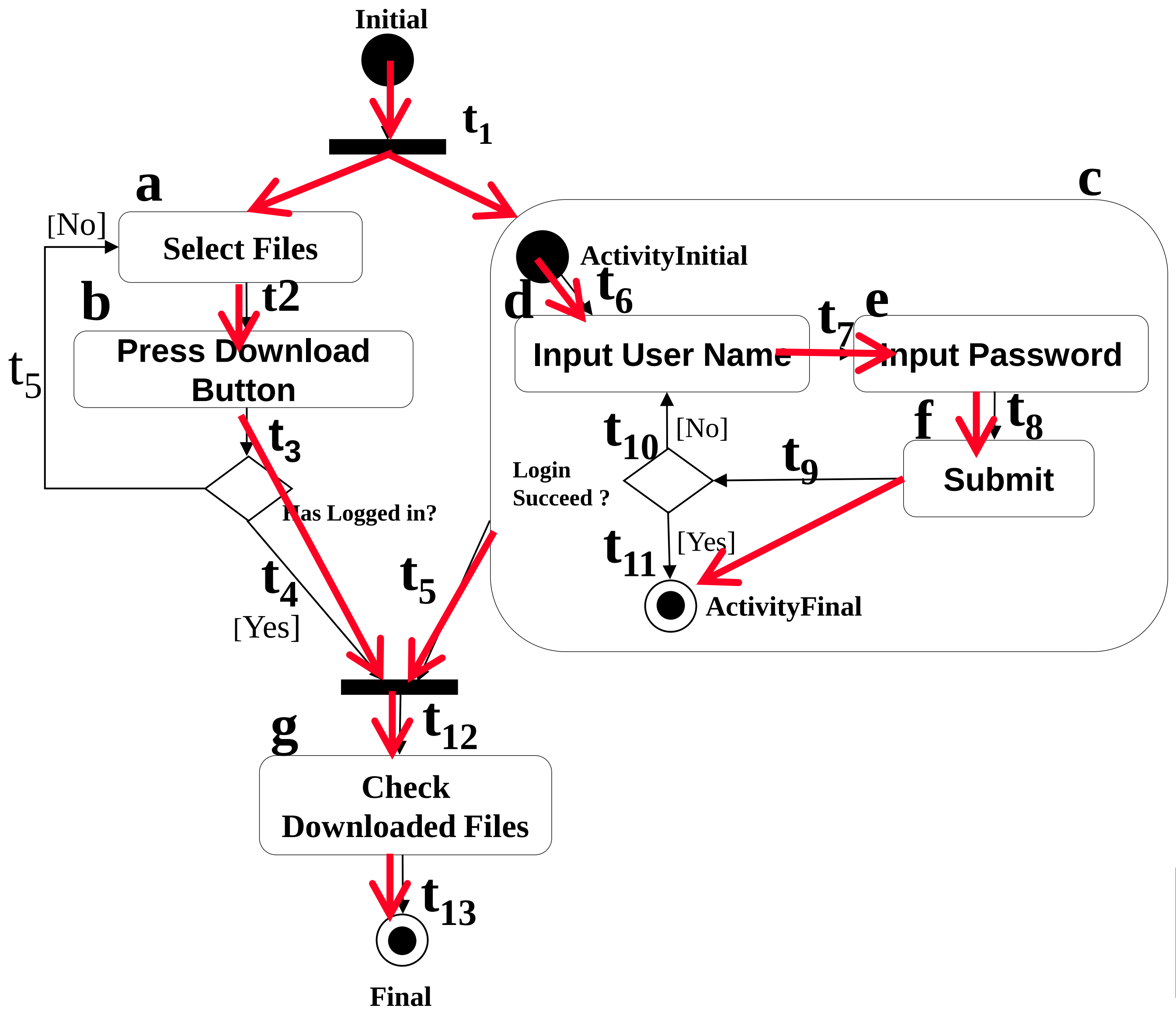
The action order of each thread should be kept. Independent actions can be interleaved.

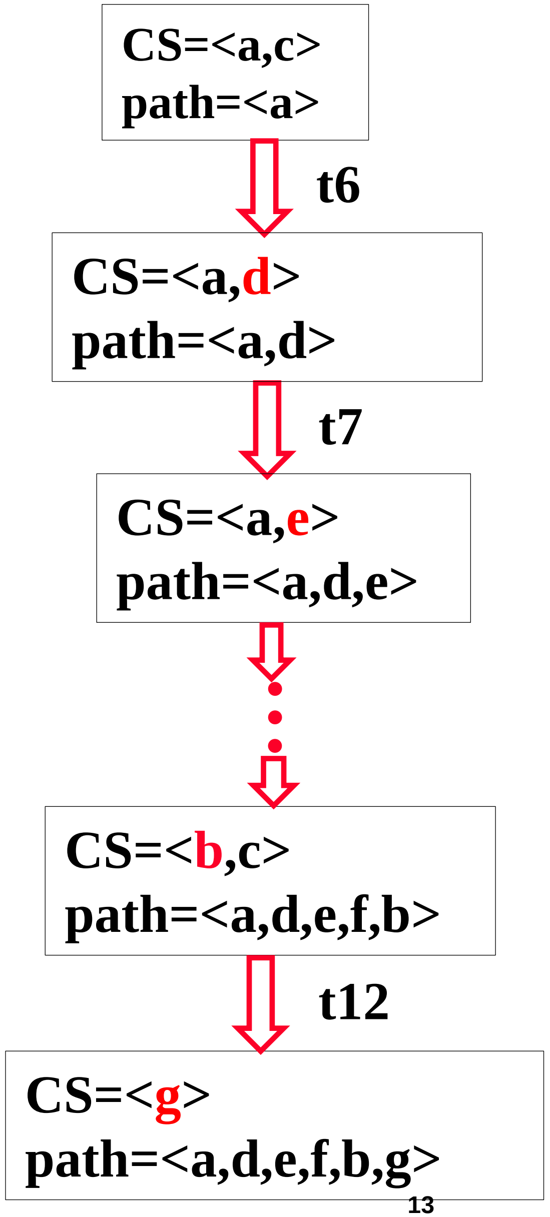
The execution of action1, action2, and action3 cannot be interrupted by the action4 and action5.

