ISSN 2186-7437

NII Shonan Meeting Report

No. 152

Release Engineering for Mobile Applications

Shane McIntosh
Yasutaka Kamei
Meiyappan Nagappan

December 09–12, 2019

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-Ku, Tokyo, Japan

Release Engineering for Mobile Applications

Organizers:
Shane McIntosh (McGill University, Canada)
Yasutaka Kamei (Kyushu University, Japan)
Meiyappan Nagappan (University of Waterloo, Canada)

December 9–12, 2019

1 Overview of the Meeting

1.1 Background

Release engineering is the process that ships code changes from a developer's workspace to the end user. It encompasses tools and technologies to accomplish continuous integration, deployment, delivery, and experimentation, such as build specifications, infrastructure-as-code, containerization, and many more.

In recent years, there has been a renewed interest in this area, driven by the need to deliver new content to users in a continuous fashion. For example, popular web browsers like Mozilla Firefox and Google Chrome have shifted from slow, semi-annual releases to rapid, six-week release cycles. In the domain of web applications, where control over the main delivery process is retained, industry has produced innovative techniques to achieve continuous delivery of new content. These techniques allow for new content to be tentatively released (e.g., canary deployment), seamlessly released (or rolled back) without customer-facing downtime (e.g., blue-green deployment), and analytically evaluated with data from the field (e.g., A/B testing).

At the same time, mobile applications have become a typical means of delivering content to users. For example, recent market studies show that about four billion mobile devices are connected in 2017[1] and predict that the global mobile app economy is expected to be worth $157 billion by 2022.[2]

1.2 Challenges and Opportunities

Rather than delivering content directly through the web browser, mobile app users install applications on their mobile devices (e.g., smartphones, tablets). Release engineers need to deliver new releases from a software organization to app stores that curate mobile apps and make them available to users. The existence of third-party app stores in the release path from software organization to user poses several challenges:

- Users must opt-in to an update of the client-side app in order for new content to be delivered. Releasing too quickly creates an overhead for

[1] https://is.gd/OVPgze
[2] https://is.gd/61uIoy

users, since downloading and installing updates may slow down access to apps. Moreover, with the explosive growth of mobile app sizes, users may need to pay expensive data fees for large downloads on cellular networks.

- App stores act as a bottleneck between software organizations and users, which may impose delays in the release process. For example, to prevent malware and exploit- vulnerable apps from impacting users, all of the apps that appear in the Apple app store need to be certified by an Apple technician. This certification process is slow, cumbersome, and expensive.

1.3 Goals of the Meeting

The purpose of this Shonan meeting is to bring together leading researchers and practitioners from the release engineering and mobile apps fields. The aim is not only to foster an exchange of ideas, but also to outline a clear set of concrete grand challenges and propose a roadmap for how those challenges can be met by future work.

A tangible outcome that we are aiming to produce is a manuscript that describes the grand challenges and our roadmap for future research. For example, a foreseeable grand challenge is how software organizations can achieve crowd-based deployment strategies that have become so popular for web applications (e.g., canary and blue-green deployment) in a mobile setting. The produced manuscript will be submitted as a vision paper to a conference or journal.

2 Meeting Schedule

Check-in Day: December 08 (Sun)

- Welcome Banquet

Day1: December 09 (Mon)

- Plenary: Introduction to the meeting
- Plenary: Introductory lightning talks
- Breakout: Breakout session 1 (Artefacts, Fragmentation, Human Aspects, Log Analysis)
- Plenary: Reports and discussion on breakout session 1

Day2: December 10 (Tue)

- Plenary: Planning for breakout session 2
- Breakout: Breakout session 2 (Definitions)
- Plenary: Reports and discussion on breakout session 2
- Plenary: Planning for breakout session 3
- Breakout: Session 2 (Continuous Experimentation, Quality Assurance, Tools)
- Plenary: Reports and discussion on breakout group 3
- Demo: Julian Hardy

Day3: December 11 (Wed)

- Plenary: Plan for breakout session 4
- Breakout: Session 4 (Seeding Collaborative Projects)
- Group photo shooting
- Excursion and Main Banquet

Day4: December 12 (Thu)

- Plenary: Reports and discussion from breakout session 4
- Breakout: Session 5 (Solidifying Collaboration Plans)
- Plenary: Meeting retrospective & closing

3 Plenary Talks

3.1 Walkthrough of Modern Release Analyics Workbench for Android Apps

Authors. Julian Harty, Commercetest Limited/Open University, UK

The walkthrough was a live demonstration by Julian Harty focusing primarily on Google Play Console including two integrated facilities: 1) Release Engineering and 2) Android Vitals analyics and reporting. We also performed very brief walkthroughs of the continuous build processes for several Android apps.

