Moving Fast with High Reliability: Program Analysis at Uber

Manu Sridharan
Software Reliability Workshop
ETH Zurich

OCTOBER 14, 2017
Uber Apps

Rider

Driver

Eats

iOS and Android
Uber Apps

- Rider
- Driver
- Eats

iOS and Android
Uber cares *a lot* about reliability

Rider crash: can’t get home

Driver crash: can’t earn

Using our apps involves a payment

Apps take significant time to patch
“Transportation as reliable as running water, everywhere, for everyone”
Uber needs to move fast

Hundreds of developers working simultaneously
  • Hundreds of commits per day

Millions of lines of code

Goal: Let builders build
  • Let developers stay in the flow
  • Let developers work independently
How can Uber move fast and keep reliability high?
Modularity in Design and Analysis
Modularity in Design and Analysis

App design emphasizes modularity

• Features can be developed and disabled independently
• Enables developers to move fast
Modularity in Design *and* Analysis

App design emphasizes modularity
- Features can be developed and disabled independently
- Enables developers to move fast

Analysis can both enforce *and* leverage design
- Leverage code modularity for greater scalability / precision
- Modular analysis avoids pollution from unrelated code
Modularity in Design *and* Analysis

App design emphasizes modularity
- Features can be developed and disabled independently
- Enables developers to move fast

Analysis can both enforce *and* leverage design
- Leverage code modularity for greater scalability / precision
- Modular analysis avoids pollution from unrelated code

Strong willingness to adjust design to help analysis
Remainder of this talk
Designing for analyzability

Remainder of this talk
Remainder of this talk

Designing for analyzability

Case studies
Remainder of this talk

Designing for analyzability  Case studies  Future projects / open problems
Designing for Analyzability
In 2016, Uber built new rider apps from scratch.

Key goals included:
- High availability of core flows, without slowing feature development
- Maximal decoupling of features

These goals help static analysis!
Plugins
Plugins: Motivation

Isolate core flows from other features
  • Core flow: getting a ride
  • Non-core: account settings

Plugins can be disabled remotely
  • Reduces risk in feature experimentation
Plugin examples

Profiles

Scheduled rides

Location shortcuts

Feed
Plugins: Under the Hood

Core code defines plugin points
- Changes get extra manual review

Core can only reference non-core via plugins
- Enforced via naming conventions + linting

80% of code in plugins

Core flows tested with all plugins disabled
Deep Scope Hierarchies
Scoping of app state
Scoping of app state

Some state needs to be shared between features

- The map
- User account information when logged in
Scoping of app state

Some state needs to be shared between features
  • The map
  • User account information when logged in

Storing in global state leads to bugs
  • Subtle dependencies creep in between features
Scoping of app state

Some state needs to be shared between features
  • The map
  • User account information when logged in

Storing in global state leads to bugs
  • Subtle dependencies creep in between features

Must manage object lifetimes carefully to avoid leaks
  • E.g., after trip, need to promptly discard trip state
  • In old app, fragile reset() methods
Deep Scope Hierarchies
Deep Scope Hierarchies
Deep Scope Hierarchies
Deep Scope Hierarchies
Deep Scope Hierarchies

Root

LoggedOut

LoggedIn

Request
Deep Scope Hierarchies

Diagram:
- Root
  - LoggedOut
  - Request
  - OnTrip
- LoggedIn
Deep Scope Hierarchies

Parent creates / destroys child scopes
Deep Scope Hierarchies

Parent creates / destroys child scopes

State must be *explicitly shared* from parent to child

*Statically* prohibited from accessing sibling state
Deep Scope Hierarchies

Parent creates / destroys child scopes

State must be *explicitly shared* from parent to child

- *Statically* prohibited from accessing sibling state
Deep Scope Hierarchies

Parent creates / destroys child scopes

State must be *explicitly shared* from parent to child
  - *Statically* prohibited from accessing sibling state

Object lifetimes tied to scopes
  - No more reset() methods
  - Helps prevent leaks
Deep Scope Hierarchies

Parent creates / destroys child scopes

State must be *explicitly shared* from parent to child
- *Static*ally prohibited from accessing sibling state

Object lifetimes tied to scopes
- No more reset() methods
- Helps prevent leaks

Based on new RIB framework
- Router-Interactor-Builder (refinement of MVC)
- Builders manage state sharing, Routers manage tree structure
Implications
Decoupling: Good for Moving Fast

Features stay independent
- A challenge with so much shared app state

Well-defined contracts
- Between features, via RIB tree
- Between core and optional code, via plugins

Result: developers stay sane
Decoupling: Good for Analysis!
Decoupling: Good for Analysis!

