A. Memon, I. Banerjee, and A. Nagarajan, "GUI ripping: Reverse engineering of graphical user interfaces for testing," *Proceedings of the 20th Working Conference on Reverse Engineering*, vol. 0, p. 260, 2003.

T. Azim and I. Neamtiu, "Targeted and depth-first exploration for systematic testing of Android apps," in *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages Applications*, OOPSLA, pp. 641–660, 2013.

W. Choi, G. Necula, and K. Sen, "Guided GUI testing of Android apps with minimal restart and approximate learning," in *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA, pp. 623–640, 2013.

W. Yang, M. Prasad, and T. Xie, "A grey-box approach for automated GUI-model generation of mobile applications," in *Fundamental Approaches to Software Engineering*, vol. 7793 of *Lecture Notes in Computer Science*, pp. 250–265, Springer Berlin Heidelberg, 2013.

- V. Rastogi, Y. Chen, and W. Enck, "Appsplayground: Automatic security analysis of smartphone applications," in *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, CODASPY, pp. 209–220, 2013.
- C. Hu and I. Neamtiu, "A GUI bug finding framework for Android applications," in *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC, pp. 1490–1491, ACM, 2011.
- A. Machiry, R. Tahiliani, and M. Naik, "Dynodroid: An input generation system for Android apps," in *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE, pp. 224–234, 2013.
- S. Anand, M. Naik, M. J. Harrold, and H. Yang, "Automated concolic testing of smartphone apps," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE, pp. 59:1–59:11, 2012.

N. Mirzaei, S. Malek, C. S. Păsăreanu, N. Esfahani, and R. Mahmood, "Testing Android apps through symbolic execution," *SIGSOFT Softw. Eng. Notes*, vol. 37, no. 6, pp. 1–5, 2012.

D. Amalfitano, A. Fasolino, and P. Tramontana, "Reverse engineering finite state machines from rich internet applications," in *Proceedings of the 15th Working Conference on Reverse Engineering*, WCRE, pp. 69–73, 2008.

D. Amalfitano, A. Fasolino, and P. Tramontana, "Experimenting a reverse engineering technique for modelling the behaviour of rich internet applications," in *Proceedings of the IEEE International Conference on Software Maintenance*, ICSM, pp. 571–574, 2009.

C.-Y. Huang, J.-R. Chang, and Y.-H. Chang, "Design and analysis of GUI test-case prioritization using weight-based methods," *Journal of Systems and Software*, vol. 83, no. 4, pp. 646–659, 2010.

L. von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum, "reCAPTCHA: Human-based character recognition via web security measures," *Science*, vol. 321, no. 5895, pp. 1465–1468, 2008.

N. Chen and S. Kim, “Puzzle-based automatic testing: Bringing humans into the loop by solving puzzles,” in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ASE, pp. 140–149, 2012.

User Guided Automation for Testing Mobile Apps

Xiujiang Li[1] Yanyan Jiang[1] Yepang Liu[2]
Chang Xu[1] Xiaoxing Ma[1] Jian Lu[1]

[1]Dept. of Computer Science and Technology
Nanjing University
Nanjing, China

[2]Dept. of Computer Science and Engineering
The Hong Kong University of Science and Technology
Hong Kong, China

The 21th Asia-Pacific Software Engineering Conference
December 2014

Outline

Motivation - Mobile market is growing

Device Type	2013	2014	2015
PCs	296.1	276.7	263
Tablets	195.4	270.7	349.1
Smartphones	1807	1895.1	1952.9
Others	21.2	37.2	62
Total	2319.6	2479.8	2627

Figure: Worldwide Device Shipments (Millions of Units)

Source: Gartner

Motivation - Mobile market is growing

86.33% 87.34% 87.63%

Device Type	2013	2014	2015
PCs	296.1	276.7	263
Mobile Devices	195.4	270.7	349.1
	1807	1895.1	1952.9
Others	21.2	37.2	62
Total	2319.6	2479.8	2627

Motivation - Mobile market is growing

	86.33%	87.34%	87.63%
Device Type	**2013**	**2014**	**2015**
PCs	296.1	276.7	263
Mobile Devices	195.4	270.7	349.1
	1807	1895.1	1952.9
Others	21.2	37.2	62
Total	2319.6	2479.8	2627

Mobile devices have become the de facto computers!

