

AppFlow: Using Machine Learning to Synthesize Robust, Reusable UI Tests

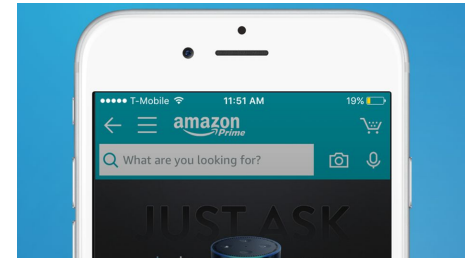
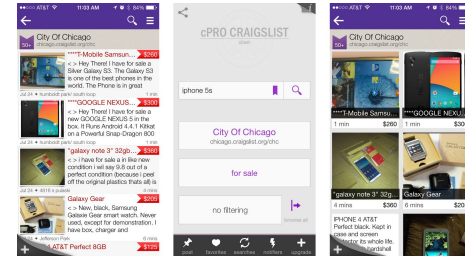
Authors: Gang Hu, Linjie Zhu, Junfeng Yang
Dept. of Computer Science, Columbia University

Presentation by: Noah Gallant
for COMS W6998 (Formal Logic, Security and ML)



Background

- UI Testing is very challenging
 - Largely relies on scripting
 - High initial development cost
 - “Testing bugs”
- “application UIs are designed for human intelligence but test scripts are low level click-by-click scripts”
- Test re-use is difficult
 - Despite flow similarity-- designs are always different so recognition is difficult
 - Even similar apps in the same category have different flows



Introducing AppFlow

- “AppFlow provides ‘smoke tests’ or build verification testing for each source code change, requiring little or no manual work”
- Learns a classifier from a training dataset of screens and widgets labeled with their intents
 - texts, widget sizes, image recognition results of graphical icons, optical character recognition (OCR)
- Training dataset comes from a developer community for an app category
 - AppFlow provides utilities to simplify data collection
 - Map variant screens and widgets to canonical ones.
 - “Your Email” or “example@email.com” on sign-in screens to **signin.username**



AppFlow research at a higher level

- Initial Benefits
 - UI can be updated without rewriting unit tests
 - Multiple screen sizes supported without re-writing tests
- Flow-based testing
 - Pre-condition
 - Post-condition
 - User steps
- Android-based
- Tested on widely used apps



An example test

```
Scenario: add to shopping cart [stay at cart]
  Given screen is detail
  And cart_filled is false
  When click @addtocart
  And click @cart
  And not see @empty_cart_msg
  Then screen is cart
  And set cart_filled to true
```

Figure 1: Flow: “add to shopping cart”.



An example test

pre-condition	Scenario: add to shopping cart [stay at cart]
	Given screen is detail
	And cart_filled is false
user steps	When click @addtocart
	And click @cart
post-condition	And not see @empty_cart_msg
	Then screen is cart
	And set cart_filled to true

Figure 1: Flow: “add to shopping cart”.



An example test

```
Scenario: add to shopping cart [stay at cart]
  Given screen is detail
  And cart_filled is false
  When click @addtocart
  And click @cart
  And not see @empty_cart_msg
  Then screen is cart
  And set cart_filled to true
```

User-defined
“abstract property”

Figure 1: Flow: “add to shopping cart”.



“Abstract properties are intended to keep track of the invisible portions of app states, which can often be crucial for writing robust tests.”



An example test

```
Scenario: add to shopping cart [stay at cart]
  Given screen is detail
  And cart_filled is false
  When click @addtocart
  And click @cart
  And not see @empty_cart_msg
  Then screen is cart
  And set cart_filled to true
```

User-defined
“abstract property”

Figure 1: Flow: “add to shopping cart”.



An example test

User defined action

```
Scenario: add to shopping cart [stay at cart]
  Given screen is detail
  And cart_filled is false
  When click @addtocart
  And click @cart
  And not see @empty_cart_msg
  Then screen is cart
  And set cart_filled to true
```

"Add to Cart"
Button

Figure 1: Flow: “add to shopping cart”.



