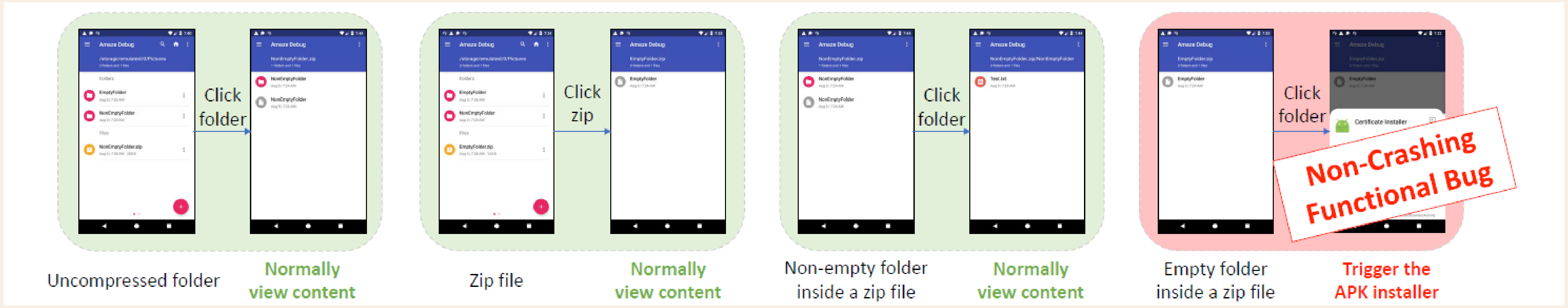
DETECTING NON-CRASHING FUNCTIONAL BUGS IN ANDROID APPS VIA DEEP-STATE DIFFERENTIAL ANALYSIS

Authors:Jue Wang, Yanyan Jiang, Ting Su, Shaohua Li, Chang Xu, Jian Lu, Zhendong Su

PROBLEM STATEMENT

- The paper addresses the challenge of detecting non-crashing functional bugs in Android apps, offering a systematic and automated approach to improve application quality, reliability, and user experience.
- There exists very few automated tools available for detecting functional bugs in Android apps, and the ones that do exist are only capable of identifying certain types of bugs.
- The paper proposes a novel approach called Deep-State Differential Analysis to address this problem. This approach leverages dynamic program analysis and deep learning techniques, the approach aims to detect anomalies or deviations indicative of potential non-crashing functional bugs.


Click folder
Click zip
Click folder
Click folder
Non-Crashing Functional Bug
Uncompressed folder
Normally view content
Zip file
Normally view content
Non-empty folder inside a zip file
Normally view content
Empty folder inside a zip file
Trigger the APK installer

MOTIVATION

- **User Experience Improvement**: Non-crashing functional bugs, while not causing the application to crash, can significantly degrade the user experience. These bugs might lead to unexpected behaviors, incorrect data processing, or deviations from the intended functionality.

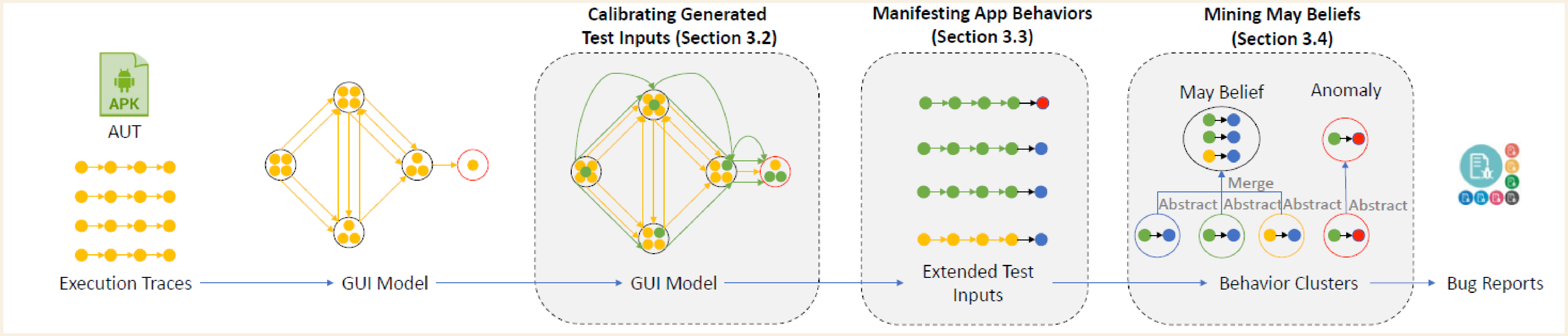
- **Market Competitiveness:** In the highly competitive landscape of mobile applications, ensuring high-quality and bug-free software is essential for success. Applications that suffer from non-crashing functional bugs are more likely to receive negative reviews, lower ratings, and decreased user engagement.