Motivation - Mobile bugs are emerging

Motivation - Mobile bugs are emerging

- Mobile apps are rarely thoroughly tested

Motivation - Mobile bugs are emerging

- Mobile apps are rarely thoroughly tested
    - Complex user interactions (rich hardware)

Motivation - Mobile bugs are emerging

- Mobile apps are rarely thoroughly tested
  - Complex user interactions (rich hardware)
  - Release apps quickly (competitive app markets)

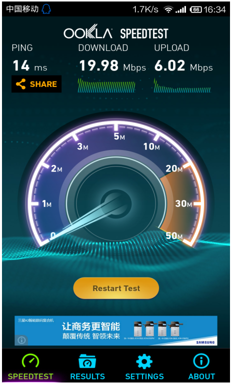
Motivation - Mobile bugs are emerging

- Mobile apps are rarely thoroughly tested
  - Complex user interactions (rich hardware)
  - Release apps quickly (competitive app markets)

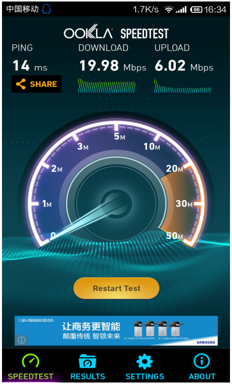
Problem

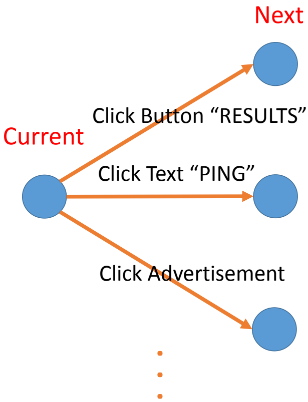
- Automated testing or testing assistance techniques are highly desirable!

Motivation - Most existing automated testing

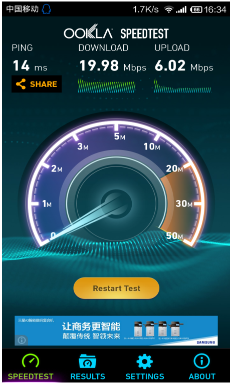


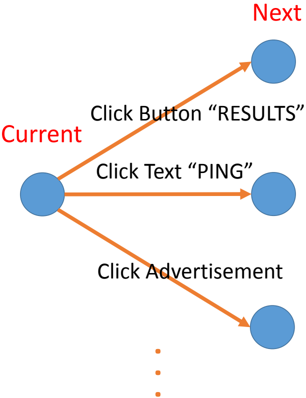
Motivation - Most existing automated testing





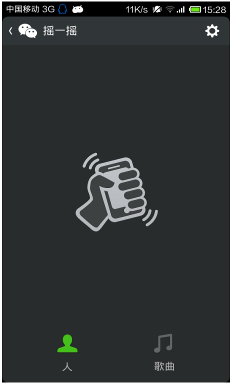
Motivation - Most existing automated testing