3.1.1 Google Play Console

Google Play Console is the interface Google provides primarily for developers of Android apps in the Google Play Store. It is intended to provide developers with tools, reports, and insights about how their apps are performing in terms of pupularity, usage, revenues, stability, ratings and reviews. Developers (with appropriate permission) can also manage releases from both a testing and in terms of managing production releases.

Each app belongs to an owner, who will have registered as a developer and paid a one-time fee. Owners can share aspects of their account with other people who have a Google account and who have accepted Google's terms and conditions, etc. The sharing can be granular and limited to a single app owned by a particular owner, the owner can also set the permissions that these other people will have for one or more apps they own. Individuals can have access to multiple accounts and distinct permissions for each app and/or account.

3.1.2 The apps

Julian Harty has been involved in each of the following projects and therefore able to introduce the tools used by each project and aspects of the development, testing, and release processes. In each case there are lead developers for the particular who may be willing to provide further details and insights related to Release Engineering.

- Android Apps from the catrob.at team at TU Graz (source code [https://github.com/catrobat/], project homepage [https://www.catrobat.org/], JIRA [https://jira.catrob.at/], CI [https://jenkins.catrob.at])
- Android Apps from the Kiwix team (source code [https://github.com/kiwix/kiwix-android/], project homepage [https://www.kiwix.org/]). (note: currently migrating from travis-ci - Kiwix on Travis-CI [https://travis-ci.com/github/kiwix/kiwix-android] to GitHub Actions - GitHub Actions for Kiwix [https://github.com/kiwix/kiwix-android/actions]).
- Android Apps from the eduVPN project (source code [https://github.com/eduvpn/android], project homepage [https://eduvpn.nl/]. A proof-of-concept CI is on Travis-CI [https://travis-ci.com/github/commercetest/android]

3.1.3 CI and CB

Characteristics of the Build Process for these apps: Each project team has a distinct build process. Of these, the build process and tooling for the Catrobat apps is the most sophisticated and mature.

Kiwix: The Kiwix Android codebase incorporates a library written in C++ that is shared across many Kiwix projects (including web servers and the iOS app). There are distinct codebases, on GitHub, for the library and for shared build processes and tools. Development builds use pre-built binaries for the native code.

Catrobat apps: The Catrobat apps are built using a farm of Jenkins servers.

eduVPN: The app is independently built by a developer who does not work directly on the development of the app's codebase.

3.1.4 Characteristics of testing for these apps

Kiwix: Until the end of 2019 the project used a combination of Travis-CI and TestDroid to run the automated tests on several hosted physical devices.

Catrobat apps: Automated tests are run by Jenkins.

eduVPN: Testing is ad-hoc and best described via one of the project's 'issues': eduVPN issue 221 [https://github.com/eduvpn/android/issues/221]

3.1.5 Characteristics of the Release Engineering for these apps

Elapsed times

- Kiwix app
- Custom Kiwix apps: WikiMed, ...
- eduVPN: eduVPN, Home edition (not currently in a CB)
- Catrobat: PocketCode, PocketPaint

3.1.6 Glossary

- CB: Continuous Build
- CI: Continuous Integration

4 Breakout Group Discussions

4.1 Session 1: Exploring Topics of Interest

4.1.1 Artefacts for Release Engineering of Mobile Apps

Participants/Authors. Cuiyun Gao, Yasutaka Kamei, Li Li, Shane McIntosh, Sebastian Proksch
Discussion Points. The problems discussed are not specific to mobile app stores, rather than to generic centralized app stores (like Steam Game Platform, MS Windows Appstore, etc.). We think that, to draw the line between an App Store and a regular package manager is the ability of users to review and rate the individual store elements.
Data Sources.

