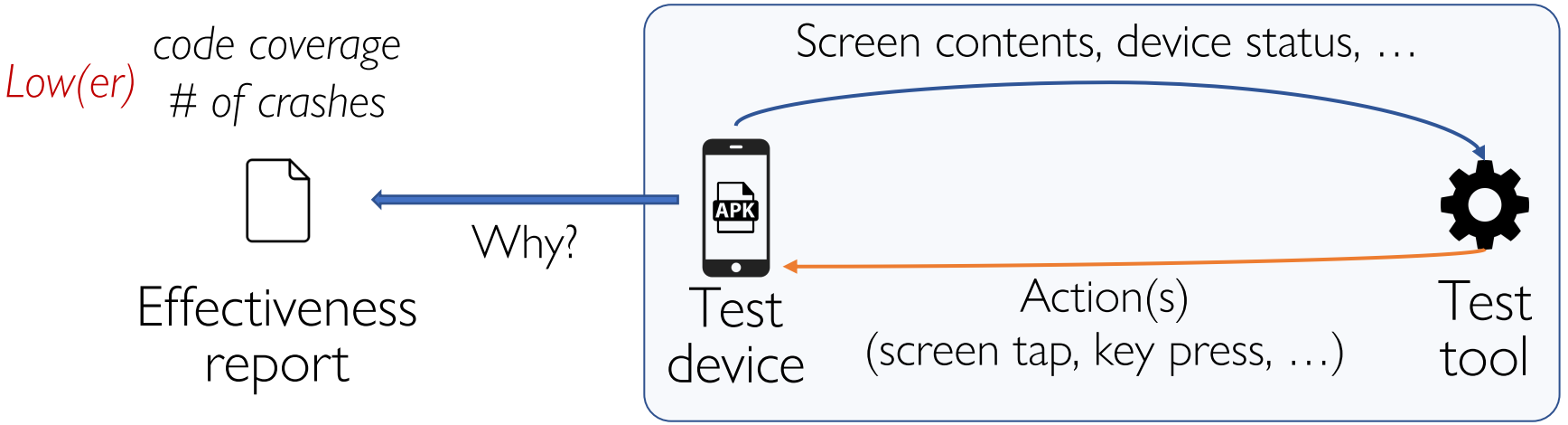
VET: Identifying and Avoiding UI Exploration Tarpits

Wenyu Wang, Wei Yang, Tianyin Xu, Tao Xie

University of Illinois Urbana-Champaign, USA
University of Texas at Dallas, USA
Peking University, China