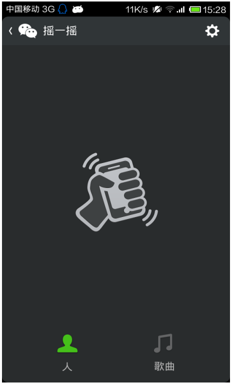


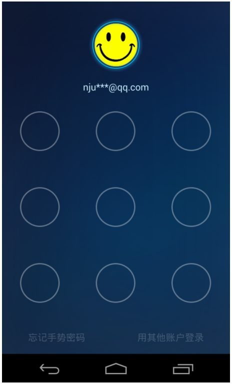
GUI model

Motivation - Big challenges

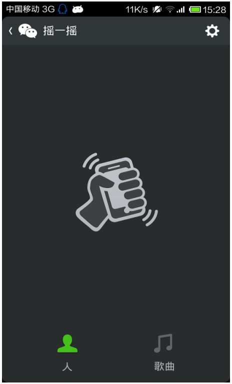


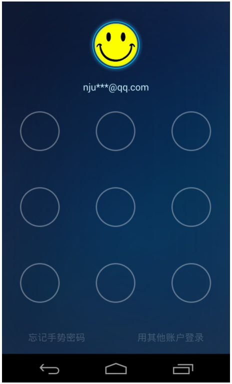
Motivation - Big challenges



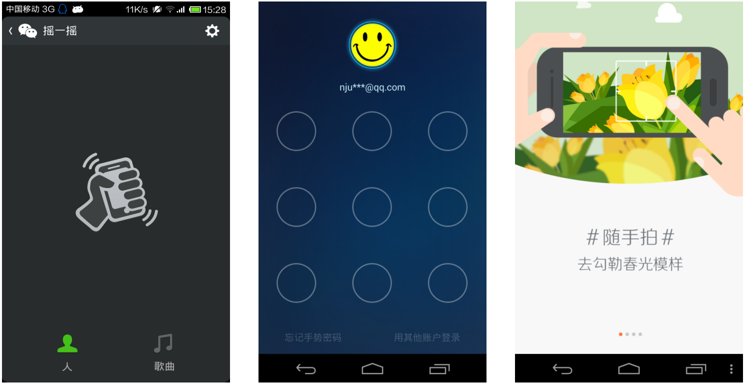


Motivation - Big challenges





Motivation - Big challenges



Motivation

- Complex interactions hinder test automation!

Overview

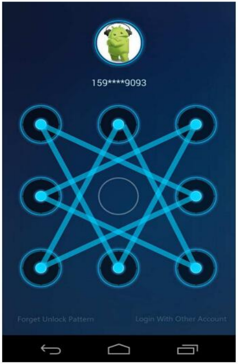
- User Guided Automation for Testing Mobile Apps

Overview

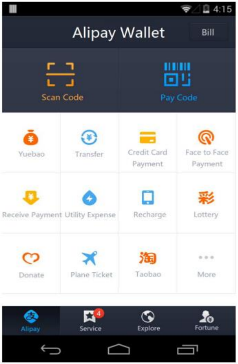
- User Guided Automation for Testing Mobile Apps
  - Record & replay for mobile device
  - Existing automated testing approaches

Overview

- User Guided Automation for Testing Mobile Apps
  - Record & replay for mobile device
  - Existing automated testing approaches



S1



S2

Approach - Step 1

- UGA: Three-stage approach

Approach - Step 1

- UGA: Three-stage approach
  - Step 1: User Trace Recording

Approach - Step 1

- UGA: Three-stage approach
  - Step 1: User Trace Recording

	Time	Device	Parameters (type, code and value)	
1.	[500.821153]	/dev/input/event2:	0003 0039 00000398	Press
2.	[500.821184]	/dev/input/event2:	0003 0035 000003eb	
3.	[500.821184]	/dev/input/event2:	0003 0036 00000478	
4.	[500.821184]	/dev/input/event2:	0003 003a 00000032	
5.	[500.821214]	/dev/input/event2:	0000 0000 00000000	
6.	[500.831011]	/dev/input/event2:	0003 0039 ffffffff	Release
7.	[500.831042]	/dev/input/event2:	0000 0000 00000000	
8.	[502.778547]	/dev/input/event2:	0003 0039 00000399	Press
9.	[502.778547]	/dev/input/event2:	0003 0035 00000535	
10.	[502.778577]	/dev/input/event2:	0003 0036 00000489	
11.	[502.778577]	/dev/input/event2:	0003 003a 00000031	
12.	[502.778577]	/dev/input/event2:	0000 0000 00000000	
13.	[502.930844]	/dev/input/event2:	0003 0039 ffffffff	Release
14.	[502.930874]	/dev/input/event2:	0000 0000 00000000	
15.	[504.000002]	…		