User Behavior Extraction

- To enable sufficient testing, we need to enumerate all possible user behaviors from activity diagrams, which can be explored in a depth-first way
- However, due to the loops, it is impossible to enumerate all the user behaviors
  - Restrict the length of path with a limited bound
- To mimic the user behaviors, the depth-first user behavior exploration needs to consider the hierarchy information of activities.
  - Consider the depth information of enclosed activity for each action
  - Only the actions within the deepest activity can be explored further

Forward Path Exploration





CS=<a,c>
path=<a>

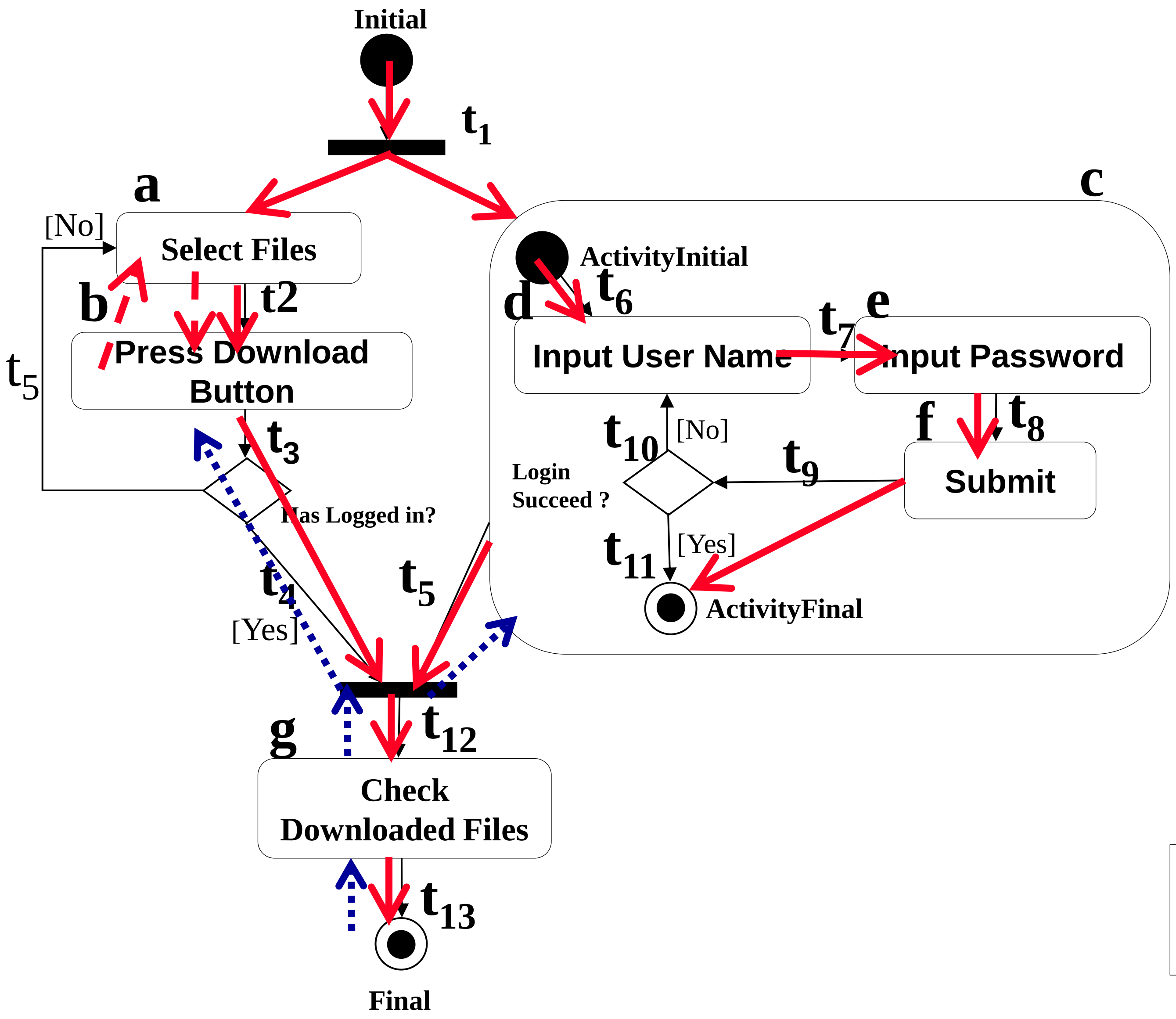
CS=<a,d>
path=<a,d>

CS=<a,e>
path=<a,d,e>

CS=<b,c>
path=<a,d,e,f,b>

CS=<g>
path=<a,d,e,f,b,g>

Backward Path Exploration



CS=<Final>
path=<a,d,e,f,b,g>

⇩ t13

CS=<g>
path=<a,d,e,f,b,g>

⇩ t12

CS=<b,c>
path=<a,d,e,f,b>

⇩ t5

CS=<a,c>
path=<a,d,e,f,b,a>

⇩ t2

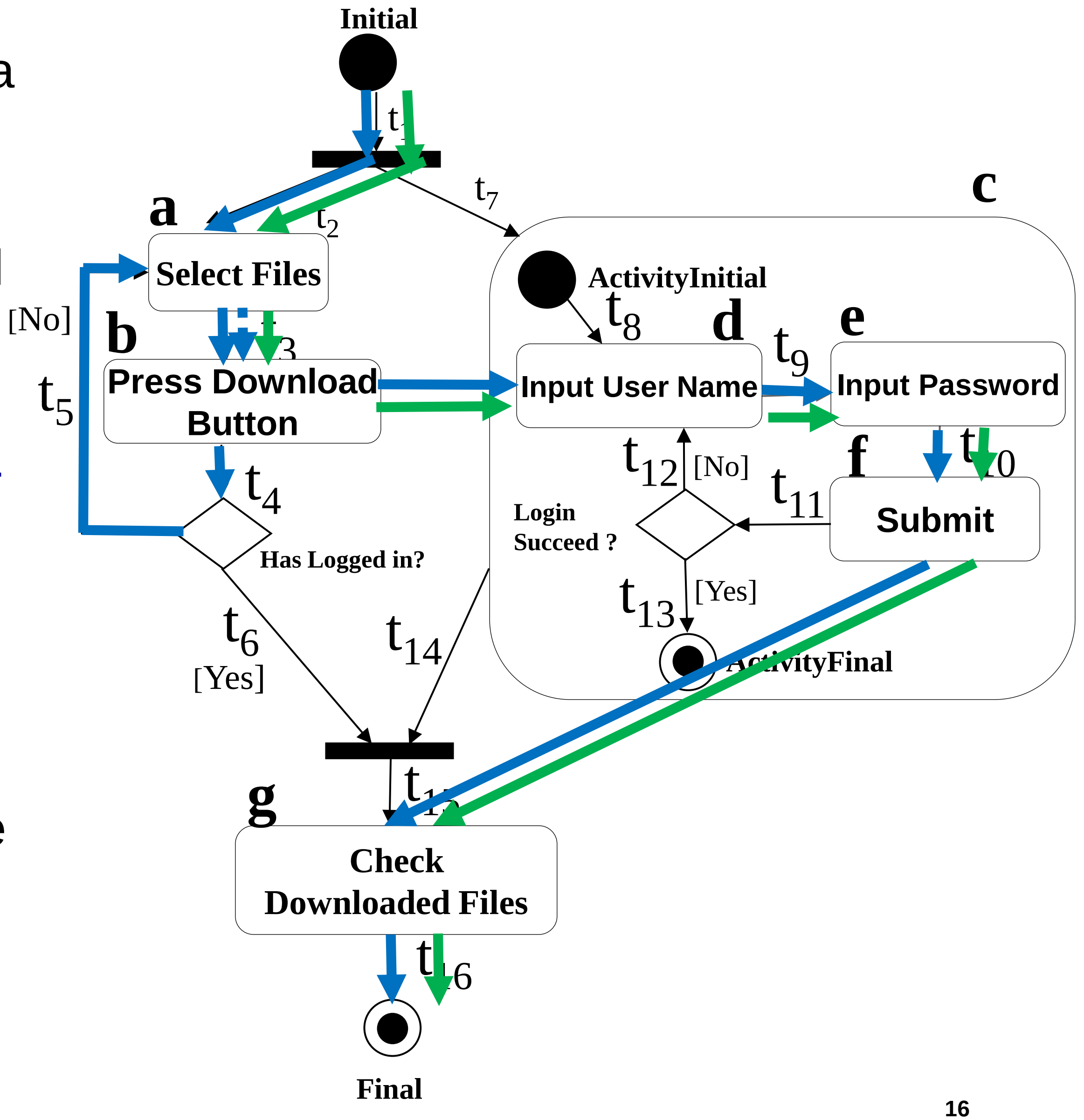
CS=<b,c>
path=<a,d,e,f,b,a,b>

User Behavior Coverage Criteria

- To measure the effectiveness of user behaviors extracted from activity diagrams, we propose three types of coverage metrics.
  - Action coverage check the reachability of each action
  - Transition coverage checks all the conditional branch along the control flows.
  - Simple path checks possible combination of actions and transitions.
    - A simple path is a path (user behavior) without action repetition.
    - When a user behavior remove all its repeated actions, it should match some simple path.