Toward Understanding UI Testing Effectiveness



Tools may get stuck with *a few* functionalities for *a long* time

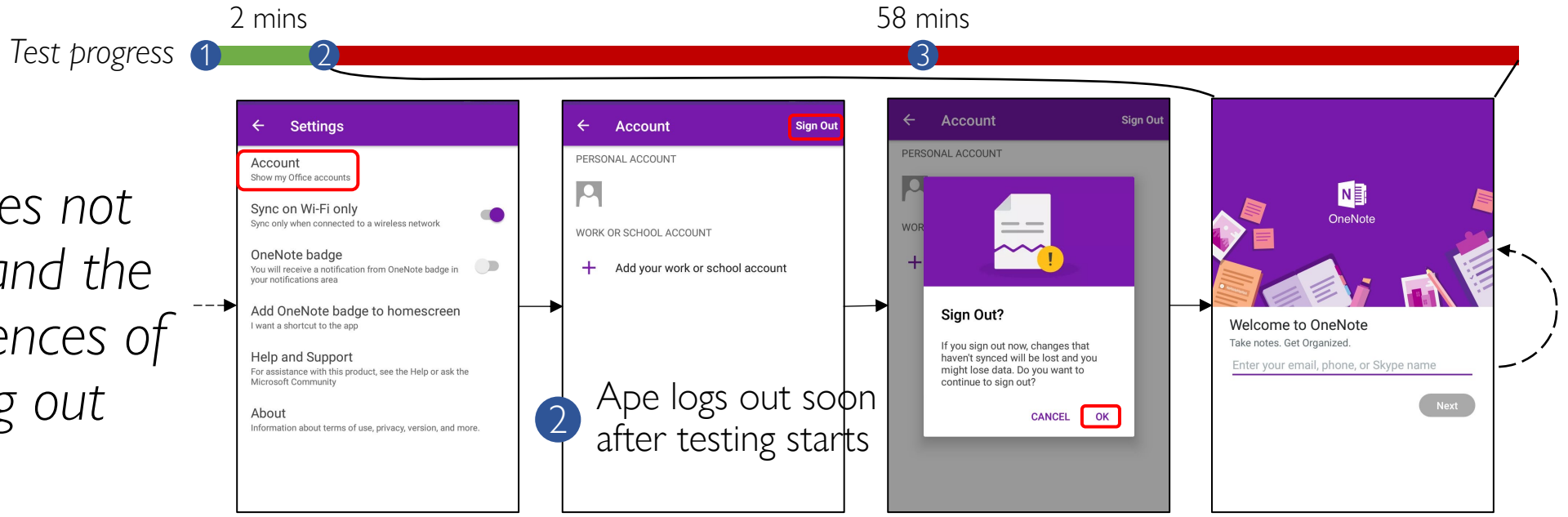
Exploration Tarpits

Motivating Example A: Logging Out

1 *OneNote* requires logging in to access main functionalities

Manually log in before testing starts

3 Ape has to explore pre-login functionalities for most of testing time



Ape does not understand the consequences of logging out