Approach - Step 2 & 3

- Step 2: Stop-point Identification

Approach - Step 2 & 3

- Step 2: Stop-point Identification
  - After executing event e, a new screen pops up (i.e., a new activity is created in Android) or a new GUI container is dynamically created

Approach - Step 2 & 3

- Step 2: Stop-point Identification
  - After executing event e, a new screen pops up (i.e., a new activity is created in Android) or a new GUI container is dynamically created
  - There has been a large interval between two continuous events, it is very likely that the user creating this trace has spent much time on checking an apps GUI to explore next interactions for trying more functionalities

Approach - Step 2 & 3

- Step 2: Stop-point Identification
  - After executing event e, a new screen pops up (i.e., a new activity is created in Android) or a new GUI container is dynamically created
  - There has been a large interval between two continuous events, it is very likely that the user creating this trace has spent much time on checking an apps GUI to explore next interactions for trying more functionalities
- Step 3: User-guided Automated Testing

Approach - Step 2 & 3

- Step 2: Stop-point Identification
  - After executing event e, a new screen pops up (i.e., a new activity is created in Android) or a new GUI container is dynamically created
  - There has been a large interval between two continuous events, it is very likely that the user creating this trace has spent much time on checking an apps GUI to explore next interactions for trying more functionalities
- Step 3: User-guided Automated Testing
  - UGA replays the app until reaching each stop point

Approach - Step 2 & 3

- Step 2: Stop-point Identification
  - After executing event e, a new screen pops up (i.e., a new activity is created in Android) or a new GUI container is dynamically created
  - There has been a large interval between two continuous events, it is very likely that the user creating this trace has spent much time on checking an apps GUI to explore next interactions for trying more functionalities
- Step 3: User-guided Automated Testing
  - UGA replays the app until reaching each stop point
  - Existing automated testing can then be applied

Approach - Step 2 & 3

- Step 2: Stop-point Identification
  - After executing event e, a new screen pops up (i.e., a new activity is created in Android) or a new GUI container is dynamically created
  - There has been a large interval between two continuous events, it is very likely that the user creating this trace has spent much time on checking an apps GUI to explore next interactions for trying more functionalities
- Step 3: User-guided Automated Testing
  - UGA replays the app until reaching each stop point
  - Existing automated testing can then be applied
    - RND
    - DFS

Approach - RND and DFS in Step 3

Approach - RND and DFS in Step 3

Algorithm 1: The RND Algorithm

```
function RND(v, ℓ)
begin
    for i ∈ {1, 2, ..., ℓ} do
        A ← GetAllEnabledActions(v);
        if A ≠ ∅ then
            a ← randomChoose(A);
            execute a, leading to activity v′;
            v ← v′;
        else
            break;
```

Approach - RND and DFS in Step 3

Algorithm 1: The RND Algorithm

function RND(v, ℓ)
begin
  for $i \in \{1, 2, \dots, \ell\}$ **do**
    $A \leftarrow$ *GetAllEnabledActions*(v);
    if $A \neq \varnothing$ **then**
      $a \leftarrow$ randomChoose(A);
      execute a, leading to activity v';
      $v \leftarrow v'$;
    else
      break;

Algorithm 2: The DFS Algorithm

Data: global variable V with initial value $\varnothing$
function DFS(v)
begin
  $V \leftarrow V \cup \{v\}$;
  for each action $a \in$ *GetAllEnabledActions*(v) **do**
    execute a, leading to activity v';
    if $v' \notin V$ **then**
      DFS(v');
    if $v \neq v'$ **then**
      back to activity v;