An Example of Simple Path Coverage

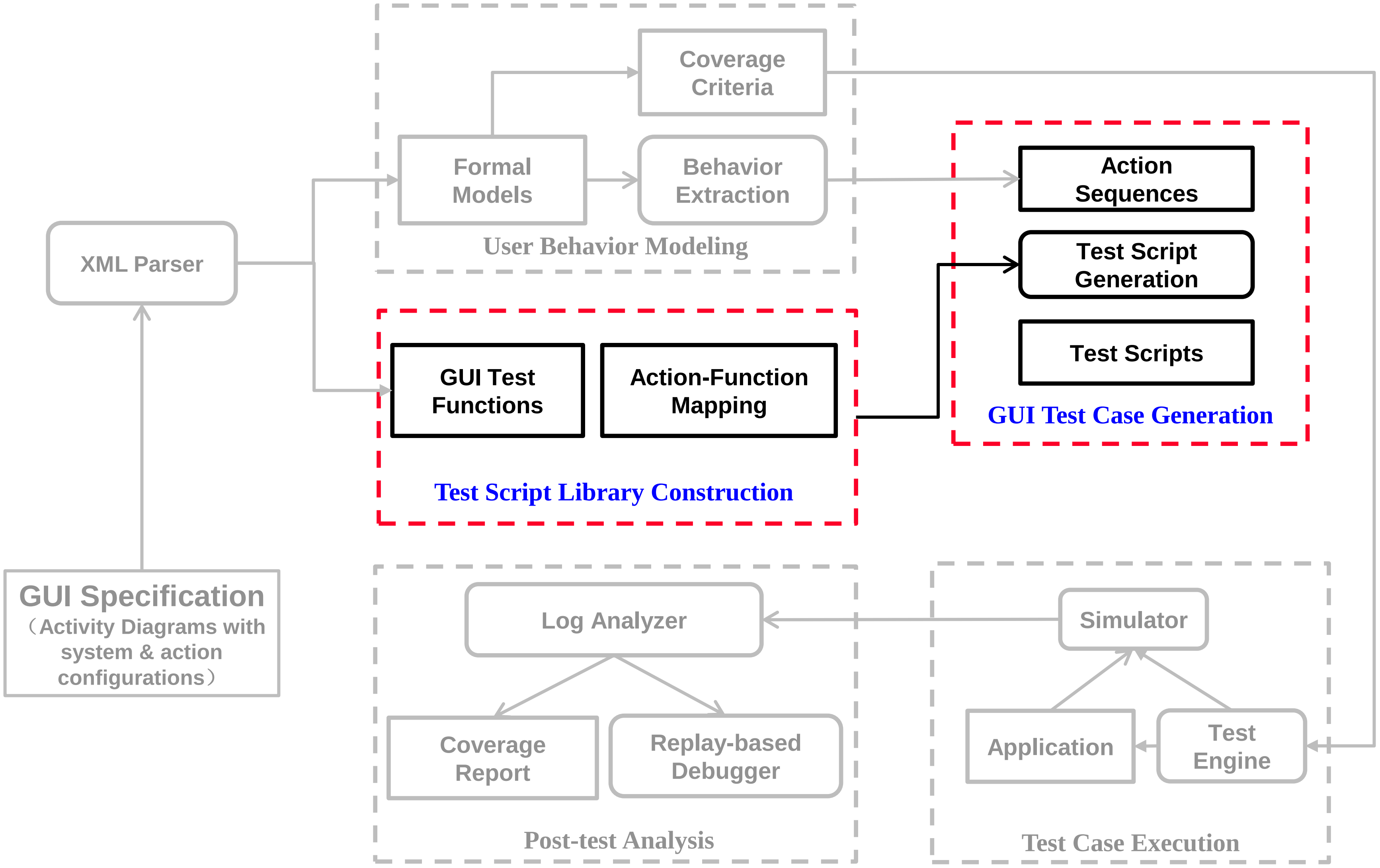
- A user behavior matches a simple path if it can simulate on the activity diagram and it contains all the actions of the simple paths.
  - E.g., the user behavior described in blue lines matches the simple path in green lines.
- Simple path coverage requires that all the simple paths to be covered.