2 Ape logs out soon after testing starts

State-of-the-art tool *Ape*[1] on app *OneNote*

[1] Tianxiao Gu, Chengnian Sun, Xiaoxing Ma, Chun Cao, Chang Xu, Yuan Yao, Qirun Zhang, Jian Lu, and Zhendong Su. *Practical GUI Testing of Android Applications via Model Abstraction and Refinement* (ICSE 2019)

Motivating Example B: Obscure Escape

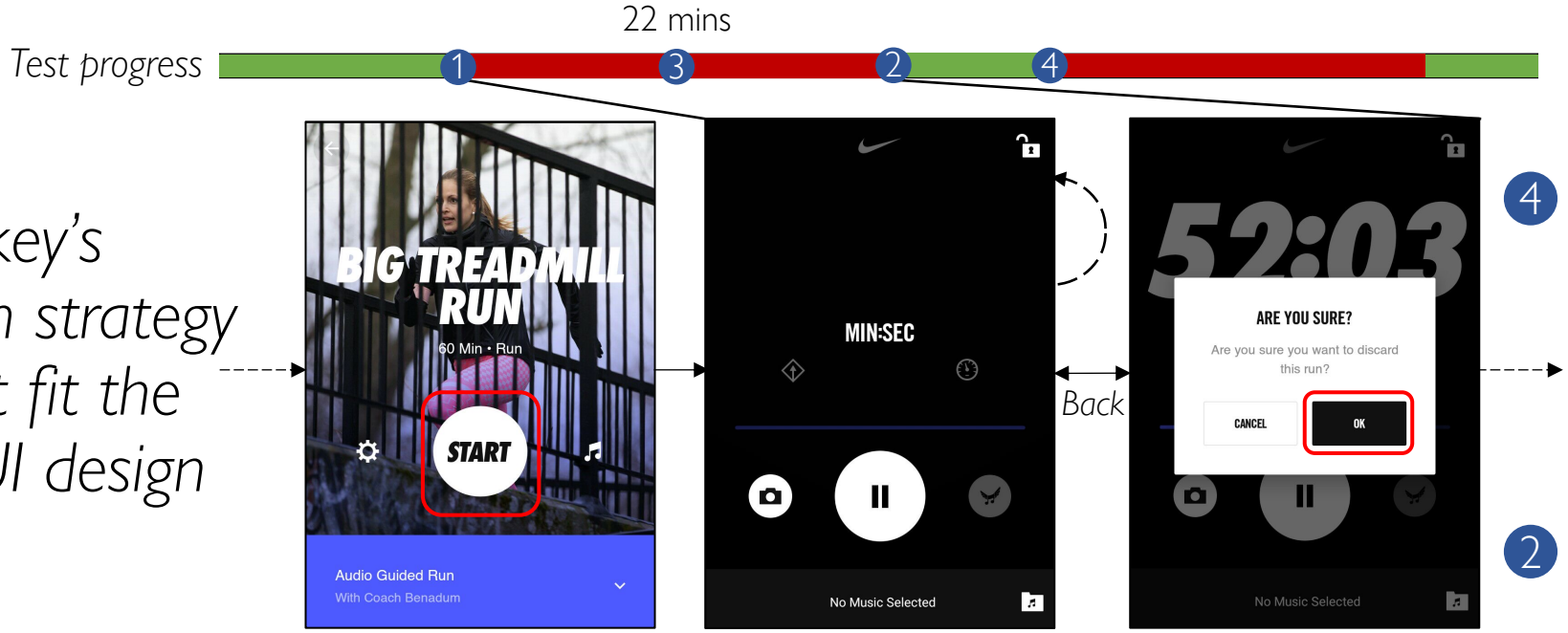
1 Monkey starts exploring a special app functionality