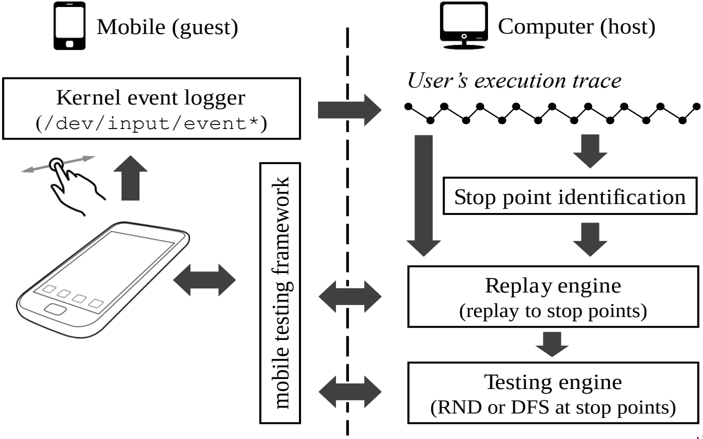
Approach - UGA implementation

Approach - UGA implementation

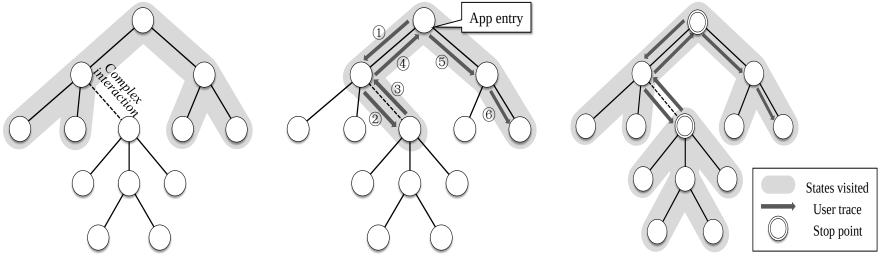
Platform: Android

Approach - UGA implementation

Platform: Android



Approach - How does UGA work (1)



Automated testing only | User trace only | UGA

Approach - How does UGA work (2)

- $\boxed{\text{Human}}$ has insights

Approach - How does UGA work (2)

- $\boxed{\text{Human}}$ has insights
    - Apps are designed for human

Approach - How does UGA work (2)

- $\boxed{\text{Human}}$ has insights
  - Apps are designed for human
  - Human can understand the hint information in the apps

Approach - How does UGA work (2)

- $\boxed{\text{Human}}$ has insights
  - Apps are designed for human
  - Human can understand the hint information in the apps
  - So, human can give the structure of the apps

Approach - How does UGA work (2)

- $\boxed{\text{Human}}$ has insights
  - Apps are designed for human
  - Human can understand the hint information in the apps
  - So, human can give the structure of the apps
- $\boxed{\text{Computer}}$ can complement the human

Approach - How does UGA work (2)

- Human has insights
  - Apps are designed for human
  - Human can understand the hint information in the apps
  - So, human can give the structure of the apps
- Computer can complement the human
  - Computer has the patience to do the boring work

Approach - How does UGA work (2)

- $\boxed{\text{Human}}$ has insights
  - Apps are designed for human
  - Human can understand the hint information in the apps
  - So, human can give the structure of the apps
- $\boxed{\text{Computer}}$ can complement the human
  - Computer has the patience to do the boring work
  - So, computer can fill in the structure above

Approach - How does UGA work (2)

- $\boxed{\text{Human}}$ has insights
  - Apps are designed for human
  - Human can understand the hint information in the apps
  - So, human can give the structure of the apps
- $\boxed{\text{Computer}}$ can complement the human
  - Computer has the patience to do the boring work
  - So, computer can fill in the structure above
- User Guided Automation (UGA)