Better whole program analysis
  • Less pollution from imprecise data flow
Decoupling: Good for Analysis!

Better whole program analysis
- Less pollution from imprecise data flow

Modular verification!
- Of core code, via plugins
- Of individual features, via deep scopes
Decoupling: Good for Analysis!

Better whole program analysis
  • Less pollution from imprecise data flow

Modular verification!
  • Of core code, via plugins
  • Of individual features, via deep scopes

May require specifications at boundaries
  • Can we infer them?
Decoupling: Good for Analysis!

Better whole program analysis
- Less pollution from imprecise data flow

Modular verification!
- Of core code, via plugins
- Of individual features, via deep scopes

May require specifications at boundaries
- Can we infer them?

Big opportunity!
Nullness Checking for Android
Principles for bug checkers
Principles for bug checkers

Block the build

All reports must be addressed.

Precision is critical
Principles for bug checkers

Block the build
All reports must be addressed.
Precision is critical

Run checks early
Aim for instant IDE feedback.
Performance is critical
Principles for bug checkers

**Block the build**

All reports must be addressed.  
**Precision is critical**

**Run checks early**

Aim for instant IDE feedback.  
**Performance is critical**

**Annotations are OK**

Makes errors more understandable, analyses more performant
NullPointerException Background

Exception in thread "main" java.lang.NullPointerException
    at com.ibm.wala.cast.java.translator.jdt.JDTJava2CAstTranslator.visit(JDTJava2CAstTranslator.java:1480)
    at com.ibm.wala.cast.java.translator.jdt.JDTJava2CAstTranslator.visitNode(JDTJava2CAstTranslator.java:2901)
    at com.ibm.wala.cast.java.translator.jdt.JDTJava2CAstTranslator.visit(JDTJava2CAstTranslator.java:1462)
    at com.ibm.wala.cast.java.translator.jdt.JDTJava2CAstTranslator.visitNode(JDTJava2CAstTranslator.java:2887)
    at com.ibm.wala.cast.java.translator.jdt.JDTJava2CAstTranslator.visitNodeOrNodes(JDTJava2CAstTranslator.java:2959)
    at com.ibm.wala.cast.java.translator.jdt.JDTJava2CAstTranslator.createBlock(JDTJava2CAstTranslator.java:1275)
    at com.ibm.wala.cast.java.translator.jdt.JDTJava2CAstTranslator.visit(JDTJava2CAstTranslator.java:1280)
    ...

Major source of Android app crashes

In mid-2015, Facebook released Infer (http://fbinfer.com/), with static detection of NPEs

Uber aggressively adopted Infer to “eradicate” NPEs
Type-Based NPE Prevention
Type-Based NPE Prevention

```java
static void log(Object x) {
    System.out.println(x.toString());
}
static void foo() {
    log(null);
}
```
Type-Based NPE Prevention

```java
static void log(Object x) {
    System.out.println(x.toString());
}

static void foo() {
    log(null); // Error: cannot pass null to @NonNull parameter x
}
```
Type-Based NPE Prevention

```java
static void log(@Nullable Object x) {
    System.out.println(x.toString());
}
static void foo() {
    log(null);
}
```
Type-Based NPE Prevention

```java
static void log(@Nullable Object x) {
    System.out.println(x.toString());
}

static void foo() {
    log(null);  // Error: de-referencing x may yield NPE
}
```
Type-Based NPE Prevention

```java
static void log(@Nullable Object x) {
    if (x == null) return;
    System.out.println(x.toString());
}
static void foo() {
    log(null);
}
```
Type-Based NPE Prevention

```java
static void log(@Nullable Object x) {
    if (x == null) return;
    System.out.println(x.toString());
}

static void foo() {
    log(null);
}
```
Type-Based NPE Prevention

```java
static void log(@Nullable Object x) {
    if (x == null) return;
    System.out.println(x.toString());
}
static void foo() {
    log(null);
}
```