- **Reduced Maintenance Costs:** Addressing functional bugs post-release can be costly and time-consuming for app developers.

- **Advancement of Research in Software Engineering:** The proposed Deep-State Differential Analysis approach represents an innovative contribution to the field of software engineering research.

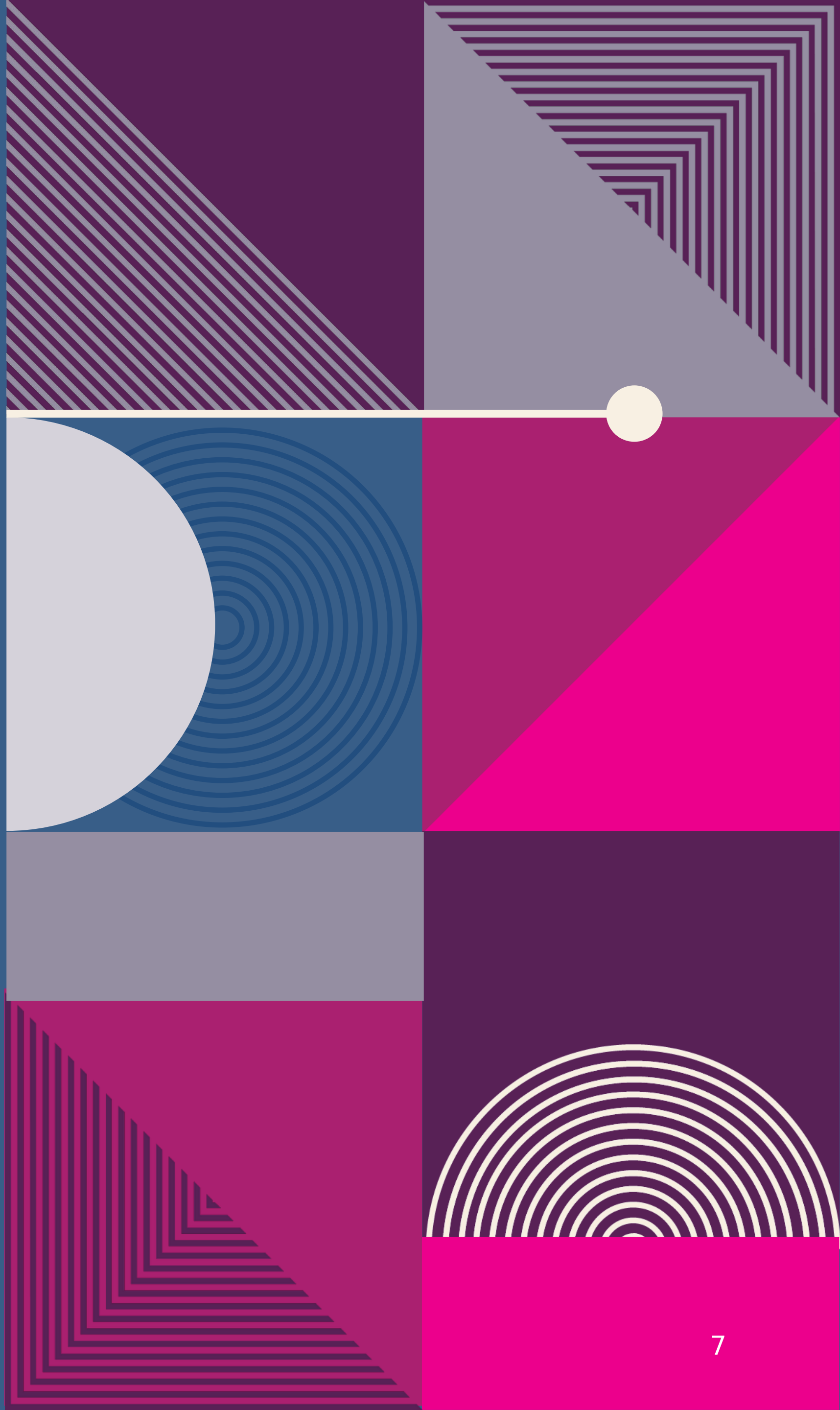
PROPOSED SOLUTION

- The “**Deep-State Differential Analysis**” employs deep learning techniques to generate representations of the "deep states" of Android applications. These deep states encapsulate the underlying behavior and interactions within the application, encompassing various states the app can assume during execution.
- Through differential analysis, the deep-state representations from different versions are compared to identify discrepancies in behavior. These disparities serve as indicators of potential non-crashing functional bugs.
- By scrutinizing the differences in deep states between versions, the solution can identify anomalies or deviations from the anticipated behavior. These anomalies are identified as potential non-crashing functional bugs, signaling the need for further investigation, False positives are refined to enhance the accuracy of bug detection, ensuring that only genuine bugs are flagged.