Approach - How does UGA work (2)

- Human has insights
  - Apps are designed for human
  - Human can understand the hint information in the apps
  - So, human can give the structure of the apps
- Computer can complement the human
  - Computer has the patience to do the boring work
  - So, computer can fill in the structure above
- User Guided Automation (UGA)
  - Combine Human with Computer

Evaluation - Experimental setup

- Compare four approaches

Evaluation - Experimental setup

- Compare four approaches
  - RND
  - DFS

Evaluation - Experimental setup

- Compare four approaches
    - RND
    - DFS
    - UGA+RND / UGA+DFS

Evaluation - Experimental setup

- Compare four approaches
  - RND
  - DFS
  - UGA+RND / UGA+DFS
    - record at most 10 minutes
    - select 10 stop points

Evaluation - Experimental setup

- Compare four approaches
  - RND
  - DFS
  - UGA+RND / UGA+DFS
    - record at most 10 minutes
    - select 10 stop points
- Evaluation metric

Evaluation - Experimental setup

- Compare four approaches
  - RND
  - DFS
  - UGA+RND / UGA+DFS
    - record at most 10 minutes
    - select 10 stop points
- Evaluation metric
  - method coverage

Evaluation - Experimental setup

- Compare four approaches
  - RND
  - DFS
  - UGA+RND / UGA+DFS
    - record at most 10 minutes
    - select 10 stop points
- Evaluation metric
  - method coverage
- PC
  - Intel Q9550 CPU
  - 4GB RAM
  - Running Ubuntu 13.04

Evaluation - Experimental setup

- Compare four approaches
  - RND
  - DFS
  - UGA+RND / UGA+DFS
    - record at most 10 minutes
    - select 10 stop points
- Evaluation metric
  - method coverage
- PC
  - Intel Q9550 CPU
  - 4GB RAM
  - Running Ubuntu 13.04
- Mobile
  - Google Nexus 4
  - Running Android 4.4.2

Evaluation - Subjects

Table : Android apps used in our experiments

Name	*Category*	*#Download*	*#Activity*	*#Method*	*#Instruction*
Amazon	shopping	5M–10M	3,264	18,431	706,645
Any.do	to-do list	5M–10M	1,433	7,326	334,118
Netease News	news client	500K–1M	1,609	10,806	507,438
Mileage	car manager	500K–1M	221	1,185	51,979
Tippy Tipper	tip calculator	50K–100K	56	238	10,138
Alarm Klock	alarm clock	500K–1M	160	673	30,027
Bing Dictionary	dictionary	10K–50K	581	2,374	137,592

Evaluation - Results (1)

Table : UGA+RND results (method coverage)

Subject	*User trace only*	*RND only*	*UGA+RND*	Δ v.s. *RND*
Amazon	21.9%	25.4%	42.5%	17.1% (1.67×)
Any.do	13.6%	10.8%	25.4%	14.6% (2.36×)
Netease News	25.5%	8.7%	49.2%	40.5% (5.65×)
Mileage	33.4%	27.3%	62.4%	35.1% (2.29×)
Tippy Tipper	45.8%	44.5%	72.3%	27.7% (1.62×)
Alarm Klock	39.2%	41.2%	73.7%	32.5% (1.79×)
Bing Dictionary	13.5%	3.2%	47.1%	43.8% (14.51×)

(median: 32.5%)

Evaluation - Results (2)

Table : UGA+DFS results (method coverage)

Subject	*User trace only*	*DFS only*	*UGA+DFS*	Δ v.s. *DFS*
Amazon	21.9%	25.6%	47.6%	22.0% (1.86×)
Any.do	13.6%	11.7%	31.8%	20.2% (2.73×)
Netease News	25.5%	9.0%	53.5%	44.5% (5.94×)
Mileage	33.4%	27.3%	79.3%	52.1% (2.91×)
Tippy Tipper	45.8%	50.8%	80.7%	29.8% (1.59×)
Alarm Klock	39.2%	45.8%	76.5%	30.8% (1.67×)
Bing Dictionary	13.5%	3.2%	70.6%	67.4% (21.78×)