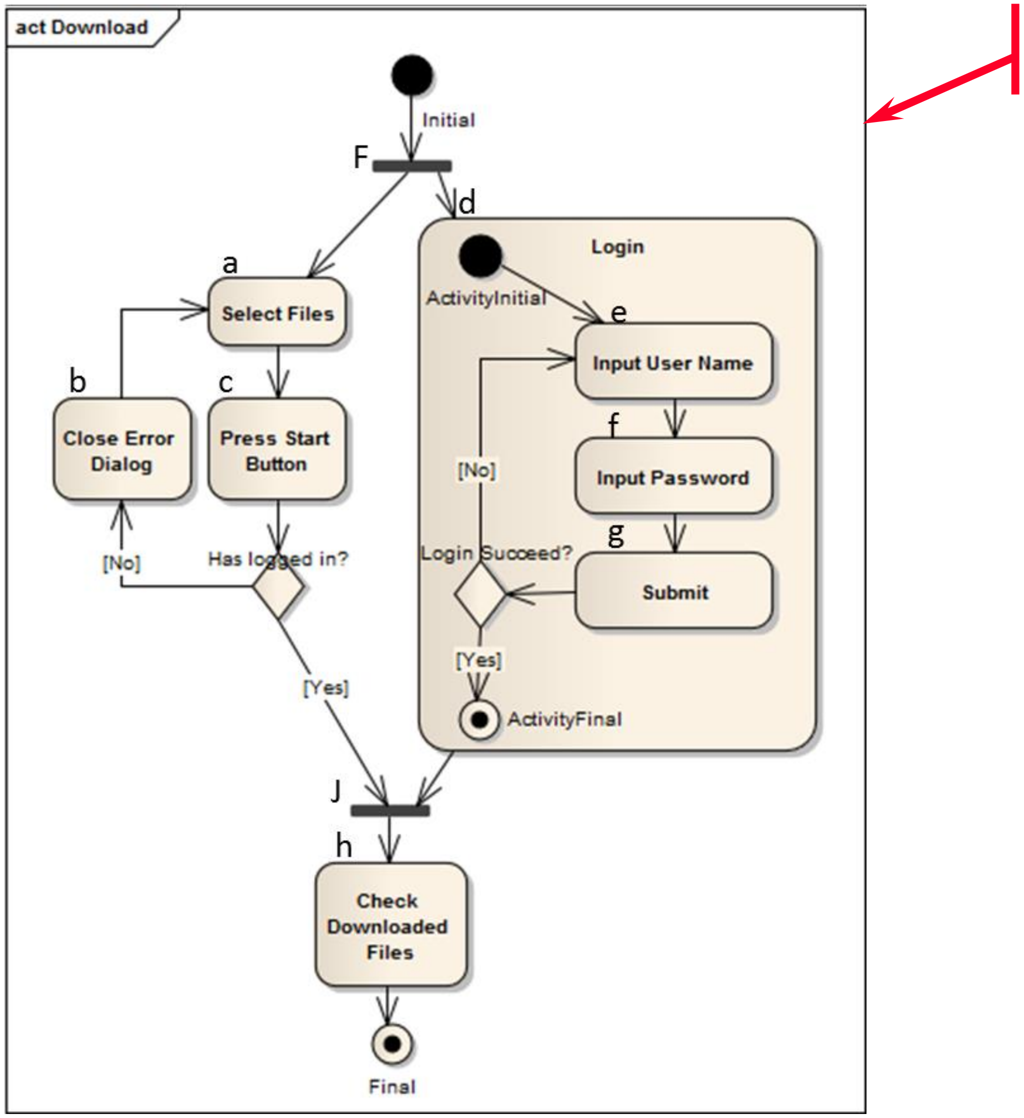
Outline

- Introduction
- Behavior Modeling Using Activity Diagrams
  - Graph–based Notations
  - User Behavior Modeling & Extraction
- Our Automated GUI Testing Framework
  - Test Script Library Generation
  - GUI Testing and Error Diagnosis
- Experiments
- Conclusion

Test Script Library Construction



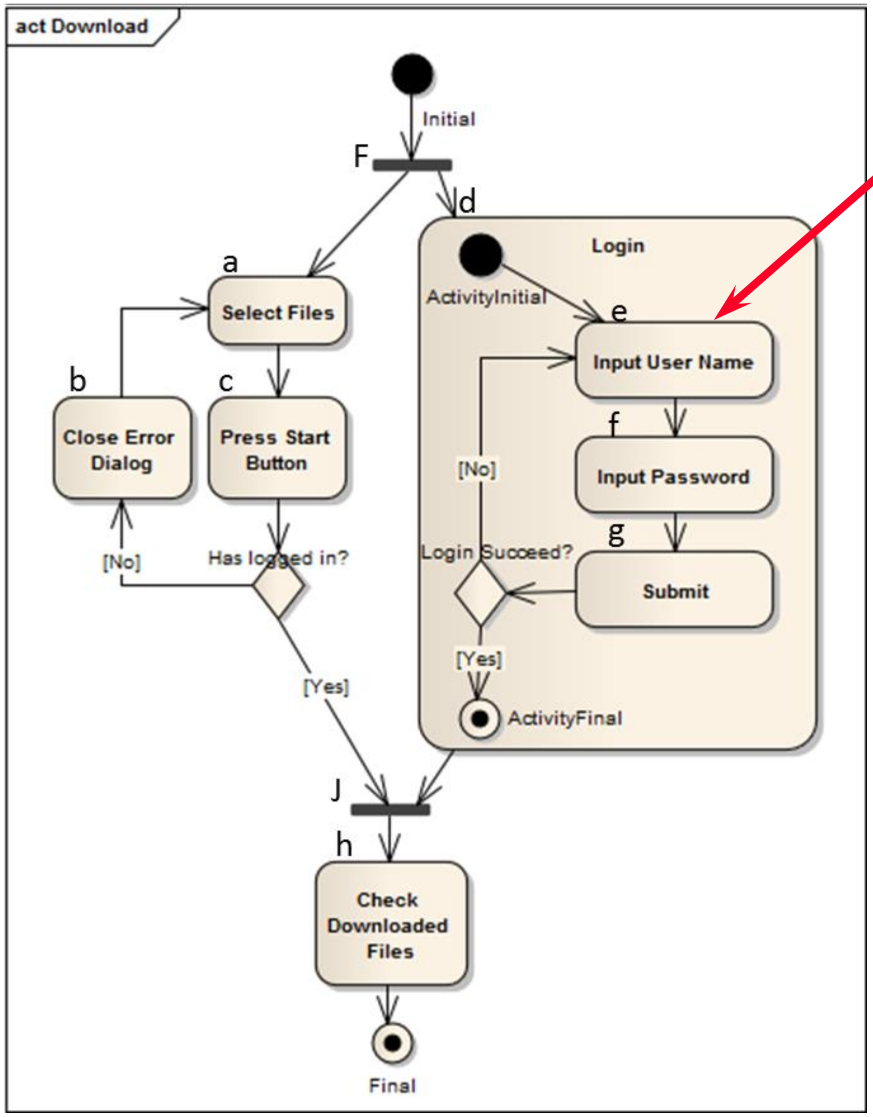
Activity Diagram Annotation



System configuration

```
<Platform>
    <Name>iOS</Name>
    <Version>6.0</Version>
    <TestEngine>
        <Name>…</Name>
        <Version>…</Version>
    </TestEngine>
    <Delay>
        <Unit>Second</Unit>
        <Value>0.2</Value>
    </Delay>
</Platform>
```

Activity Diagram Annotation



Action configuration

- Widget features specify the attributes of corresponding GUI widgets such as ID, name, position, size, etc.
- GUI operations describe user operations conducted on associated GUI widgets.
- Test logs instrument proper log information based on the result of GUI operations.

Action-Function Mapping

- For each action, our tool will generate one corresponding function. Therefore, there will be a mapping between the action and its associate test function.
- For the name of derived test function, we use the following convention
  - Action node information
    - Name=“Select Files”
    - ID=1
  - Function name
    - _1_SelectFiles

Test Script Library

- A Test script library is a set of system settings and generated test functions
  - System configuration → System settings
  - Action configuration → Test Functions
- A skeleton of a test script library library.js in JavaScript

```
var delay = 0.2;    // System setting
function screenshot( ){ ……} // System Functions
function _1_SelectFiles( ){……}
function _2_PressDownloadButton ( ){……}
function _3_InputUserName( ){
    UIALogger.logMessage("Action:InputUserName");
    win.textFields()[0].setValue("…");
    screenshot( );
}
function _4_InputPasswd( ){……}
function _5_Submit( ){……}
function _6_CheckDownloadFiles( ){……}
……
```