- User facing
  - social media (reddit, blogs, twitter, FB, forums)
- Developer facing
  - App Stores
    - Apple, Google, Steam (commercial)
    - F-Droid (probably not representative), Linux Package Managers ... (open source)
  - app releases (app lineage)
  - commit history
  - StackOverflow Discussion
  - Android Framework Evolution
- In between
  - Logs
  - Crash Reports

Pipeline. Table 4.1.1 shows the artifacts in each phase of release pipeline. The bold phases are the ones that can benefit most from the use of social media. We can use posts as indicators of the release process or to mine new requirements. The other phases are a black box for commercial apps. It is hard to study these, because we lack access to reliable sources.

In general, several research challenges have to be solved, like how to automatically map social media posts and apps.

Table 1: Release Pipeline and artifacts

RelEng Phase	Artifacts
Requirements Engineering	social media, user reviews
Integration	Open Source App Stores (e.g., F-Droid)
Build + Test	Git, CI Providers
Deployment	app lineage
Monitoring + Reaction	logs, social media, user reviews

4.1.2 Overcoming Fragmentation of Android Versions in Deployment of Mobile Apps

Participants/Authors. Carmine Vassallo, Keheliya Gallaba, Raula Gaikovina Kula, Li Li, Daniel Dominguez
Definitions. Push-based deployment model (in case of a web app): Build → Deploy → Release

Pull-based deployment model (in case of mobile apps): A pull request is generated when the app is submitted to the App Store. The App store can refuse/accept it as well as the user.
Challenges.

- For developers
  - Misalignment on Frontend & Backend
  - Traceability of app reviews to an app version (and therefore to a changelog)
  - Traceability between user and developer artifacts
  - Can not force users to update
  - Support legacy software
- For researchers
  - Lack of data for studies. In particular backend information

Research directions.

- Feature toggle updates to emulate canary updates (How to get them right and proper)
- Factors that drive changes and updates to the users (What is a good release note that make people want to update faster?)
- Usage of network logs for approximating the backend behavior Compare the behavior of the webapp to the frontend against the same backend

4.1.3 Human Aspects of Release Engineering for Mobile Apps

Participants/Authors. Tom Zimmermann, Daniel German, Mike Godfrey, Fabio Palomba, Pick Thongtanunam, Kla Tantithamthavorn, Toshiki Hirao, Al-Subaihin, Masanari Kondo
Contents (Figure 4.1.3). Release engineering is not about bug fixes, but it's about social contract / interactions between users and developers. We break users into 2 types, i.e., new users and existing users. Are we in the age of release economics? We try to release apps to get more money ($$$), get more users, and change users' behaviors. For example, we are about to release a new version with bug fixes and updates but don't forget to pay $5. So, we hypothesize that users can change release engineering, and release engineering can change users' behaviors.

- Q1: How do users accept releases (1st release for new users and later for existing users)? is it based on automatically updates?
- Q2: How do developers positively and negatively change the users' behaviors.
- Q3: How do app stores / ecosystems allow dev/users to accept releases?

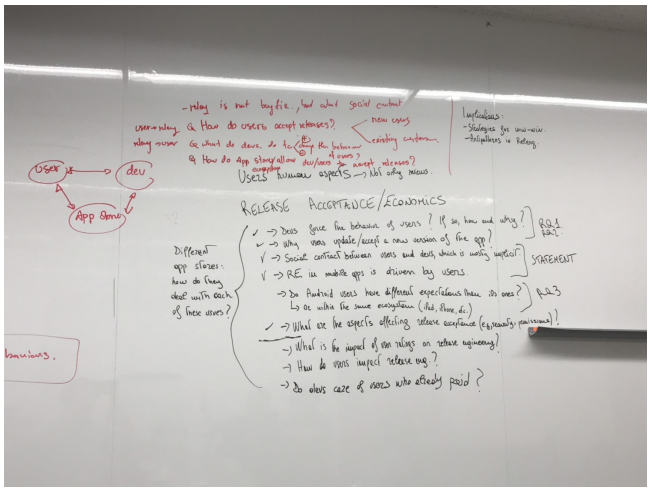

Figure 1: Brainstorming session for human aspects of release engineering for mobile apps

- Implications/Benefits: win-win strategies for both users and developers + anti-patterns in release engineering.
- Other questions:
  - Why mobile apps do not use issue tracking systems? Why do App stores do not provide ITSs for users?
  - Do developers care about bugs?
  - How do users accept releases?
  - Do users care about security? What would users do or take to care about security? What are security-critical apps? (not all apps/users need to care about security)
  - Do different app stores have different users' expectation / users' behaviors? (games have high reviews than educational app reviews)
  - Why do the same apps but in different stores have different price?
  - Do users from different app stores have different expectation/behaviors?