(median: 30.8%)

Related Work (1)

- Model-based GUI testing for mobile apps

Related Work (1)

- Model-based GUI testing for mobile apps
  - GUITAR [1]

Related Work (1)

- Model-based GUI testing for mobile apps
  - GUITAR [1]
  - States: screens [2] or enabled GUI elements [3]

Related Work (1)

- Model-based GUI testing for mobile apps
  - GUITAR [1]
  - States: screens [2] or enabled GUI elements [3]
  - Transitions: source code [4] or at runtime [2] [3]

Related Work (1)

- Model-based GUI testing for mobile apps
  - GUITAR [1]
  - States: screens [2] or enabled GUI elements [3]
  - Transitions: source code [4] or at runtime [2] [3]
  - Different strategies can be applied on the GUI model

Related Work (1)

- Model-based GUI testing for mobile apps
  - GUITAR [1]
  - States: screens [2] or enabled GUI elements [3]
  - Transitions: source code [4] or at runtime [2] [3]
  - Different strategies can be applied on the GUI model
    - Depth-first search [2] [4]
    - Heuristic-rule based search [3] [5]
    - Random [6] [7]
    - Symbolic or concolic analysis [8] [9]

Related Work (2)

- Human-assistance in GUI testing

Related Work (2)

- Human-assistance in GUI testing
  - Extract GUI model from collected user traces for testing [10] [11]

Related Work (2)

- Human-assistance in GUI testing
  - Extract GUI model from collected user traces for testing [10] [11]
  - Manually identify and prioritize transitions in GUI model [12]

Related Work (2)

- Human-assistance in GUI testing
  - Extract GUI model from collected user traces for testing [10] [11]
  - Manually identify and prioritize transitions in GUI model [12]
- Human-assistance in other fields

Related Work (2)

- Human-assistance in GUI testing
  - Extract GUI model from collected user traces for testing [10] [11]
  - Manually identify and prioritize transitions in GUI model [12]
- Human-assistance in other fields
  - reCAPTCHA [13]

Related Work (2)

- Human-assistance in GUI testing
  - Extract GUI model from collected user traces for testing [10] [11]
  - Manually identify and prioritize transitions in GUI model [12]
- Human-assistance in other fields
  - reCAPTCHA [13]
  - PAT: decompose complex problems into small puzzles for humans to solve [14]

Conclusion

- Propose a novel user-guided testing technique for mobile apps, which is the first attempt that integrate user insights into automated testing techniques
- The experiments show its improvement over existing automated testing techniques in terms of code coverage
- Set a new direction towards cost-effective testing for mobile apps

Future Work

- It could be argued that our UGA's test effectiveness might depend on what user traces are collected and how stop points are identified

Future Work

- It could be argued that our UGA's test effectiveness might depend on what user traces are collected and how stop points are identified
  $\boxed{\text{Human}}$ —> $\boxed{\text{Machine}}$

Future Work

- It could be argued that our UGA's test effectiveness might depend on what user traces are collected and how stop points are identified

  $\boxed{\text{Human}}$ —> $\boxed{\text{Machine}}$

  $\boxed{\text{Machine}}$ —> $\boxed{\text{Human}}$ —> $\boxed{\text{Machine}}$

Future Work

- It could be argued that our UGA's test effectiveness might depend on what user traces are collected and how stop points are identified

  $\boxed{\text{Human}}$ —> $\boxed{\text{Machine}}$

  $\boxed{\text{Machine}}$ —> $\boxed{\text{Human}}$ —> $\boxed{\text{Machine}}$
- Theoretically, any automated testing techniques can be extended for such user guidance

That's all

Thank you!

Thank you!

Q and A