AppFlow Workflow

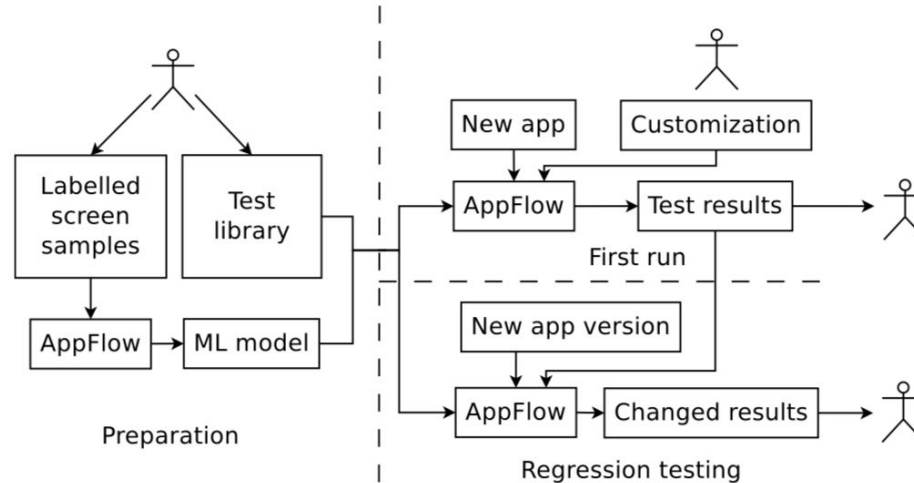


Figure 2: Workflow of APPFLOW. The stick figure here represents developer intervention.



AppFlow Workflow

Phase 1

mostly one-time,
prepares
AppFlow for
testing a **new
category of
apps**

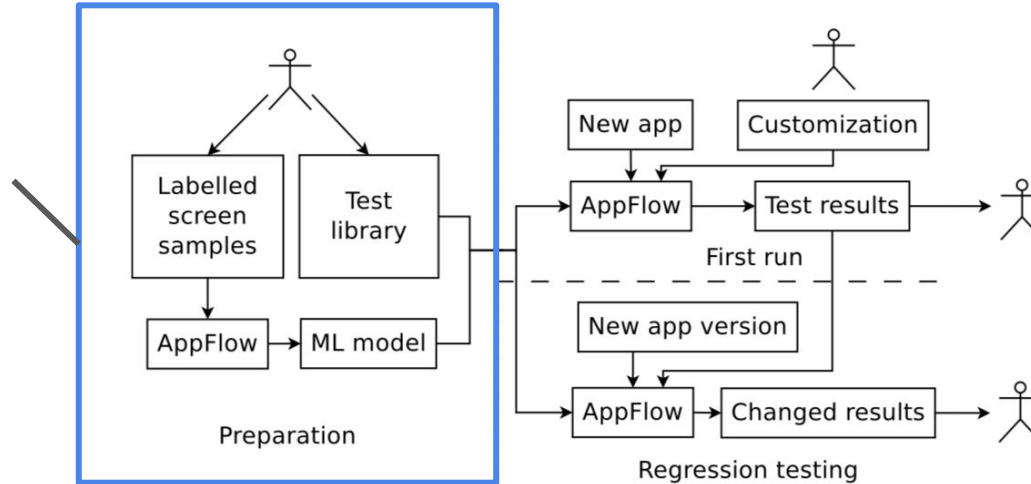


Figure 2: Workflow of APPFLOW. The stick figure here represents developer intervention.