4.1.4 Log Analysis for Mobile Apps

Participants/Authors. Meiyappan Nagappan, Julian Harty, Maurício Aniche, Luca Pascarella, Weiyi Shang
Overview. This document discusses:

- Empirical studies in log engineering that we should do
- (Existing) techniques we need to study and develop
- Challenges faced by researchers in mobile log research
- Existing tools/codebases to support researchers
- Overall concerns in mobile logging

Our focus is on logging in the apps (the front-end) unless otherwise indicated.
Empirical studies in log engineering.

- Why is logging used? Why do mobile devs log? Which logging libraries are used in mobile apps?
  - `https://users.encs.concordia.ca/~shang/pubs/Zeng2019_Article_StudyingTheCharacteristicsOfLo.pdf`
- Performance overhead: how much logging is too much?
  - Signal-to-noise levels in the log messages?
  - How to determine the frequency of the logs being delivered back to the developers.
- What do mobile developers use logs for?
  - e.g., Are logs being used for performance monitoring?
  - Why do developers (even) use logs in apps where they cannot read the logs

Techniques to be studied and developed.

- How much existing techniques should be changed/evolved for mobile apps? Do we need to devise specific tools for mobile?
  - Log analysis research currently focus on Anomaly detection, Security and Privacy, Root cause analysis, Failure prediction, Software testing, Model inference and invariant mining, Reliability and dependability
- How to build different anomaly detection (ML) models for different types of user behavior?
  - In enterprise systems, we have been observing that a single model is not able to capture all the different behaviors of the different companies. We expect the same for mobile apps.
- Can we automatically cluster the different user behaviors?
  - How do we cope with different versions? Maybe Robust Log-Based Anomaly Detection on Unstable Log Data by Zhang et al. might help.
- When do we want to send the logs back to the cloud? And how?
  - Problems: cost, availability/connectivity, energy consumption.
  - Companies like Crashlytics, and Splunk are being by developers.
- How do deal with the inconsistencies and the ever evolving log code base?
  - Robust Log-Based Anomaly Detection on Unstable Log Data. Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, Junjie Chen, Xiaoting He, Randolph Yao, Jian-Guang Lou, Murali Chintalapati, Furao Shen, and Dongmei Zhang
- Can we introduce dynamic logging? I.e., logging only when it's really needed. Saves energy, bandwidth.
  - Can we send a remote command to start logging that area? i.e., only log when it's needed? Remember that you can't deploy your app every second, so dynamically activating logs might be a specific problem of mobile apps.

- Security of the logs: before shipping the logs to the cloud, maybe any app can access those logs. How to store them in a secure way?

Overall concerns.

- Avoid privacy leaks
  - Facebook ad library bug where the facebook token was written to the log, other apps can read
  - GDPR in EU.
    * Who is liable to check if the logging/log files in the apps are compliant?
    * If you use your 'own logging library to collect data', do you have to ask permission for the user? (Note that for logs you get in the app store, directly from Google, you don't need, as Google already asked for it)
  - As a user, do you know what's being logged about you?
- Size of the applications. Mobile apps are often smaller than enterprise traditional projects. Mobile apps focus on single tasks
- Is scalability as big of an issue as in other domains, e.g., enterprise applications?
- Who is responsible to make sure that the log frameworks are good-faith actors who do not have backdoors in the logging.
- How much of the localization of log lines impact the tools and techniques?

Research challenges.

- We need log datasets from mobile applications
  - LogPai benchmark: `https://github.com/logpai`, but it does not consider mobile apps.
- Representative datasets of mobile apps source code
  - FDroid might be not that representative...

Existing tools. Note: ISNIT0 is Joseph Reeve's Github account. Joe works with Julian Harty on creating code to help research logging, mobile analytics.