As in Eradicate (http://fbinfer.com/docs/eradicate.html),
Checker Framework (https://checkerframework.org/)
NullAway

Eradicate: huge success, but significant running time
• Only ran on submit queue (final stage of CI)

Error Prone (http://errorprone.info/): custom checkers within Java compiler

Efficient: Re-use work that compiler has already done
• E.g., AST construction, type analysis

NullAway: Eradicate-like checking in Error Prone
Experience with NullAway

Runs on **all builds** instead of just submit queue
- < 10% overhead
- Devs get much faster feedback

Found hundreds of new issues
- Greater flexibility in handling third-party jars

Open source! [https://github.com/uber/NullAway](https://github.com/uber/NullAway)
Soundness?

Analysis is not sound
  • Not even soundy! (http://soundiness.org)
  • Contrast with Checker Framework

Holes: multithreading, initialization, arrays, mutation, …

In practice, gaps have been in library models

RAVE (https://github.com/uber-common/rave) ensures nullness assumptions valid for data from disk/network
Recent enhancement: stream handling
Recent enhancement: stream handling

class Data {
    int age() { ... };
}
class Person {
    @Nullable Data data() { ... };
}
Stream<Person> pplStream = ...;
int ageSum = pplStream
    .filter((p) -> p.data() != null)
    .mapToInt((p) -> p.data().age())
    .sum();
Recent enhancement: stream handling

class Data { int age() { ... }; }  
class Person { @Nullable Data data() { ... }; }  
Stream<Person> pplStream = ...;  
int ageSum = pplStream  
  .filter((p) -> p.data() != null)  
  .mapToInt((p) -> p.data().age())  
  .sum();

Error: de-referencing p.data() may yield NPE
Recent enhancement: stream handling

class Data { int age() { ... }; }
class Person { @Nullable Data data() { ... }; }
Stream<Person> pplStream = ...;
int ageSum = pplStream
  .filter((p) -> p.data() != null) Result: Stream<{ p | p.data() != null}>
  .mapToInt((p) -> p.data().age())
  .sum();
Recent enhancement: stream handling

class Data { int age() { ... }; }
class Person { @Nullable Data data() { ... }; }
Stream<Person> pplStream = ...;
int ageSum = pplStream
    .filter((p) -> p.data() != null) Result: Stream<{ p | p.data() != null}>
    .mapToInt((p) -> p.data().age())
    .sum();

👍
Multithreading
Multithreading
Multithreading

Many nasty bugs possible

- Data races
- Accessing UI off main thread
Multithreading

Many nasty bugs possible

- Data races
- Accessing UI off main thread

Most multithreading via ReactiveX

- Functional reactive programming for asynchronous streams
- Very structured use of threads
Many nasty bugs possible

- Data races
- Accessing UI off main thread

Most multithreading via ReactiveX

- Functional reactive programming for asynchronous streams
- Very structured use of threads

```java
sourceObservable()
    .observeOn(COMPUTATION)
    .map(x -> expensive(x))
    .observeOn(MAIN)
    .subscribe(i -> display(i));
```
Multithreading

Many nasty bugs possible
  • Data races
  • Accessing UI off main thread

Most multithreading via ReactiveX
  • Functional reactive programming for asynchronous streams
  • Very structured use of threads

Opportunity: specialized analyses!

```java
sourceObservable()
  .observeOn(COMPUTATION)
  .map(x -> expensive(x))
  .observeOn(MAIN)
  .subscribe(i -> display(i));
```
Rx Thread Checker

gsourceObservable()
  .observeOn(COMPUTATION)
  .map(x -> expensive(x))
  .subscribe(i -> display(i));
Rx Thread Checker

sourceObservable()
    .observeOn(COMPUTATION)
    .map(x -> expensive(x))
    .subscribe(i -> display(i));
sourceObservable()
   .observeOn(COMPUTATION)
   .map(x -> expensive(x))
   .subscribe(i -> display(i));

@UIEffect
void display(i) { ... }
sourceObservable()
  .observeOn(COMPUTATION)
  .map(x -> expensive(x))
  .subscribe(i -> display(i));

@UIEffect
void display(i) { ... }
sourceObservable()
    .observeOn(COMPUTATION)
    .map(x -> expensive(x))
    .subscribe(i -> display(i));