AppFlow Workflow

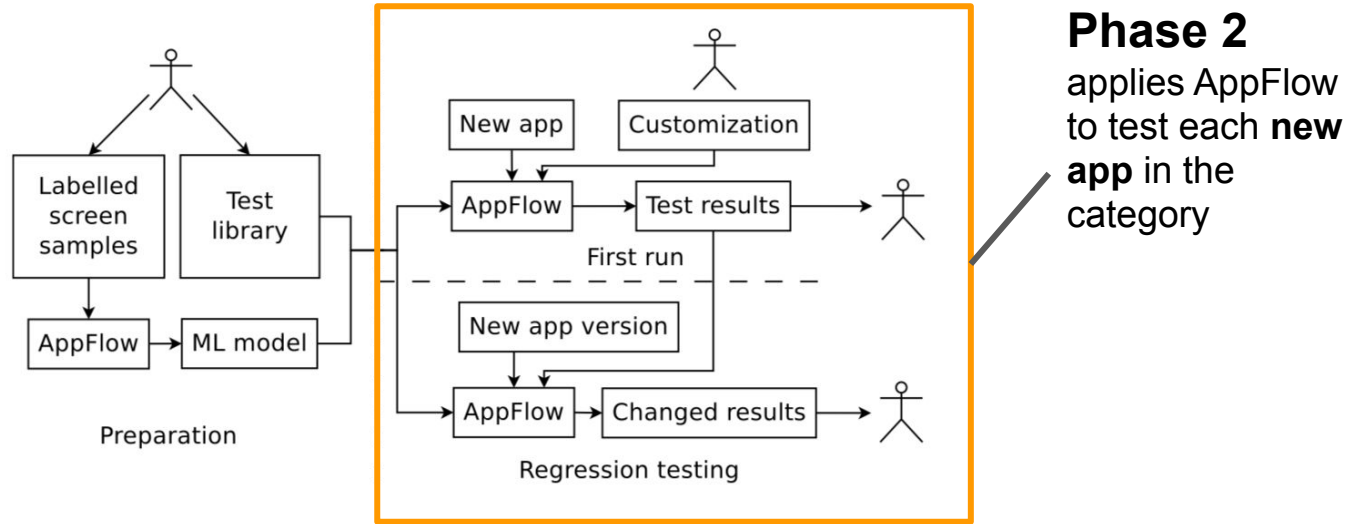


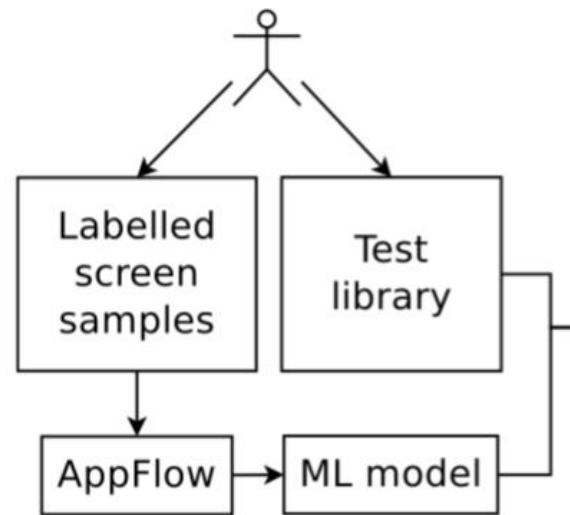
Figure 2: Workflow of APPFLOW. The stick figure here represents developer intervention.



Phase 1: Preparing a new category

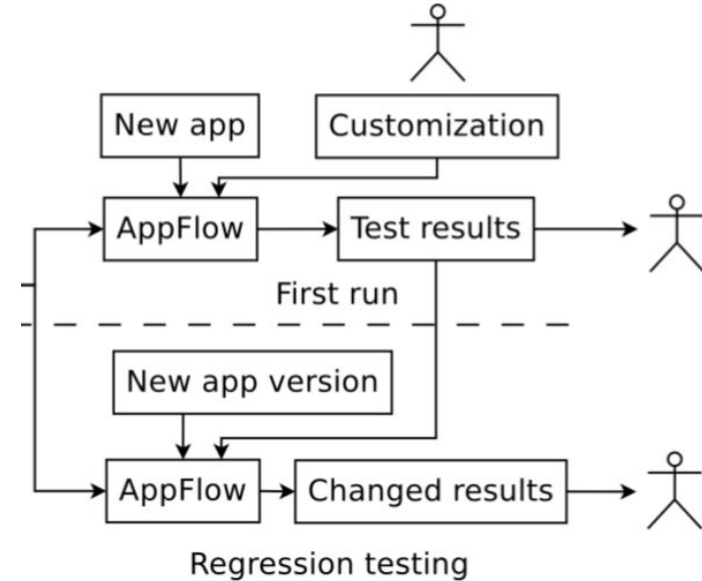
1. Create a test library in AppFlow that contains common flows for category
2. Define canonical *screens* and *widgets*
3. Use AppFlow utilities to **capture** and **label** a dataset of screens and widgets
4. Add samples from other app categories*
5. AppFlow **extracts key features** from each sample and **learns classifiers** to **recognize** screens and widgets based on them

*Sometimes apps in different categories share similar screens



Phase 2: Adding a new app

1. Customize library for app
 - a. Use AppFlow GUI to detect and fix errors in UI detection
 - b. Add custom test flows to accommodate app
2. Run test cases
 - a. AppFlow uses the flows in the test library to synthesize full tests
 - b. At first, only the “start app” flow is active, discovers more flows
 - c. Process terminates when no more flows need to be tested