3 Very difficult for Monkey to escape the functionality

Monkey has no understanding of on-screen contents

22 mins

Test progress



Monkey's exploration strategy does not fit the specific UI design

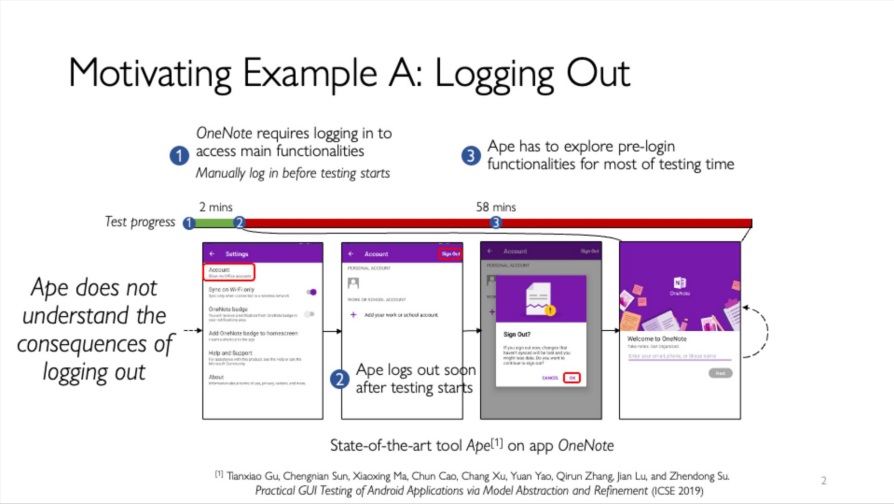
4 Monkey will get trapped again

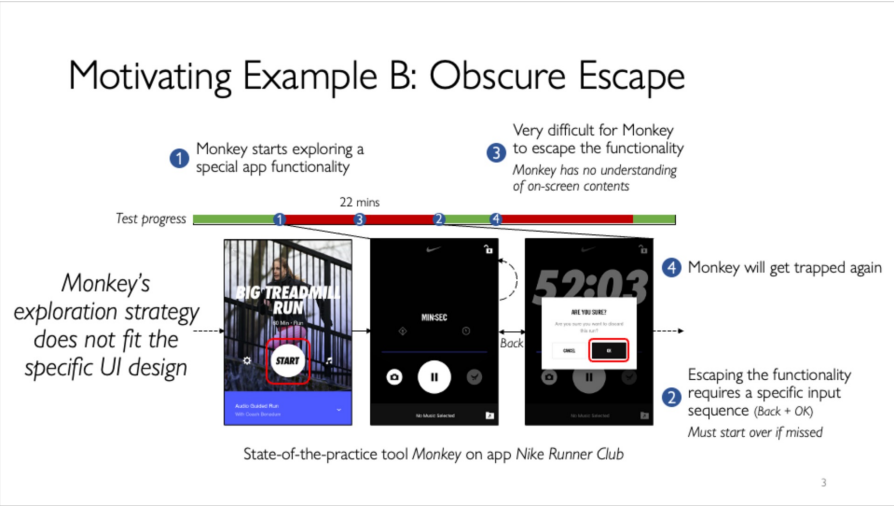
2 Escaping the functionality requires a specific input sequence (*Back* + *OK*)

Must start over if missed

State-of-the-practice tool *Monkey* on app *Nike Runner Club*

Addressing UI Exploration Tarpits





🤔 Manually involve domain knowledge?
- 😢 Need to know tarpits in advance
- 😢 Barely adaptive (apps, tools, environment, …)

🤔 Monitor & Recover[1]?
- 😢 Recovery can be non-trivial (external states)
- 😢 Tarpits still happen (before recovery kicks in)

Learn from history?
- 👍 Prevent / quickly escape from tarpits
- 👍 Achievable with automated techniques
- 👍 Provide human testers with insights

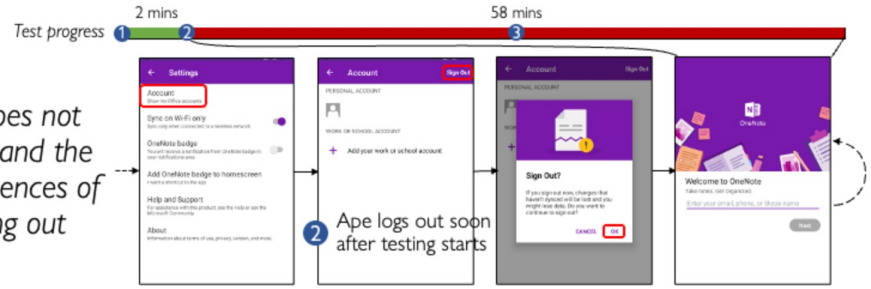
[1] Zhen Dong, Marcel Böhme, Lucia Cojocaru, and Abhik Roychoudhury. *Time-travel Testing of Android Apps* (ICSE 2020)

Identifying Tarpits with Pattern Matching

Motivating Example A: Logging Out

1 *OneNote* requires logging in to access main functionalities
Manually log in before testing starts

3 Ape has to explore pre-login functionalities for most of testing time



State-of-the-art tool *Ape*[1] on app *OneNote*

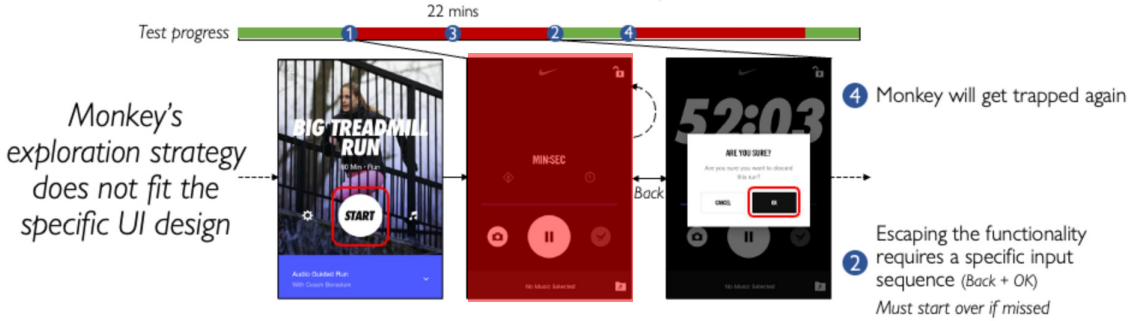
[1] Tianxiao Gu, Chengnian Sun, Xiaoxing Ma, Chun Cao, Chang Xu, Yuan Yao, Qirun Zhang, Jian Lu, and Zhendong Su. *Practical GUI Testing of Android Applications via Model Abstraction and Refinement* (ICSE 2019)

