Final Report

Group Name

BugBugGo

Members

11710208 Tianyuan Xu（徐天元）

11710108 Ting Sun（孙挺）

11711727 Yi Wu（武羿）

Open-source project

Name	Link	Person in charge
kontalk	https://github.com/kontalk/androidclient/	Tianyuan Xu
AntennaPod	https://github.com/AntennaPod/AntennaPod/	Yi Wu
AnkiDroid	https://github.com/ankidroid/Anki-Android/	Ting Sun

Q2. Introduction of all the selected apps

- Kontalk
  - Kontalk is a Real-time Network Communication Software. • It has basic functions such as chatting, group chatting, sending emoji, video, location and so on.
- AntennaPod
  - AntennaPod is a podcast manager and player that gives you access to millions of free and paid podcasts, such as the BBC, NPR and CNN.
- AnkiDroid
  - AnkiDroid is an app that uses flashcards to help people memorize important information.

Q3. Describe the collaboration

- Project proposal

    - Each team member has their own task to choose their own app, but we also communicate with each other to exchange ideas.
    - Each team member has their own task to test their own app, but we also communicate with each other to exchange ideas and discuss testing techniques.
    - Tianyuan Xu
        - App Introduction
        - Time Line
        - Testing Techniques
    - Yi Wu
        - Testing Techniques
        - Experimental Results
    - Ting Sun
        - Conclusions
        - Future Works
- Final Report
    - Tianyuan Xu
        - Names and IDs
        - Introduction
        - Collaboration
        - Compare Different Ways of Modeling Software
        - Tools and Methodology
    - Yi Wu
        - Systematic testing
        - Tools and Methodology
        - Experimental Results
    - Ting Sun
        - Conclusion
        - Future Works
        - After Question

Q4. Compare and contrast different ways of modeling software and how they are related to your project.

IDM

- The input domain of a program is all the possible inputs to that program.

- Interface-based
    - Develops characteristics directly from individual input parameters.
- Functionality-based
    - Develops characteristics from a behavioral view of the program under test.

- We used interface-based and functionality-based IDM to test our apps. However, we did not find any bugs through IDM.

Graph

- The CFG of a method describing all the execution processes of this method.
- We can use Node-Coverage, Edge-Coverage or Edge-pair Coverage.
- Event-level CFG tests user's inputs of GUI.
- We found four bugs using this method because a clear understanding of the overall process is a great way to realize the intention of the programmer.

Logic

- Logic corresponds to the internal structure of the code. And there are two kinds of Logic coverage, Predicate Coverage and Clause Coverage.
- However this method is not very useful for our app testing and it's hard to implement in real programs. We did not find bugs with this method.

Syntactic Expression

- Lots of software artifacts follow strict syntax rules
- The problem of this method is that we are testing the real-world program and it's hard to simplify into several expressions.

Q5. Systematic testing

- **When testing your app, do you think that randomly clicking on the app are more effective than systematic testing using the four ways of modeling software described above?**
  - We think that using systematic testing techniques to find bugs is more effective than randomly clicking. Because these testing techniques will cover most of the cases, and it has a larger probability to find bugs.
  - Moreover, testing techniques can help people localize the bugs.
- **Compute the statistics of number of bugs that your group found through random clicking/systematic testing**
  - The ratio is 0:5. We found five bugs using event graph coverage.

Q6. Tools and Methodology: Describe your experience of using the following approaches:

GitHub classroom versus other code submission system

- It is more convenient to do the group project by the GitHub classroom.
- We can easily discuss our codes and problems with other students in a classroom in the GitHub classroom.

Monkey

- It's a technique in software testing where the user tests the application by providing random inputs and checking the behavior.
- Monkey is a good way to find bugs, but we think it can cost too much time to localize and trace the bugs found.

JUnit

- JUnit is a unit testing framework for the Java programming language. It is very important in software development because writing unit testing by JUnit can find bugs at the very beginning of the developing process where the price of solving bugs is cheapest.
- We think Junit is a basic and important teqchniue to test an app. We use it in our multiple teqchniues, such as IDM, graph coverage, logic coverage.

