Navi Droid

Guided Bug Crush: Assist Manual GUI Testing of Android Apps via Hint Moves
缺陷消消乐：通过视觉提示帮助测试人员测试安卓应用程序

Zhe Liu , Chunyang Chen , Junjie Wang, Yuekai Huang, Jun Hu, Qing Wang

In IEEE/ACM CHI Conference on Human Factors in Computing Systems (CHI'22)

联系人：刘哲，王俊杰，王青   联系方式：{liuzhe2020, junjie, wq}@iscas.ac.cn

NaviDroid Link: *https://github.com/20200829/Navidroid*

Demo video: *https://www.bilibili.com/video/BV1Ba411W7u2*

Background

- **Automated GUI Testing**
  - *Explore App with random actions*
  - *Mainly functional testing*
  - *Spot critical crash bugs*
  - *Low test coverage*
- **Manual GUI Testing**
  - *Explore App with complex actions*
  - *Discover more diverse bugs*
  - *Test more pages*
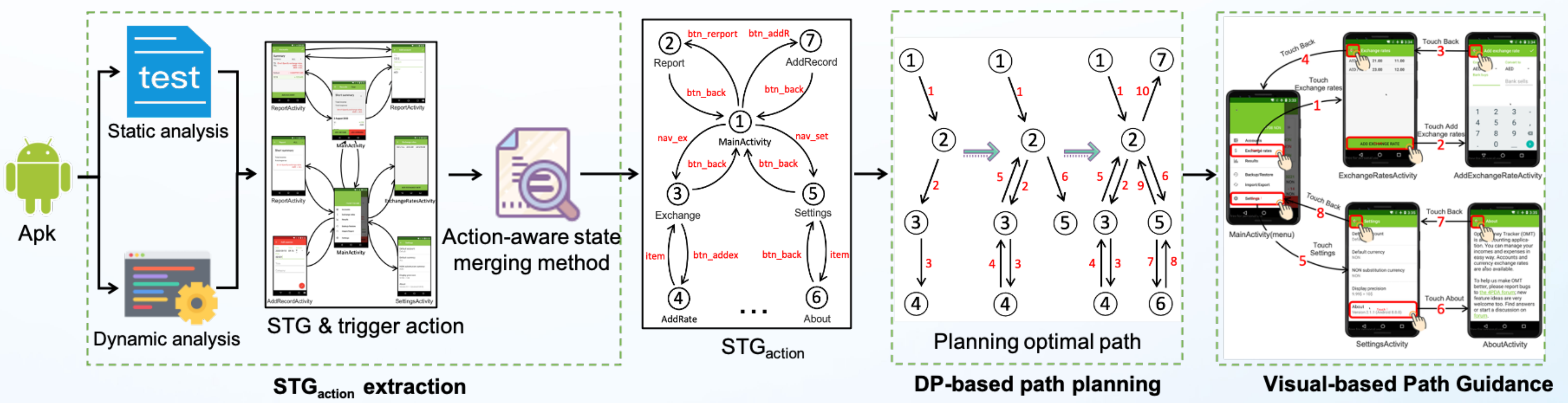 - *Related to user experience*

Motivation

- Combine the advantages of automated & manual testing
- Guide testers to avoid repetition & missing