2

Motivating Example B: Obscure Escape

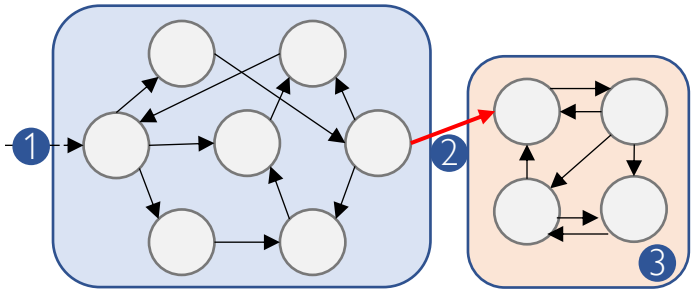
1 Monkey starts exploring a special app functionality

3 Very difficult for Monkey to escape the functionality
Monkey has no understanding of on-screen contents

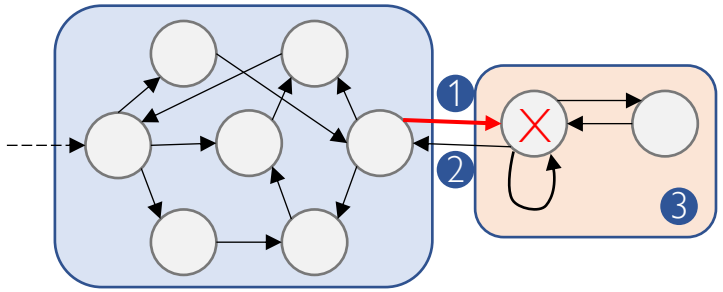


State-of-the-practice tool *Monkey* on app *Nike Runner Club*