@UIEffect
void display(i) { ... }

Enforced with effect analysis (Gordon et al., ECOOP’13)
Rx Thread Checker

```java
sourceObservable()
   .observeOn(COMPUTATION)
   .map(x -> expensive(x))
   .subscribe(i -> display(i));

@UIEffect
void display(i) { ... }
```

Thread types for Observables (Checker Framework)

Enforced with effect analysis (Gordon et al., ECOOP’13)
Rx Thread Checker

sourceObservable() @MainThread
    .observeOn(COMPUTATION)
    .map(x -> expensive(x))
    .subscribe(i -> display(i));

@UIEffect
void display(i) { ... }

Thread types for Observables (Checker Framework)

Enforced with effect analysis (Gordon et al., ECOOP’13)
Rx Thread Checker

sourceObservable() @MainThread
   .observeOn(COMPUTATION) @CompuThread
   .map(x -> expensive(x))
   .subscribe(i -> display(i));

@UIEffect
void display(i) { ... }
 Rx Thread Checker

sourceObservable() @MainThread
  .observeOn(COMPUTATION) @ComputeThread
  .map(x -> expensive(x)) @ComputeThread
  .subscribe(i -> display(i));

@UIEffect
void display(i) { ... }

Enforced with effect analysis (Gordon et al., ECOOP’13)
Rx Thread Checker

sourceObservable() @MainThread
    .observeOn(COMPUTATION) @ComputeThread
    .map(x -> expensive(x)) @ComputeThread
    .subscribe(i -> display(i)); **Error: UI effect off main thread**

@UIEffect
void display(i) { ... }

Thread types for Observables (Checker Framework)

Enforced with effect analysis (Gordon et al., ECOOP’13)
Rx Thread Checker

Thread types for Observables (Checker Framework)

```java
sourceObservable() @MainThread
  .observeOn(COMPUTATION) @ComputeThread
  .map(x -> expensive(x)) @ComputeThread
  .subscribe(i -> display(i)); Error: UI effect off main thread

 Thread types for Observables

@UIEffect
void display(i) { ... }
```

Enforced with effect analysis (Gordon et al., ECOOP’13)

Active deploying across Android codebase

**Future work**: extend to enforce side-effect freedom
Other projects /
open problems
Performance: Speed and Memory

Micromax Spark
Performance: Speed and Memory

Need to run smoothly on all devices

• Many low-end devices in growth markets
Performance: Speed and Memory

Need to run smoothly on *all* devices
- Many low-end devices in growth markets

How to attack with analysis?
- Statically detect slow code on main thread
- Give visibility into UI-blocking network requests
- Reduce OutOfMemoryErrors (statically enforce scope hierarchies)
Performance: Speed and Memory

Need to run smoothly on *all* devices
  - Many low-end devices in growth markets

How to attack with analysis?
  - Statically detect slow code on main thread
  - Give visibility into UI-blocking network requests
  - Reduce OutOfMemoryErrors (statically enforce scope hierarchies)

A challenge with new features going in constantly!

Micromax Spark
Infrastructure: Swift

Future of iOS development

Uber has invested heavily

• New rider app written entirely in Swift
• Google “swift with a hundred engineers”
• Same architecture! Plugins + deep scopes + Rx

Little analysis infrastructure available

• We released NEAL for linting: https://github.com/uber/NEAL

Cross-platform analyses? Swift-specific?
And more…

Verification of startup code

Preventing code duplication

Test generation / selection

Dead code elimination
Conclusions

Need high app reliability

Need developers to move fast

Approach: modular app design + program analysis

• Analysis helps enforce design
• Design increases analysis effectiveness
• Tons of opportunities
Relevant resources

Blog posts
App architecture: https://eng.uber.com/new-rider-app/
Scope hierarchies: https://eng.uber.com/deep-scope-hierarchies/
Plugins: https://eng.uber.com/plugins/

Open source
NullAway: https://github.com/uber/NullAway
NEAL: https://github.com/uber/NEAL
Uber Programming Systems Group

Raj Barik
Lazaro Clapp
Murali Krishna Ramanathan
Manu Sridharan
Adam Welc
Benno Stein (Intern)
Thank you