// Test functions

Test Case Generation

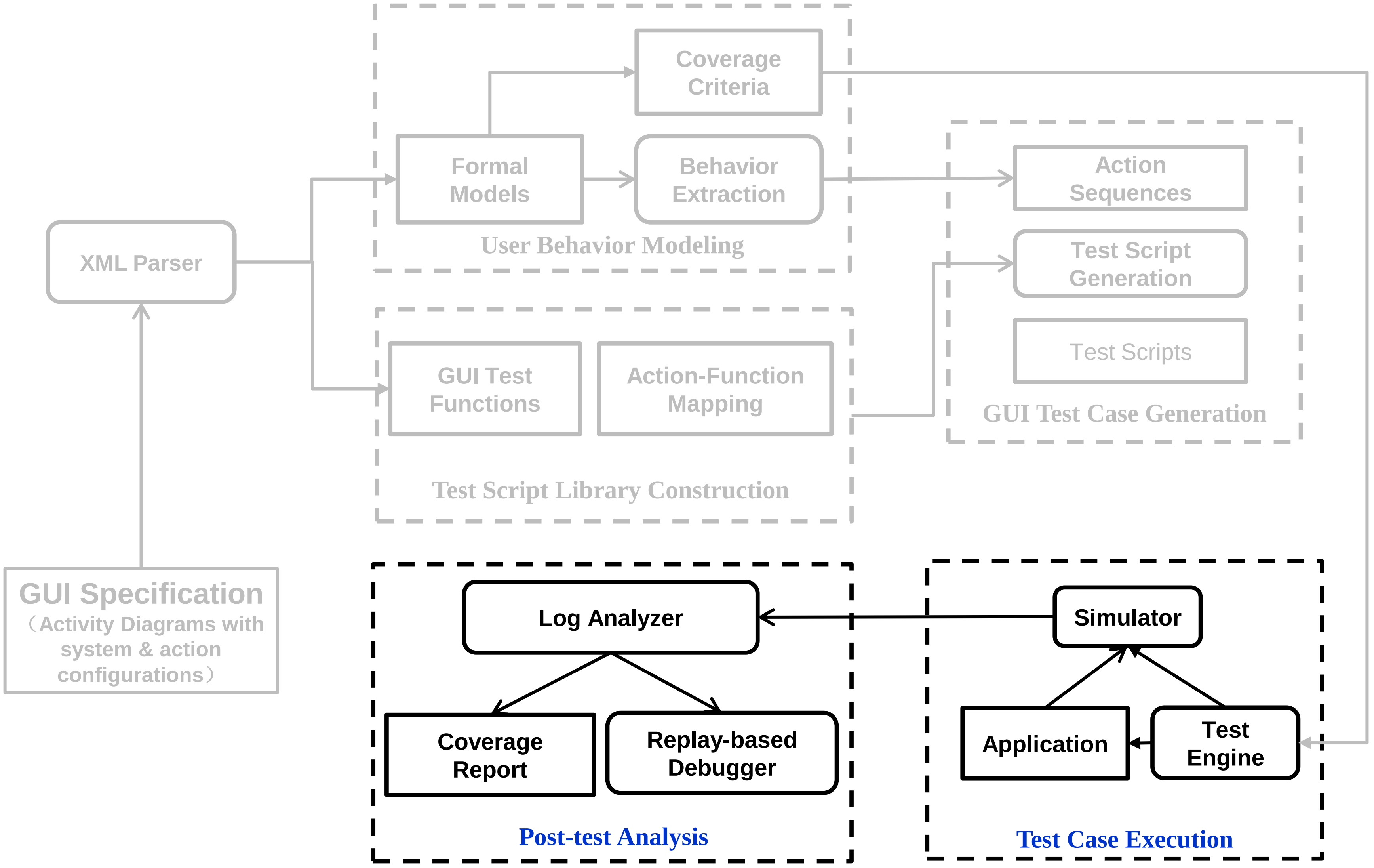
- The process of test case generation is a one-one mapping from a user behavior to a test case
  - An extracted user behavior example

    <Initial, "Select Files", "Input User Name", "Input Password", "Submit", "Press Download Button", "Check Download Files", Final >

  - The corresponding generated test case

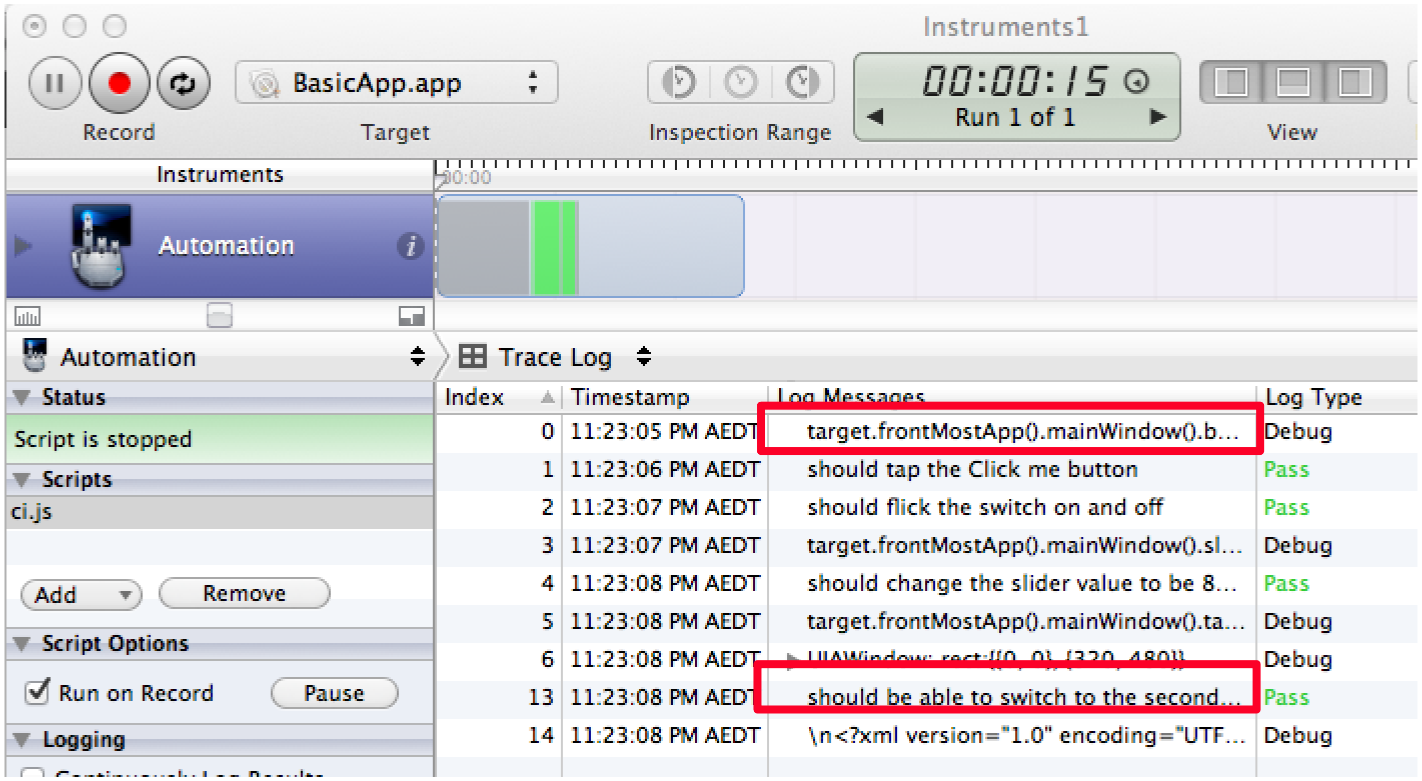
```
#import "Library.js"
UIALogger.logStart("Testing starts");
_1_SelectFiles( );
_3_InputUserName();
_4_InputPasswd();
_5_Submit( );
_2_PressDownloadButton ( );
_6_CheckDownloadFiles( );
UIALogger.logPass("Testing ends");
```