3



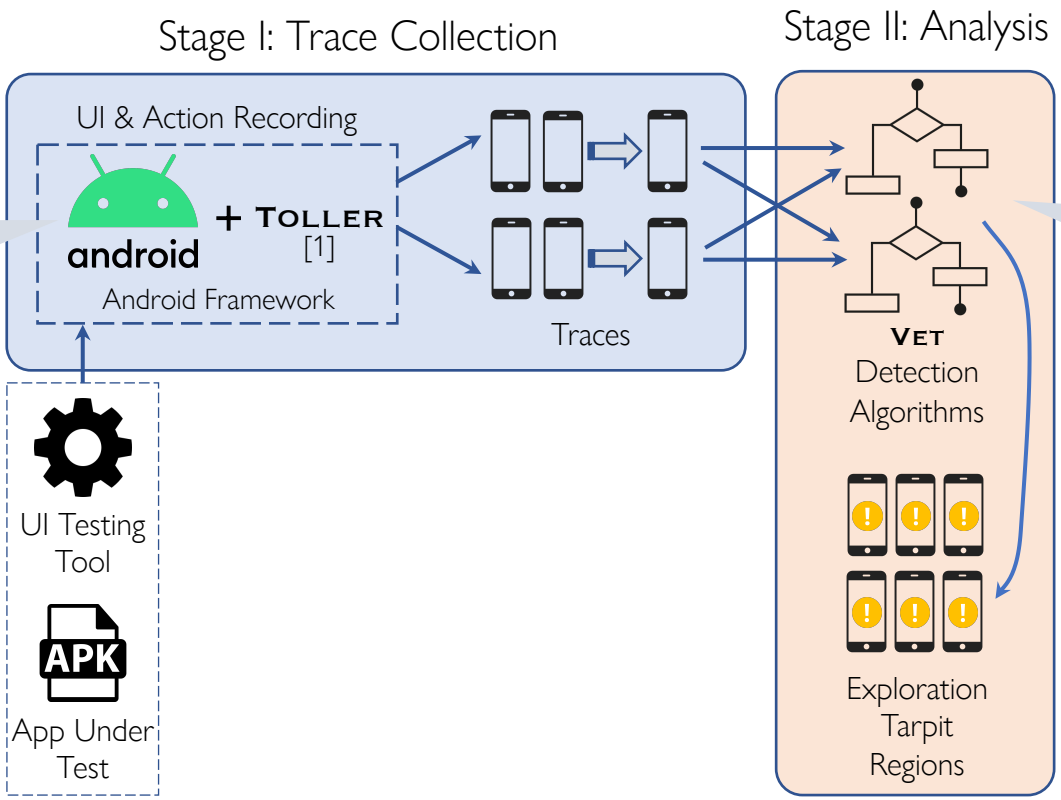
Exploration Space Partition



Excessive Local Exploration

Addressing UI Exploration Tarpits with VET

Ultra-low overhead & tool-agnostic
Works with large industrial apps



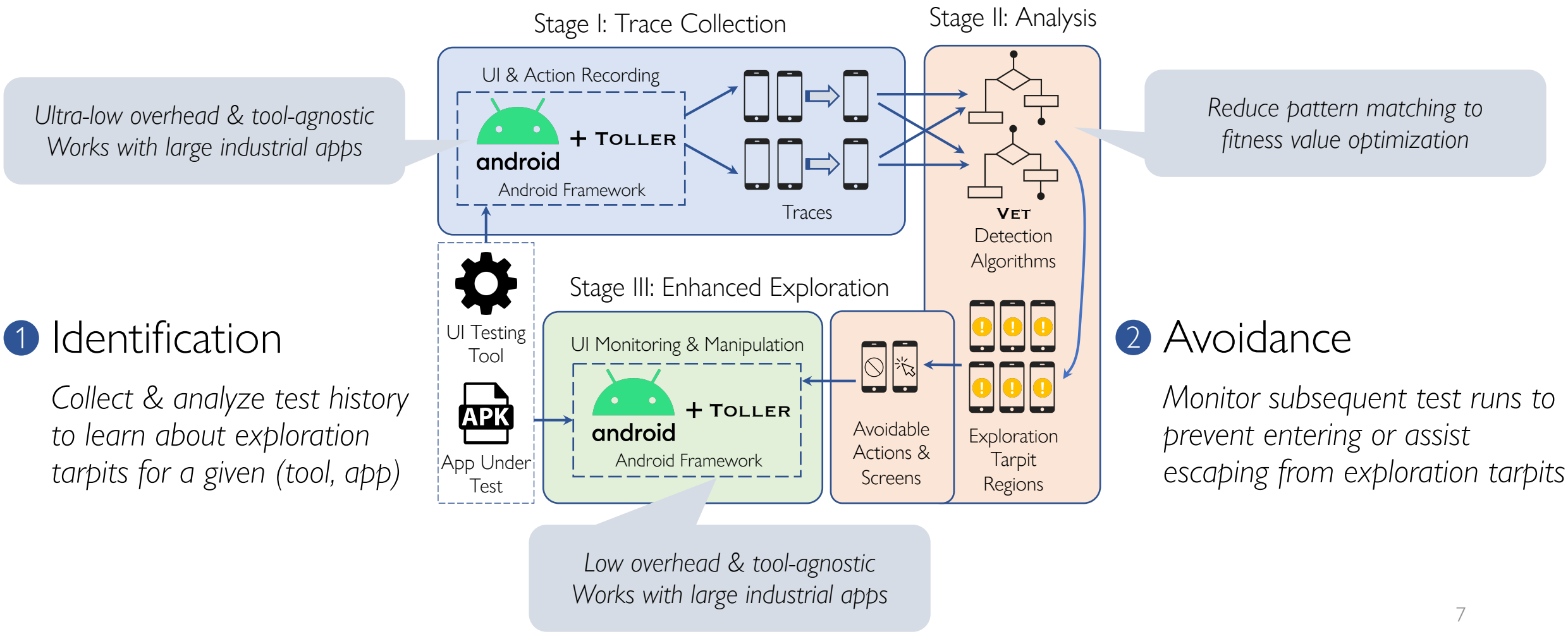
Reduce pattern matching to fitness value optimization