- `https://github.com/ISNIT0/AndroidCrashDummy` a small exploratory/demo app to test/exercise other aspects of logging and analytics
- `https://github.com/ISNIT0/AndroidLogAssert` a Log Assertion Library for use with Android Espresso tests
- `https://github.com/ISNIT0/logcat-filter` Logcat-filter and analysis tool
- `https://github.com/ISNIT0/log-searcher` A tool for searching Android codebases and analysing usage of "Log.*"
- `https://github.com/ISNIT0/log-complexity-comparison` This tool helps find complex code that does not have much logging to back it up. It was originally built to help with research done by Julian Harty and Joseph Reeve on logging placement. The output is a JSON file and an HTML report, describing which files need more logging attention.

Logging by the Operating System. Google Android includes logging at the device level. Users have the option to opt-in/out to automatically provide usage and diagnostics data to Google [Play]. This logging includes usage, crashes, ANRs and performance issues.

4.2 Session 2: Establishing Working Definitions

In this session, we discuss the same topic about "What Makes Mobile Special for Releng?", but make three small groups to ensure that different voices and opinions are heard.

4.2.1 Group 1. What Makes Mobile Special for Releng?

Participants/Authors. Daniel M. German, Shane McIntosh, Thomas Zimmermann, Li Li, Keheliya Gallaba

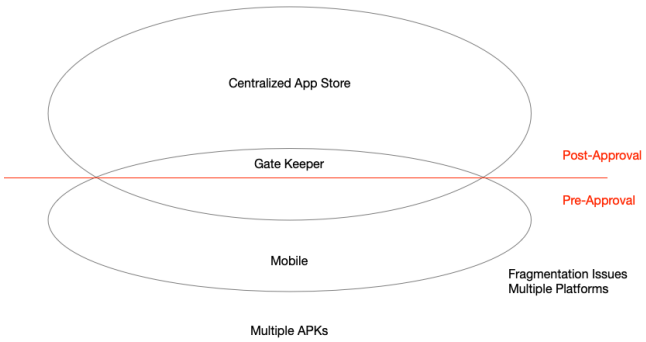

Figure 2: Overview

Discussion (Figure 4.2.1). Lines are blurry about what is a mobile device.

- Is a chromebook with android apps a mobile device?
- Is it a device designed for mobility?
- Is it a device with limited hardware? Even defining what's mobile is a challenging topic.

Then we narrow down on two dimensions on release engineering challenges for mobile.

- **Approval/Post-approval**: Centralized app stores (gate keepers) Not necessarily mobile
- **Pre-approval**: Different mobile device fragmentation based challenges (for development and testing)

Approval/ post-approval challenges.

- Shares challenges with other gate-keeper based eco systems
  - steam game store, atlassian appstore etc
- Not only technical issues. Social/economical/policy challenges

- Multiple gate keepers (company + government)
- Different customer bases
- How to make more money? recurring subscription or one time (different business models)
- How to make users install/updates

Pre-approval challenges.

- Development is done in a different machine from the one it's deployed
- Fragmented by different OSes, hardware, testing challenges (runtime)
- Variants in devices
- Testing for low bandwidth / limited hardware
- **But main challenge: As researchers, we don't really know what's happening here. (we don't have access to this information)**

4.2.2 Group 2. What Makes Mobile Special for Releng?

Participants/Authors. Afnan, Mauricio, Mike, Julian, Toshiki, Raula, Sebastian, Pick, Yasu
History. The mobile developer's guide to the galaxy charts the history of developing mobile apps. The 18th edition is currently online at `https://www.open-xchange.com/resources/mobile-developers-guide-to-the-galaxy/`
Discussion.