Example of one tester's exploration graph. Number below each page is the visit time of the tester. Some pages waste time due to repeated exploration, and others are not explored.
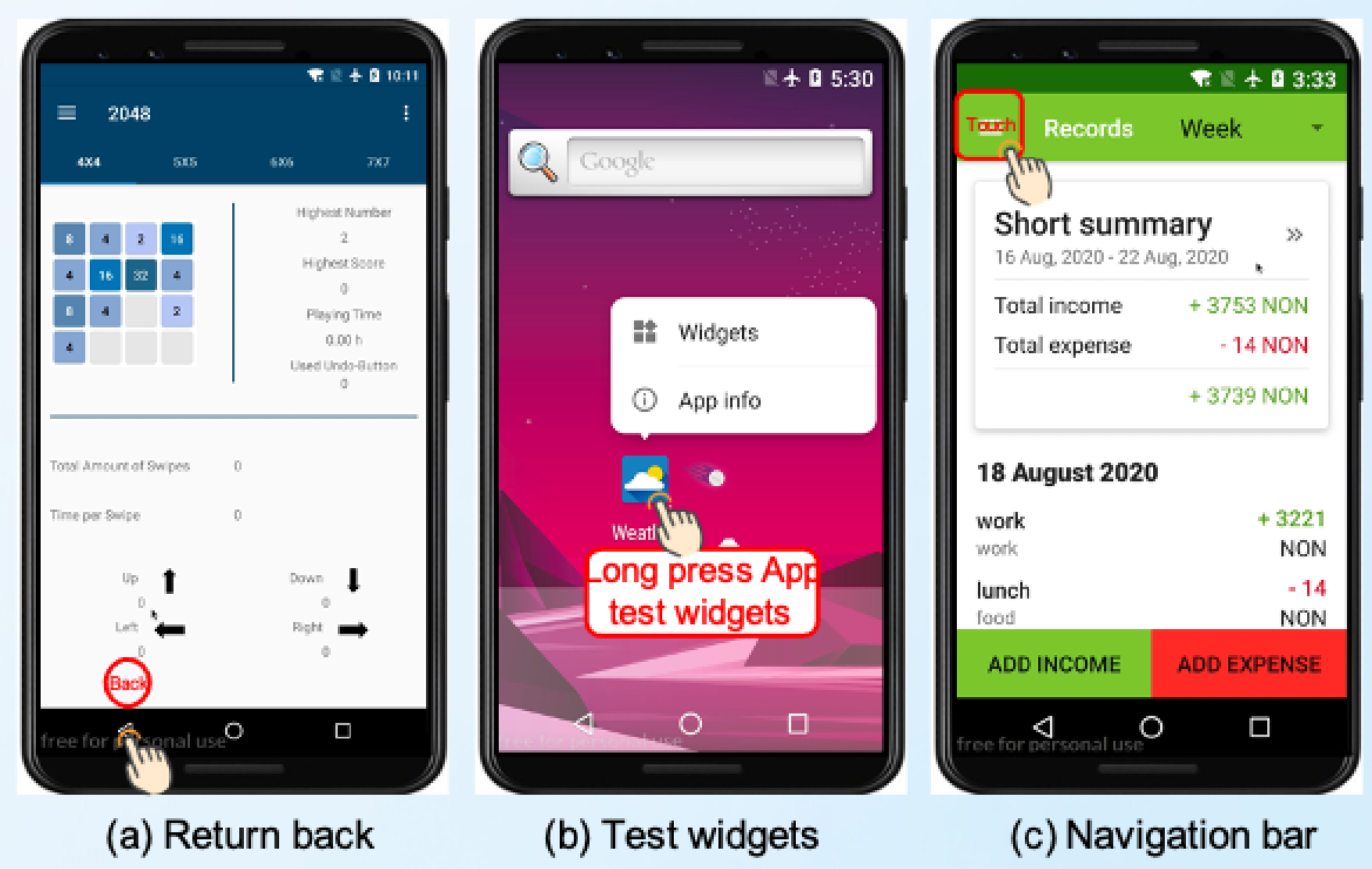
Approach

NaviDroid will trace testers' testing steps and help navigate or remind testers with un-explored pages by explicit visual annotations (e.g., red bounding box) in the run-time page. This process is similar to the flashing candies (hint/suggested moves) when a player hesitates to make a move in playing Candy Crush.

STG_action extraction   STG_action   DP-based path planning   Visual-based Path Guidance

- **STG_action Extraction**
  - *Combine both static analysis and dynamic exploration to extract state transition graph*
  - *Static STG extraction & action detection*
  - *Dynamic STG & action extraction*
  - *Context-aware state merging*
- **DP-based Path Planning**
  - *Aim at covering all the states of the app with few repeated exploration steps*
  - *Real-time path planning*

- **Visual-based Path Guidance**
  - *Three main types of trigger actions*

(a) Return back   (b) Test widgets   (c) Navigation bar

Evaluation

- **Effectiveness Evaluation**
  - *74% activity coverage and 81% state coverage much better than baselines*
  - *Save 20% to 42% exploration steps*

- **Usefulness Evaluation**
  - *20 apps with 30 bugs, 32 testers*
  - *Cover 62% more states*
  - *Cover 61% more activities*
  - *Detect 146% more bugs*
  - *Within 33% less time*

(a) Activity coverage of different methods   (b) State coverage in 30 minutes   (c) Path exploration steps