Overview of ADAutomation Framework



Error Diagnosis

- Error Diagnosis
  - Log information

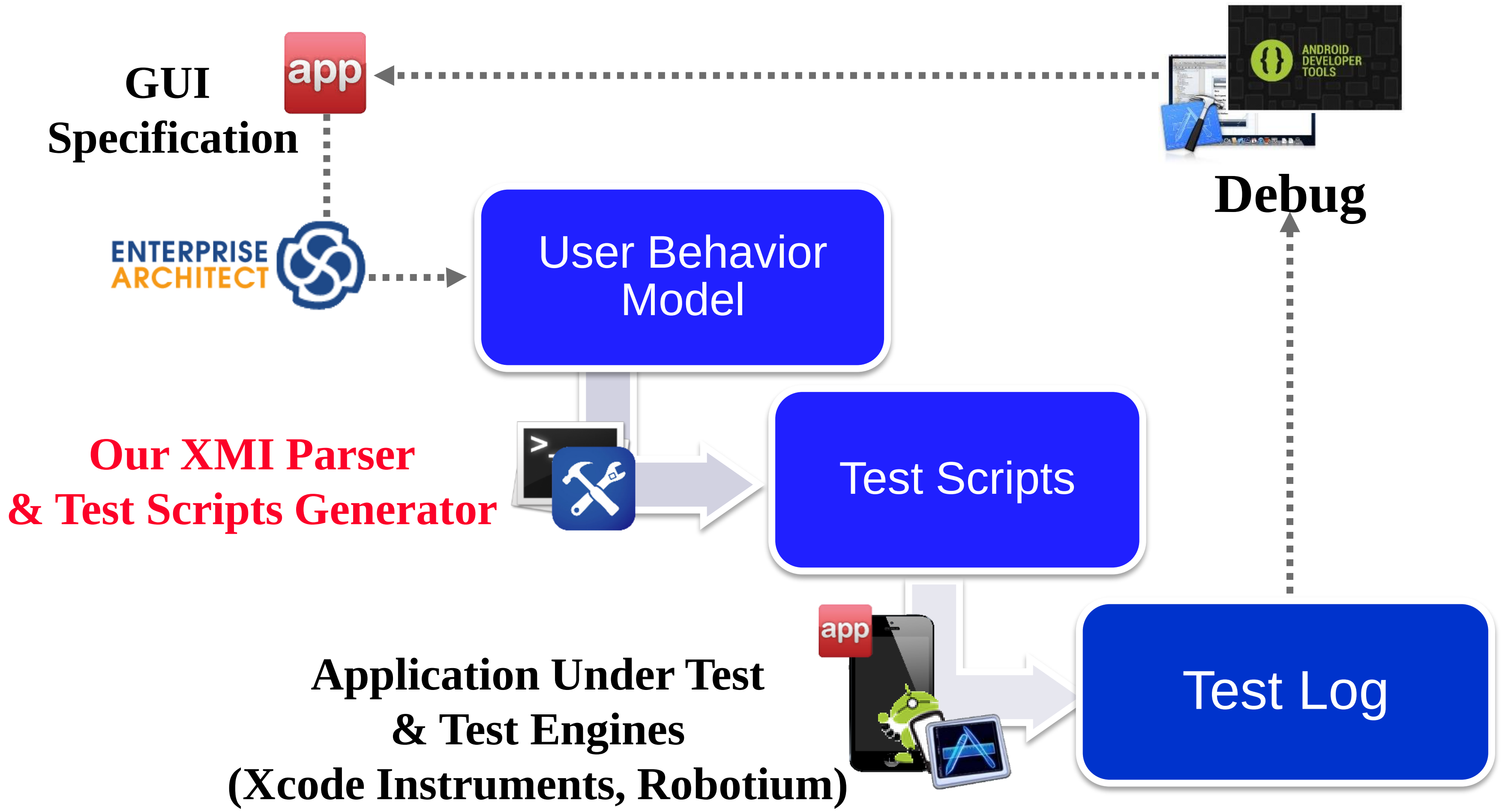


Index	Timestamp	Log Messages	Log Type
0	11:23:05 PM AEDT	target.frontMostApp().mainWindow().b…	Debug
1	11:23:06 PM AEDT	should tap the Click me button	Pass
2	11:23:07 PM AEDT	should flick the switch on and off	Pass
3	11:23:07 PM AEDT	target.frontMostApp().mainWindow().sl…	Debug
4	11:23:08 PM AEDT	should change the slider value to be 8…	Pass
5	11:23:08 PM AEDT	target.frontMostApp().mainWindow().ta…	Debug
6	11:23:08 PM AEDT	UIAWindow: rect:{{0, 0}, {320, 480}}	Debug
13	11:23:08 PM AEDT	should be able to switch to the second…	Pass
14	11:23:08 PM AEDT	\n<?xml version="1.0" encoding="UTF…	Debug