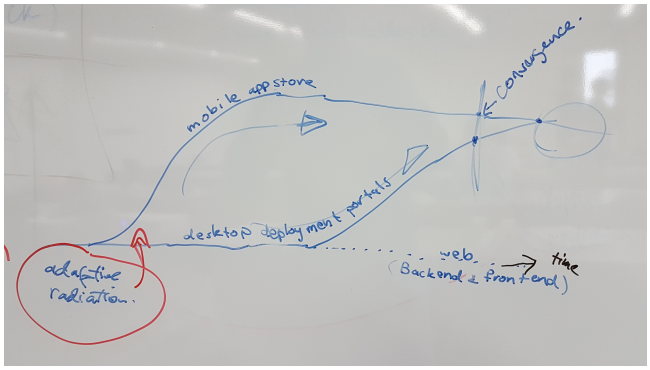

Figure 3: Convergence with Mobile

- Many open questions:
  - Do developers have different technical concerns?
  - Have the stakeholders changed?
  - What is mobile?
    - Is it something with limited hardware, or bandwidth?
    - Or is a laptop a mobile device?
- Clear challenges:
  - Gate keeper
  - Deployment

  - Feedback

- The future is the convergence!
  - Desktops will adopt a lot of the ideas from the mobile world
  - The differences will get smaller and smalle

Next steps. What do researchers need to explore, in order to clarify the similarities/differences among mobile release engineering vs non-mobile release engineering:

- In one dimension:
  - Package manager
  - App store (mobile)
  - App store (non mobile, e.g., Steam)
- In another dimension:
  - Stakeholders (users and devs)
  - Pipeline (control gatekeeper, speed, frequency)
  - Feedback (issue tracking)
  - Economics (revenue/models, market share, exclusivity)

4.2.3 Group 3. What Makes Mobile Special for Releng?

Participants/Authors. Meiyappan Nagappan, Julian Harty, Maurício Aniche, Luca Pascarella, Weiyi Shang
Differences between App Store and Package Manager. We analyzed the main differences between those two providers. In case of App store the focus is on releasing application for general users, while the package manager focuses on releasing libraries for a limited audience (e.g., CS users).

During our discussion we realized that app stores are conceptually an extension of package managers. The main features provided together with the app stores are:

- More direct interaction with the users
- There are several ranking criteria for users (e.g., downloads, scores)
- Mobile Apps are expected to provide revenues
- Libraries packaged together with the mobile app cannot be downloaded
- The approval process is more strict
- Mobile apps require permission from the user
- One point of sale
- Large population and more diverse users

This motivates the need for addressing the following challenges in release engineering for mobile apps.
Challenges for Mobile-app release engineering.

- Design testing strategies for non-functional requirements. Especially in case of small companies, testing for accessibility, usability, and energy consumption might be challenging.

- Setting-up the testing infrastructure is more difficult to the fragmentation
- A/B Testing is more complex
- To address complaints in the reviews there is the need for releasing fast. However, the approval process is quite slow and prevent rapid releases.
- Because of the need for super-fast releases, typicall development activites as code review need to be improved.
- Traceability bugs <-> bad reviews.

4.2.4 Consensus

- Advantages and disadvantages of a slow/fast release process
- Do not know what devs are doing
  - OSS/companies we talk to may not be representative
- Test infrastructure
  - Fast lane → pipelines for app store
- What is the effect of gate keeper?
  - Adv: gets usage data / easy movement of money
  - Disadvantage: mercy of the gate keeper

4.3 Session 3: The State of the Practice

4.3.1 Continuous Experimentation for Mobile

Participants/Authors. Mei, Carmine, Sebastian, Shane, Tom, Raula, Keheliya, Afnan

Continuous Experimentation

- 'whether you built the right thing (validation)'
- *Testing:* whether you built the thing right (verification)

Processes

- A/B Testing
- Canary Releases
- Dark launches

Mechanisms

- Blue/Green
- Feature toggles
- Alpha users

State-of-the-art

- Third-party libraries
- App store (i.e., gatekeepers) support (i.e., location based releases)

Challenges

1. How to do all these mechanisms in Mobile App?
2. Because there is no native support (by gatekeeper), can third-parties or gatekeeper be trusted, and what information needs?
3. Can micro-transactions (by gatekeeper) be used for continuous experimentation?
4. Are the processes mapping reviews+metrics to the deployed version?
5. How to architect for allowing dynamic views?
6. How to effectively use feature toggles with gatekeepers?
7. What are the challenges to collecting telemetry?
8. Continuous experimentation for the losers masses?
9. How to do rollbacks with gatekeeper control?

Other challenges

- Deploying hot fixes

4.3.2 Quality Assurance in the Realm of Release Engineering of Mobile Apps

How does industry define QA?