Regression testing

- Regression testing is the process of testing changes to computer programs to make sure that the older programming still works with the new changes.
- This testing method is usually used in a large project. We do not have the experience of it. However, we think that this method is very important in large companies.

Mutation testing

- We learnt about the mutation testing through an online game called Code Defenders. For the existing code, an attacker needs to create mutants and acquire corresponding score while the defender will write unit tests to kill this mutant and acquire corresponding score. Through this game, we found out that mutation testing can help developers find the weakness in their test data.

Delta Debugging

- Delta Debugging is a methodology to automate the debugging of programs using a scientific approach of hypothesis-trial-result loop.
- We try to use delta debugging to localize the bug found by monkey, but it is indeed a bulky dataset-demand method so we did not succeed. We also used delta debugging for fuzzing test to minimize the random string.

Z3

- Z3 is a state-of-the-art SMT solver from Microsoft Research. It is used to check the satisfiability of logical formulas over one or more theories.
- we learned some basic commands of Z3 and tried it on lab exercise.

Collaborative bug fixing using similar GitHub Issues (MP3)

- Bug sharing is one of the methods that is widely used to find bugs. We find some bugs that can be possibly fixed by referring to similar issues. However, we think collaborative bug fixing is not effective and is time-consuming if we refer to similar GitHub Issues in other similar apps becasue the codes are quite different.

Others approaches covered in class

- We used Monkey runner to record and replay testing procedure, which is helpful for reproducing bugd.

Summarize the statistics of the total bug founds:

App Name	Testing method	Issue name	Issue Link
AntennaPod	Graph Coverage	Mark played/unplayed status should update immediately	https://github.com/AntennaPod/AntennaPod/issues/4776
AntennaPod	Graph Coverage	Playtime and progress bar don't work when play and download at the same time	https://github.com/AntennaPod/AntennaPod/issues/4770
AnkiDriod	Graph Coverage	Crash after "Check database"	https://github.com/ankidroid/Anki-Android/issues/7918
AnkiDriod	Graph Coverage	After switch the language in general setting, the words are not fully translated until the app is reopened	https://github.com/ankidroid/Anki-Android/issues/7921
Kontalk	Fuzzing Test	Can't search messages with special symbols like "\","/"," <",">".	https://github.com/kontalk/androidclient/issues/1310

Q7. Experimental Results

Explain the criteria (e.g., time taken to find bugs) you used to measure the approaches mentioned above and explain how you gathered the data. Choose at least 2 criteria.

- The more possible a method can find a bug, the higher the effectiveness is. The table below is the our statistics.

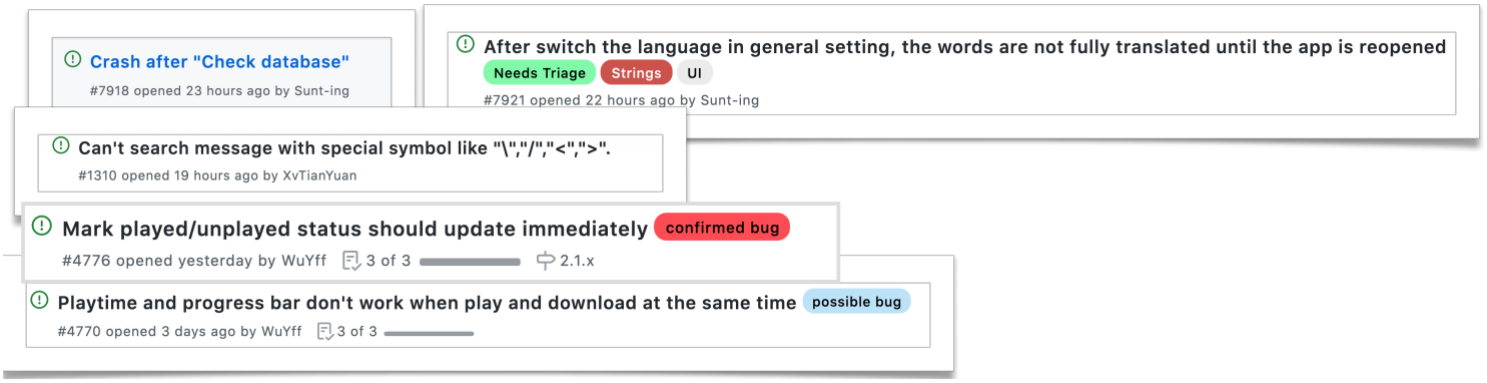
Testing Method	IDM	Graph Coverage	Logic Coverage	Fuzzing Test	Monkey
Bug Number	0	4	0	1	0
Effectiveness	Low	High	Low	Middle	Low

- The more time a method consume, the lower the efficiency. The table below is the result of our evaluation according to our experience of testing.