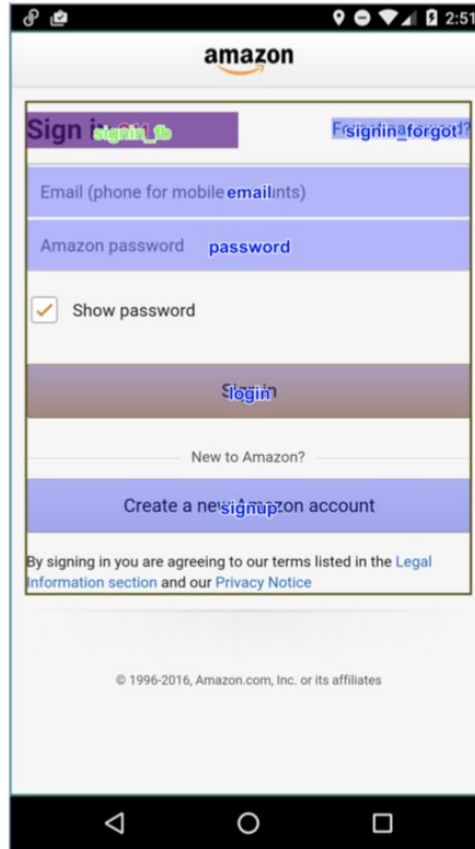


UI Recognition

- Feature selection includes description text, size, whether it is clickable; the UI layout of the object; and the graphics.
- **Classifying Screens**
 - Inputs: UI screenshot, code class-naming
 - Output: canonical screen
- **Classifying Widgets (“interactables”)**
 - Inputs: Widget text, widget context, widget metadata, neighbour information, OCR, graphical features
 - Output: canonical widget or “not a widget”



AppFlow GUI



Please enter label

label for 11: (Sign in) #

Label:

signin_forgot signin_fb signin_gg

account cart_count email password login

share placeorder signin signup cat

menu back cart search continue

Cancel OK



Writing flows

- Follows ‘Gherkin’s syntax’ (Behavior-Driven Development)
 - “Unlike in Gherkin which use natural languages for the conditions and step, AppFlow uses visible and abstract properties”
 - Pre-condition → `Given`
 - Steps → `When`
 - Post-condition → `Then`
- Verbs are common operations and checks, such as “see”, “click”, and “text”
- Widgets can be canonical (assigned) ones or real (defined in library)
 - Canonical ones are referenced with `@<canonical widget name>`



Examples

Scenario: perform user login

Given screen is signin

And loggedin is false

When text @username '@email'

And text @password '@password'

And click @login

Then screen is not signin

And set loggedin to true

Scenario: enter shopping cart [signed in]

Given screen is main

And loggedin is true

When click @cart

Then screen is cart



AppFlow Best Practices

1. Flows should be modular for re-use
2. Test-flows should only refer to canonical screens and widgets
 - a. Avoids string checking-- if you are looking for a screen refer to that screen
3. Reduce rare flows in library
 - a. Avoids test-debugging for future developers
4. Keep flows simple
 - a. More properties increases test-time
 - b. (User-centered!)



Results

- Six main metrics for evaluation centered around usability and impact
- For 40 and 20 top apps in shopping and news respectively
 - 55.2%, 53% (test re-use)
 - 90.2%, 81.5% (screen detection)
 - 88.7%, 85.9% (widget detection)
 - 5.7, 4.5 (# of average flow lines)
- JackThreads
 - 46.6% of the test cases can be created automatically

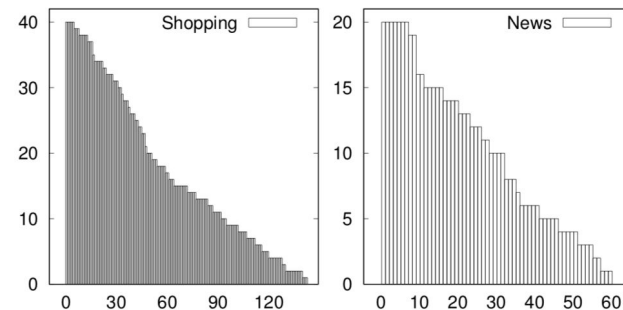


Figure 4: Number of apps each flow can test. The x-axis shows the flows, and the y-axis show the number of apps.

Discussion

- AppFlow market impact
- AppFlow for prototyping and building
- Your thoughts???