- Check: basically static analysis
- Testing: more like exercising the app
- Artifacts that need to be tested:
  - Software code
  - Resource files
  - Unit tests
  - Test in platform (by means of an emulator)
  - Many devices

What are quality attributes for mobile?

- Number of bugs, as it was always done?
- Can we use *user rating* or *satisfaction* as quality attribute?
  - Is that what we should care more about?
  - Why similar apps have higher ratings than others?
  - What do an app need to be successful? What does it mean 'to be successful'?
- Performance

- Energy consumption

Overall QA pipeline

1. Static analysis
2. Prediction
3. Code reviews
4. User reviews
5. Automated testing
6. (to be continued)

Static analysis for mobile apps

- Can we use existing tools, like Findbugs for mobile? Yes.
- Specific for Android: *AndroBugs*. Vulnerability scanner, linters.
- Still not enough, lots of false positives. Use of basic/simple rules.
  - This is a general problem in static analysis (not only for mobile)
  - Android's linter has accessibility checks.
- Do we have a tool that statically checks for energy consumption, accessibility, or other non-functional requirements?
- Challenges: obfuscation, multi-language development (e.g., in Android, codebases have Java and C) ¡– maybe a challenge for black box testing?

Open questions and challenges

- What are the characteristics of non-functional bugs?
  - We need patterns/anti-patterns for detecting energy consumption, accessibility, usability, etc.
  - See paper by Luis Cruz et al. (EMSE special issue on mobile apps)
- How well can we develop static analysis to detect non-functional problems?
- How can we design static analysis for resources?
  - See paper by Suelen Goularte et al. (EMSE special issue on mobile apps)
- Can we design tools/techniques that are platform-agnostic, or do they need to be platform-specific?

Prediction

- Do we need defect prediction techniques? They are small files, so maybe you need less prioritization.
  - Maybe we need more defect localization, rather than prediction.

Code reviews

- We can't see differences when it comes to code review in traditional apps.
- RQ: What is different when reviewing mobile apps? Do they actually do it?
  - Maybe related to the non-functional differences we discuss before.
- In a lot of small apps, there's just one developer. Is there a need for code review?
  - Self-review might still be useful. Ex: if you program in VS Code, and review it on Gerrit, the difference in the IDE might help you in seeing things you don't.

Leveraging user reviews

- How to extract bug reports from user reviews?
  - Lots of papers on this topic already
- User reviews are noisy
  - We need NLP approaches to remove noise

Testing

- Challenges:
  - Fragmentation
  - (Environment) dependencies
  - Interactions with other devices
- Tools exist for UI testing, such as Facebook's one
  - Monkey testing is still the best (by Alex Orso)
- From a pragmatic point of view:
  - Apps are made of a lot of UI code, which is hard to unit test.
  - When to do unit testing or when to do system tests for Android?
  - Mocking is a way, but in a UI-heavy application, do we want to mock the UI?
  - RQ: What kind of tests do developers really *do* in practice? What kind of tests they really *need* in practice?
- Can we do (automatic) crash reproduction?
  - Julian: We would be ok if the reproduced crash is not an end-to-end. If it's a "unit test" (one, two, three classes), that'd be already useful for the developer to start working on the issue.

Big questions

- Should we help the 'big app developers' or the 'small app developer' (who are a huge part of the market)?
- Can we learn the best QA practices from the data/developers we have available?
- Given limited resources, what type of tests should we do?

4.4 Session 4 and 5: Seeding Collaborative Projects

Plenty of productive discussions were had during the first three sessions. In the last two sessions, we chose to focus on three topics that will seed future collaborative projects.

4.4.1 Continuous Experimentation with Mobile App Restrictions

Motivation. To mitigate the risk of delivering buggy features to users, developers adopt incremental releasing strategies known as Continuous Experimentation (CE). CE also helps developers to determine if the new features are successful in serving the business goals, refining them if needed. Due to the app-store-centered delivery of mobile applications, CE in the mobile domain presents different challenges and opportunities than other SE disciplines. However, the extent to which CE is adopted in the mobile app development context is not well-understood.

Research Questions.

RQ1 What are the common CE strategies practiced by mobile app developers?

RQ2 How are current tools supporting the practice of CE in mobile?