Calibrating Generated Test Inputs (Section 3.2)
Manifesting App Behaviors (Section 3.3)
Mining May Beliefs (Section 3.4)
APK
AUT
Execution Traces
GUI Model
GUI Model
Extended Test Inputs
May Belief
Anomaly
Merge
Abstract
Abstract
Abstract
Abstract
Behavior Clusters
Bug Reports

RESEARCH QUESTIONS ADDRESSED

- RQ1 : How effective does Odin automatically find non-crashing functional bugs in real Android apps comparing with state-of-the-art techniques?
- RQ2 :How beneficial is input calibration in establishing beliefs and finding noncrashing functional bugs?
- RQ3 :How beneficial is our may-belief mining algorithm in identifying non-crashing functional bugs?
- RQ4 :How precise does Odin report non-crashing functional bugs?
- RQ5 :What types and characteristics of non-crashing functional bugs can Odin find?



EVALUATION METRICS

- Bug Finding
- Test Input Calibration
- May-Belief Mining
- False Positives
- Bug Types

NOVELTY OF PROPOSED SOLUTION

- **Deep-State Representation**: The paper introduces the concept of "deep states" and utilizes deep learning techniques to create representations of these states.
- **Differential Analysis**: The paper proposes a differential analysis approach between deep-state representations of different versions of the same application.
- **Focus on Non-Crashing Functional Bugs**: While existing bug detection methods often prioritize crash detection or security vulnerabilities, the paper specifically targets non-crashing functional bugs.
- **Integration of Deep Learning**: The proposed solution integrates deep learning techniques into the bug detection process.
- **Experimental Validation**: The paper provides experimental validation of the proposed approach using real-world Android applications.

ASSUMPTIONS MADE

- **Consistency of App Behavior Across Versions:** The effectiveness of the Deep-State Differential Analysis approach relies on the assumption that the behavior of an Android application remains relatively consistent across different versions, except for intentional changes or bug fixes.
- **Quality and Accuracy of Deep-State Representations:** The proposed approach assumes that the deep-state representations generated by the deep learning models accurately capture the underlying behavior and interactions within the application.
- **Availability of Training Data and Expertise in Deep Learning:** The paper assumes the availability of sufficient training data consisting of deep-state representations from various Android applications to train the deep learning models effectively.

LIMITATIONS

- **Computational Overhead and Scalability:** Performing deep-state differential analysis on large and complex Android applications can introduce significant computational overhead.
- **Dependency on Training Data and Expertise:** The effectiveness of the proposed approach relies heavily on the availability of sufficient training data consisting of deep-state representations from various Android applications.
- **Detection of False Positives and False Negatives:** Differential analysis between deep-state representations may result in the detection of false positives, where deviations in behavior are not indicative of actual bugs but rather benign changes or differences between app versions.

PRACTICAL SIGNIFICANCE

- **Enhancing User Experience:** By detecting non-crashing functional bugs that may otherwise go unnoticed, the proposed solution helps improve the overall user experience of Android applications.
- **Maintaining App Reliability:** Non-crashing functional bugs, while not causing app crashes, can still significantly impact app reliability and performance.
- **Reducing Maintenance Costs:** Addressing functional bugs post-release can be costly and time-consuming for app developers.
- **Improving Developer Productivity:** The proposed solution automates bug detection processes using deep learning and differential analysis techniques.
- **Enabling Continuous Improvement:** The proposed solution provides insights into the behavior and performance of Android applications across different versions.
- **Supporting Third-party Developers:** Third-party developers often face challenges in ensuring the quality and reliability of their Android applications.
- **Advancing Research in Bug Detection:** The integration of deep learning and differential analysis techniques in bug detection represents an advancement in the field of software engineering research.

DISCUSSION POINT 1

- How do the findings of the study highlight the difficulty in detecting hidden non-crashing functional bugs, even in well-maintained apps with extensive manual test cases? What implications does this have for app developers and the testing process?

DISCUSSION POINT 2

- The paper discusses existing techniques for automatically detecting non-crashing functional bugs, such as Thor, SetDroid, and Genie, which rely on differential-based metamorphic relations. How does Odin's approach differ from these techniques, and what are the potential advantages and limitations of each approach?

DISCUSSION POINT 3

- The authors mention threats to the validity of the evaluation, including the representativeness of selected test subjects and the non-deterministic nature of evaluated techniques. How do the authors address these threats, and what steps are taken to ensure the reliability and accuracy of the evaluation results?

THANK YOU