Testing Method	IDM	Graph Coverage	Logic Coverage	Fuzzing Test	Monkey
Time Efficiecy	Middle	High	Middle	Middle	Low

Present your key results

We found 5 new bugs in total in our 3 Apps and we all reported an issue.



App Name	Testing Method	Breif Description	Reproduciable
AntennaPod	Graph Coverage	The played/unplayed status do not update immediately	Yes
AntennaPod	Graph Coverage	Playtime and progress bar don't work when play and download at the same time	Yes
AnkiDriod	Graph Coverage	After a user clicked “Check database”, it crashed.	Yes

AnkiDriod	Graph Coverage	After switch the language in general setting, the words are not fully translated until the app is reopened	Yes
Kontalk	Fuzzing Test	Can't search messages with special symbols like "\","/"," <",">".	Yes

Discuss your results

Any interesting observations/unexpected results?

- Tianyuan Xu: “When I used monkey to test kontalk, it crashed so seriously that I could not open the app again. And I lost the contact’s messages to reproduce it…..”
- “ Some bugs should be easy to find yet developers did not well test their app and find them..”

Describe the root cause (e.g., memory error, activity lifecycle, concurrency) of the new bugs that you found.

- https://github.com/ankidroid/Anki-Android/issues/7921. This cause of this issue is a hard area and our member is reading code to discuss with the developers.
- Activity lifecycle. Some programs react abnormally to the behavior. The status is not updated concurrently when the button is pressed.

Q8. Conclusions

- Insights can be grasped from others’ works.
  - Bugs that have been found either in the repo matter. Thus we think Collaborative bug finding is useful.
- Some bugs can easily be triggered in some scenarios.
  - For example, `NullPointerException` and `IndexOutOfBoundException` can be easily triggered with double-taps on buttons like Delete or Add.

Q9. Future Work

What lessons did you learn from your project?

- Think Different yourself, or let your tools think differently.
  - Usually, normal paths have been well tested, so we should test differently to find bugs.
  - Monkey and Fuzzing tests are always your friends.
- Do as the teacher teaches, and that's how you find bugs.
  - Graph coverage, monkey testing... You can always find some bugs in these ways.
  - Learning and attention are all you need.

What was difficult?

- Fault localization and Reproduce our bugs is hard when using monkey.

- It require efforts to think and test differently to discover potential bugs as normal paths have been well tested by the developers.

What do you wish you could have done (or done differently)?

- Spend more time on finding bugs.
- Practice the learned methods more on our projects.
- Reading more materials about testing, which allows us to take a large step towards experienced bug hunters.

How could your project be extended...what's next? Are there any interesting problems or questions that resulted from your work?

What's next?

- Help to fix these bugs.
- Extend the found bugs to find more bugs in the project.

Interesting problems or questions

We come up with the following questions:

- How to avoid such bugs in the developing process?
- Are there any more efficient methods for finding bugs?
- Are there any classic bug-prone scenarios in Android apps?

Q1: After listening to the presentation of all teams, answer the following questions based on the decisions of all group members:

Which bugs presented could be possibly reproduced in the app for your group? Or have you learn about new technique/evaluation criteria that will help to improve your report? (2 point)

- https://github.com/moezbhatti/qksms/issues/1674
  This bug can be possibly reproduced in our app Kontalk because Kontalk is also a chatting app . We think empty message issue is common and clicking button twice could usually trigger bugs.

Vote for the best group (you cannot choose your own group). Why do you think that this group deserve to be the best group? (1 point)

- We vote for group 6, because we think this group practiced and applied various testing techqinues effectively. They found 3 bugs and fixed all of them by themselves.