1 Identification

Collect & analyze test history to learn about exploration tarpits for a given (tool, app)

[1] Wenyu Wang, Wing Lam, and Tao Xie.
An Infrastructure Approach to Improving Effectiveness of Android UI Testing Tools (ISSTA 2021)

Addressing UI Exploration Tarpits with VET



Ultra-low overhead & tool-agnostic
Works with large industrial apps

Reduce pattern matching to fitness value optimization

1 Identification

Collect & analyze test history to learn about exploration tarpits for a given (tool, app)

2 Avoidance

Monitor subsequent test runs to prevent entering or assist escaping from exploration tarpits

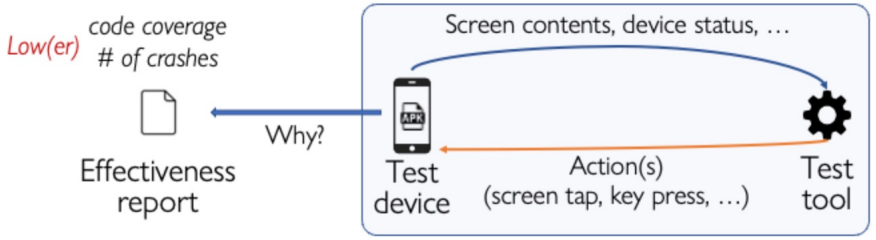
Low overhead & tool-agnostic
Works with large industrial apps

Evaluation

- 16 popular industrial apps, 3 state-of-the-art/practice tools
  - 3 one-hour traces per (tool, app), 144 in total.
  - 131 reported regions, each spanning 10~59 minutes.
- RQ1: How effectively can VET help reveal Android UI testing tool issues with the identified exploration tarpit regions?
  - Manual inspection: 96 regions with identified issues of various categories.
- RQ2: What is the extent of effectiveness improvement of Android UI testing tools through automatic enhancement by VET?
  - Code coverage: cumulative # of distinct methods averaged across apps, +4.4% ~ +15.3%.
  - Crash-triggering capability: # of distinct crashes accumulated across apps, 1.9x ~ 2.1x.
- RQ3: How likely do VET algorithms miss tool issues in their identified exploration tarpit regions?
  - Minor. Please see paper[1] for details.

[1] https://wenyu.io/pub/fse21-vet.pdf

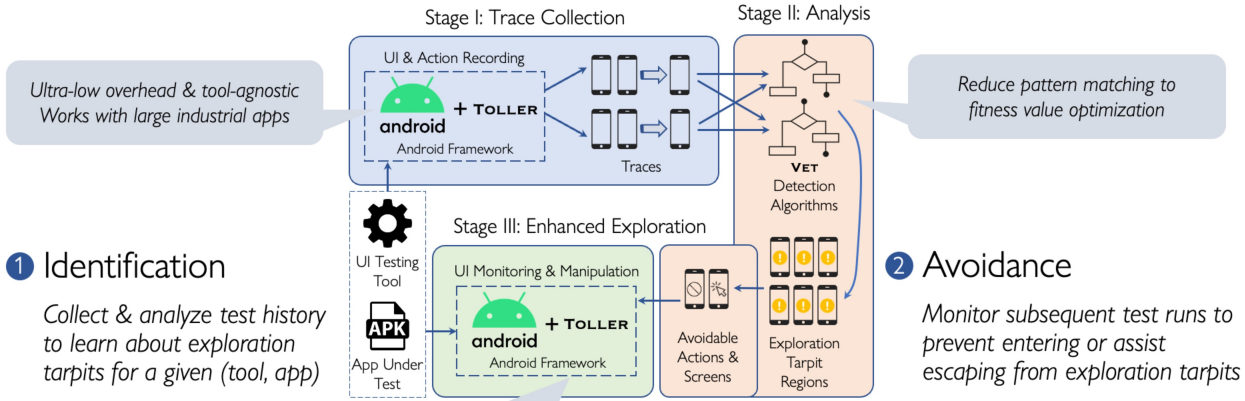
Toward Understanding UI Testing Effectiveness