RQ3 What are the limitations and challenges in the state-of-the-art CE strategies and tools?

Initial Approach. We plan to conduct a mixed method study to understand the state of practice in Continuous Experimentation among mobile developers. First, a developer survey will provide insights on the level of maturity of CE practices, the limitations that developers face and the tooling that developers are using to solve the problems in CE. Based on the results from this qualitative study, we will mine the historical data in the source code repositories of mobile apps to quantify and characterize CE tool usage in practice.

4.4.2 The Ubiquity of the App Store Paradigm (Beyond Mobile Apps)

Motivation. It quickly became clear during the meeting that the App Store paradigm has farther reaching implications than were anticipated prior to the meeting. App stores have permeated a large number of sectors of the software market. Users now expect to interact with app stores as the standard way to acquire software. However, the implications of variations of app stores are not well understood.

Research Questions.

RQ1 What are the emergent patterns in app stores?

RQ2 What are the implications of different app store patterns on stakeholders (e.g., developers, release engineers)?

Initial Approach. We plan to apply qualitative research methods to systematically curate a catalog of patterns of app stores and their perceived implications for stakeholders. This catalog will be a useful resource for more than just addressing our research questions and will be made openly available to the research community.

4.4.3 Centralized Logging for Mobile Apps

Motivation. Logging is one of the main resources of information when software is running by the end users. Similarly, mobile app developers would also benefit from the logging information that provides to understand the quality of their apps, and the behaviour of the software in end users' mobile devices. However, the traditional logging mechanism makes it difficult for developers to collect the information by logs. Recent years, the advances of centralized logging infrastructure, like firebase, has made tremendous support on the applicably of having logging information available for the developers. Yet, the practices and the challenges on the use of these centralized logging infrastructure is not well-understood.

Research Questions.

RQ1 To what extent does centralized logging infrastructure adopted by mobile app developers?

RQ2 What are the information collected by centralized logging infrastructure?

RQ3 What are the limitations and challenges in the state-of-the-art usage of centralized logging on mobile apps?

Approach. We have started to conduct an empirical study on the use of centralized logging infrastructure on open-source projects. In particular, we focus on the use of firebase centralized logging infrastructure and search the entire github for projects that leverage firebase. As a preliminary results, we have found over 100 open source projects that leverage the benefit of centralized logging infrastructure with firebase. By studying the usage of firebase, we find that developers often do not directly use firebase's API to log information. Instead, a customized logging utility class is often developers to support the ease of log for such centralized logging infrastructure. For next steps, we plan to study the rationale of developers adopting centralized logging infrastructure during their development history and making automated tool supports for developers based on our findings.

5 List of Attendees

5.1 Co-organizers

- Shane McIntosh (McGill University, Canada)
- Yasutaka Kamei (Kyushu University, Japan)
- Meiyappan Nagappan (University of Waterloo, Canada)

5.2 Meeting Participants

- Afnan Al-Subaihin (King Saud University, Saudi Arabia)
- Maurício Aniche (Delft University of Technology, Netherlands)
- Daniel Dominguez (IMDEA Software Institute, Spain)
- Keheliya Gallaba (McGill University, Canada)
- Cuiyun Gao (Nanyang Technology University, Singapore)
- Daniel M. German (University of Victoria, Canada)
- Mike Godfrey (University of Waterloo, Canada)
- Julian Harty (Commercetest Limited/Open University, UK)
- Toshiki Hirao (Nara Institute of Science and Technology, Japan)
- Masanari Kondo (Kyoto Institute of Technology, Japan)
- Raula Gaikovina Kula (NAIST, Japan)
- Li Li (Monash University, Australia)
- Fabio Palomba (University of Zurich, Switzerland)
- Luca Pascarella (Delft University of Technology, Netherlands)
- Sebastian Proksch (University of Zurich, Switzerland)
- Weiyi Shang (Concordia University, Canada)
- Chakkrit (Kla) Tantithamthavorn (Monash University, Australia)
- Patanamon (Pick) Thongtanunam (University of Melbourne, Australia)
- Carmine Vassallo (University of Zurich, Switzerland)
- Lili Wei (The Hong Kong University of Science and Technology, China)
- Thomas Zimmermann (Microsoft Research, USA)