  - Playback

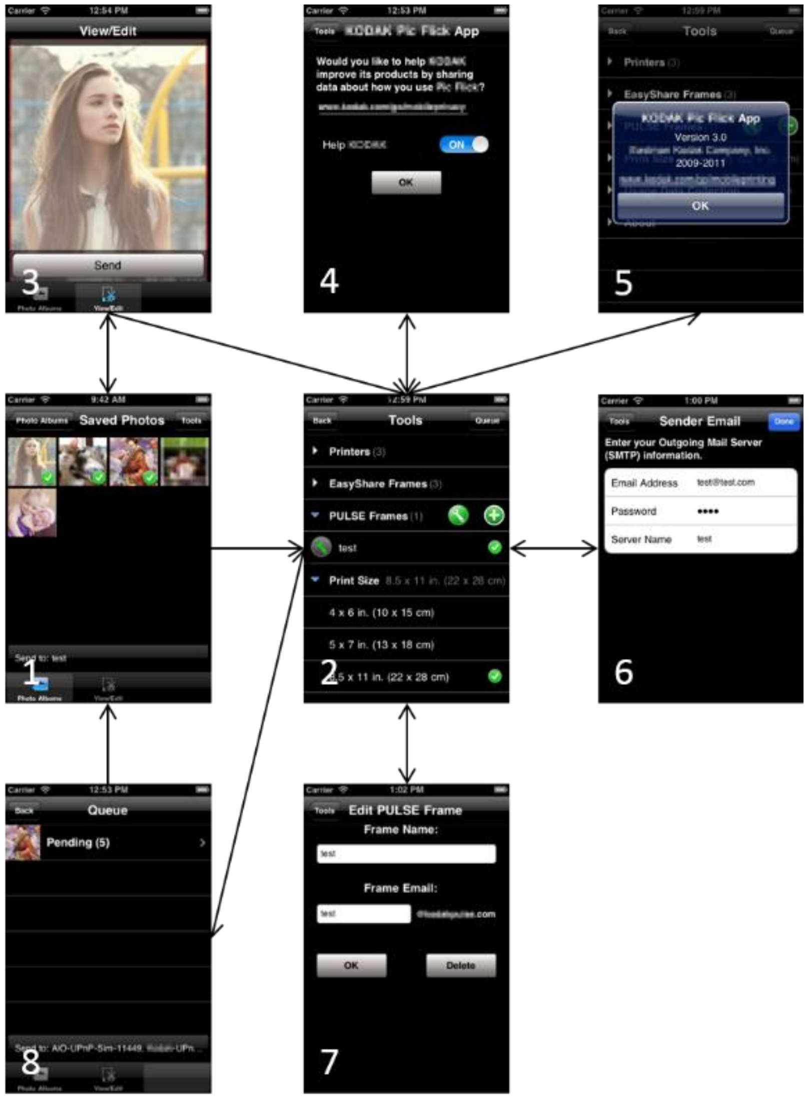
Outline

Tools Chain for Experiment



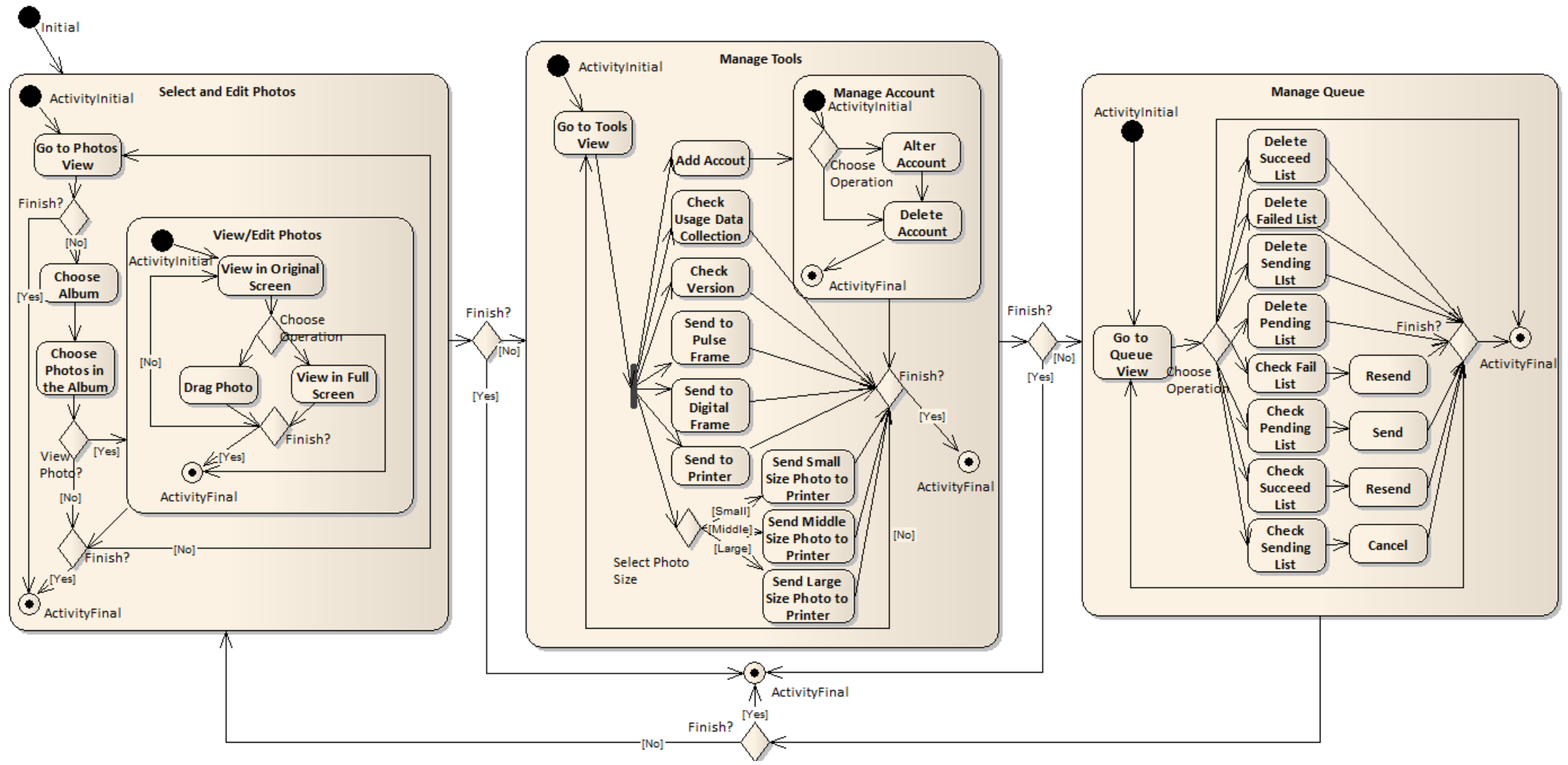
- ❑ All the experimental results were obtained on a MacBook Pro machine with Intel Core i5 2.4GHz processor and 4 GB RAM.

Case Study 1: *PicFlic*



- *PicFlic* is a free Wi-Fi based remote picture print management application developed by Eastman Kodak company.
- The left picture shows the eight views of *PicFlick* on iOS and the corresponding view switches indicated by arrow lines .
- We did GUI testing on both iOS and Android versions of *PicFlick*.

The Activity Diagram of *PicFlick*



- Designing this activity diagram from user specification needs around 6 hours.
- 30 actions, 86 transitions, and 955 simple paths are derived from this activity diagram.
- It can generate 12776 test cases with a bound limit 18.

Test Results of *PicFlick*

- Testing Time (test case generation time + simulation time):
  - 70217 seconds (less than 20 hours) for PicFlick (iOS).
  - Compared to 2-3 man month manual testing in industrial.

Bound Size	Test Case #	Failed Case #	Test Time (s)	Action Coverage	Transition Coverage	Sim. Path Coverage
2	2	0	16.06	13.33%	13.95%	0.21%
4	13	0	103.37	43.33%	46.51%	1.05%
6	59	0	438.57	66.67%	67.44%	2.62%
8	176	5	1207.15	83.33%	82.56%	5.03%
10	432	14	2773.44	100%	100%	8.07%
12	881	22	5231.80	100%	100%	19.71%
14	2224	83	13235.72	100%	100%	45.60%
16	5784	331	33481.70	100%	100%	81.66%
18	12776	993	70217.14	100%	100%	98.95%

- Android version shows the similar testing results.

Bugs Found in *PicFlick*

- 993 of 12776 test cases resulted in application crashes on iOS, and 5 suspected bugs were found.

Index	Error Scenarios	Failed #	Reasons of the failures
1	If the picture is too large, then the drag of the picture may crash.	121	Due to the limited resource for the smartphone application, the drag of big pictures will use up all the allocated CPU and memory resources.
2	If users send pictures to digital frames and printers at the same time, the application will crash.	806	The implementation of the task scheduling between sending list and pending list is wrong.
3	Fail to delete tasks from pending list.	40	The implementation of the delete operation of the pending list is wrong.
4	Fail to tap the sending list button in the Queue view.	12	After selecting devices to send photos, the sending list button is disabled by mistake.
5	Fail to find printers which appear in the Tools view.	14	The implementation of the connection between PicFlick and the drivers of printers is wrong.

- 925 of 12776 test cases resulted in application crashes on Android, but only bug 1 and 2 were reported.