Tools may get stuck with *a few* functionalities for *a long* time

Exploration Tarpits

Addressing UI Exploration Tarpits with VET



1 Identification

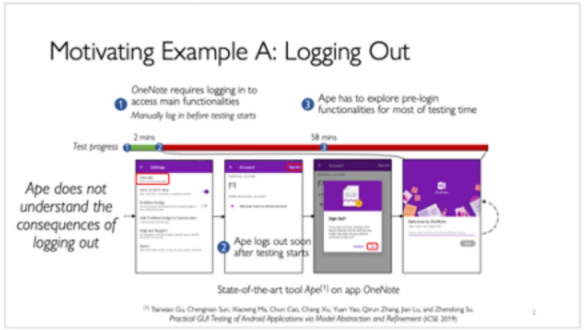
Collect & analyze test history to learn about exploration tarpits for a given (tool, app)

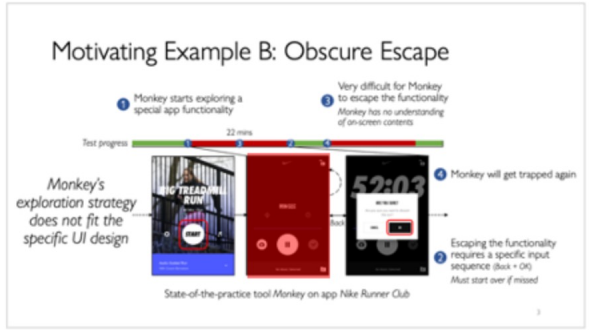
2 Avoidance

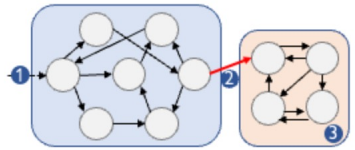
Monitor subsequent test runs to prevent entering or assist escaping from exploration tarpits

This work is partially supported by 3M, UT Dallas, NSF (CNS-1564274, SHF-1816615, CNS-1956007, CCF-2029049), Facebook Research, Microsoft Azure, and Google Cloud.

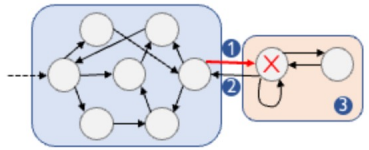
Identifying Tarpits with Pattern Matching







Exploration Space Partition



Excessive Local Exploration

Evaluation

- 16 popular industrial apps, 3 state-of-the-art/practice tools
  - 3 one-hour traces per (tool, app), 144 in total.
  - 131 reported regions, each spanning 10~59 minutes.
- **RQ1:** How effectively can VET help reveal Android UI testing tool issues with the identified exploration tarpit regions?
  - Manual inspection: 96 regions with identified issues of various categories.
- **RQ2:** What is the extent of effectiveness improvement of Android UI testing tools through automatic enhancement by VET?
  - Code coverage: cumulative # of distinct methods averaged across apps, +4.4% ~ +15.3%.
  - Crash-triggering capability: # of distinct crashes accumulated across apps, 1.9x ~ 2.1x.
- **RQ3:** How likely do VET algorithms miss tool issues in their identified exploration tarpit regions?
  - Minor. Please see paper[1] for details.

[1] https://wenyu.io/pub/fse21-vet.pdf