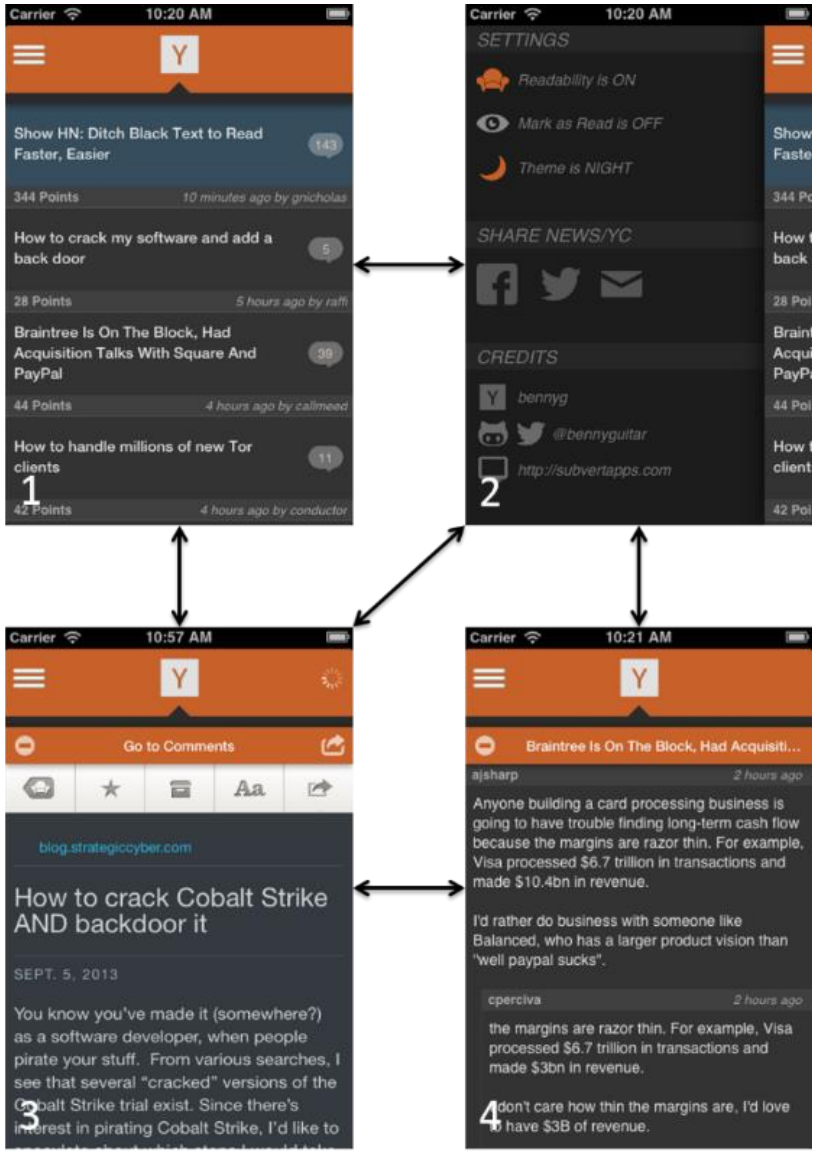
Random Testing for *PicFlick*

- We conduct the random testing using UI AutoMonkey on iOS

Bound Size	Action Coverage	Transition Coverage	Simple Path Coverage	Bugs Found
20	47.22%	59.34%	0.93%	0
50	97.22%	97.80%	65.60%	1
100	97.22%	97.80%	85.80%	2
200	97.22%	97.80%	85.80%	2
400	100%	100%	93.89%	3
unlimited*	100%	100%	93.89%	3

- Random testing took 24 hours, but only 3 out of 5 known bugs were found. No new errors were detected using the random approach.
- Simple path coverage is 93.89% using the random approach compared to 98.95% using ADAutomation.
- Random testing on Android shows the similar results.

Case Study 2: *Newsyc*



Show HN: Ditch Black Text to Read Faster, Easier 143
344 Points 10 minutes ago by gnicholas
How to crack my software and add a back door 5
28 Points 5 hours ago by raffi
Braintree Is On The Block, Had Acquisition Talks With Square And PayPal 39
44 Points 4 hours ago by callmeed
How to handle millions of new Tor clients 11
42 Points 4 hours ago by conductor

SETTINGS
Readability is ON
Mark as Read is OFF
Theme is NIGHT
SHARE NEWS/YC
CREDITS
bennyg
@bennyguitar
http://subvertapps.com

Go to Comments
blog.strategiccyber.com
How to crack Cobalt Strike AND backdoor it
SEPT. 5, 2013
You know you've made it (somewhere?) as a software developer, when people pirate your stuff. From various searches, I see that several "cracked" versions of the Cobalt Strike trial exist. Since there's interest in pirating Cobalt Strike, I'd like to

Braintree Is On The Block, Had Acquisiti...
ajsharp 2 hours ago
Anyone building a card processing business is going to have trouble finding long-term cash flow because the margins are razor thin. For example, Visa processed $6.7 trillion in transactions and made $10.4bn in revenue.

I'd rather do business with someone like Balanced, who has a larger product vision than "well paypal sucks".
cperciva 2 hours ago
the margins are razor thin. For example, Visa processed $6.7 trillion in transactions and made $3bn in revenue.

I don't care how thin the margins are, I'd love to have $3B of revenue.

- Newsyc is an open source *Hacker News* client.
- The GUI implementation has four views (i.e., news list view, news browsing view, comments view, and system setting view).
- Its activity diagram has12 actions, 35 transitions, and 5 simple paths
- 4995 test cases were generated form this diagram with a bound limit of 15

Testing Result for *Newsyc* (iOS)

Bound Size	Test Case #	Test Time (s)	Action Coverage	Transition Coverage	Sim. Path Coverage
2	1	8.48	25.00%	29.41%	6.67%
3	3	26.11	33.33%	35.29%	20.00%
4	7	62.45	33.33%	47.06%	20.00%
5	11	99.66	33.33%	47.06%	20.00%
……					
15	4995	47145.0	33.33%	47.06%	20.00%

- One Bug Found - When users enter the news browsing view for the first time, they cannot bookmark the news, share the news, or modify the font size.
- Test time (test case generation time + simulation time) is 47145 seconds (about 13 hours).

Random Testing for *Newsyc*

Bound Size	Action Coverage	Transition Coverage	Simple Path Coverage	Bugs Found
10	100.00%	97.06%	53.33%	0
20	100.00%	100.00%	80.00%	0
50	100.00%	100.00%	100.00%	0
100	100.00%	100.00%	100.00%	0
200	100.00%	100.00%	100.00%	0
unlimited*	100.00%	100.00%	100.00 %	0

- Random testing takes12 hours.
- Achieved 100% coverage in all categories.
- No bug was found.

Conclusion

- GUI testing is becoming a major bottleneck in smartphone application development.
- This paper proposes an efficient automated GUI testing framework for smartphone applications
  - User behavior modeling using activity diagram
  - Three test adequacy criteria
  - Automated GUI test library construction
  - Tool chain for automated GUI testing
- Successfully applied on various smartphone applications
  - Significant reduction in overall GUI testing time
  - More bugs found than random methods

Thank you !