Migrating Zomato's Mixed Source iOS Codebase to Swift Package Manager

Swift Bengaluru

Inder Deep Singh - July 27, 2024

Inder Deep Singh

SDE II, Zomato

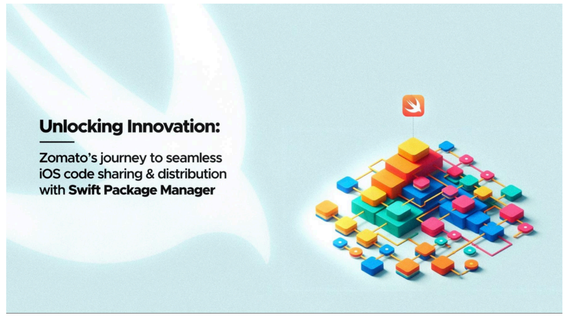
Features:

Zomaland
Zomato Gold
Homepage Banners
Augmented Reality Menu

Articles:

Inder Deep Singh | August 16, 2023 | 11 min read

Building Kimchi for hack-a-noodle 2022 – Zomato's on-the-fly iOS UI rendering engine



Inder Deep Singh | June 6, 2024 | 15 min read

Unlocking Innovation: Zomato's journey to seamless iOS code sharing & distribution with Swift Package Manager

Prelude

Zomato’s iOS Apps

Contrary to common belief…

Why did we need the migration?

Why did we need the migration?

- Our sister apps ran on old copy-pasted code from our app
- There was no central syncing mechanism for the codebases

Project Structure

Internal Kits

Project Structure

- 21 Kits + 1 App Project
- Mostly mixed sources with 20% Objective-C code on average
- Xcode projects for each kit, linked in the main app project
- Hosted across 4 different repos

External Dependencies

Project Structure

- Each internal kit had its own external dependencies
- Each kit had its own cartfile
- Script in the app which would fetch Carthage dependencies for all kits
- Hard to upgrade to newer versions of Xcode
- Had to fork dependencies to make them work for us

Which way to go?

Migrating to Swift Package Manager

First Kit Migration

```
swift package init
```

Move files to the designated folder

First Kit Migration ✅

Integrating the first external dependency

Integrating the first external dependency

Super easy, barely an inconvenience

- First dependency we added was - Alamofire, and adding it was really easy.

```
dependencies: [
  .package(
    url: "https://github.com/Alamofire/Alamofire",
    exact: "5.4.4"
   )
]
```

- All of our external dependencies were supported in SPM

Integrating the first external dependency ✅

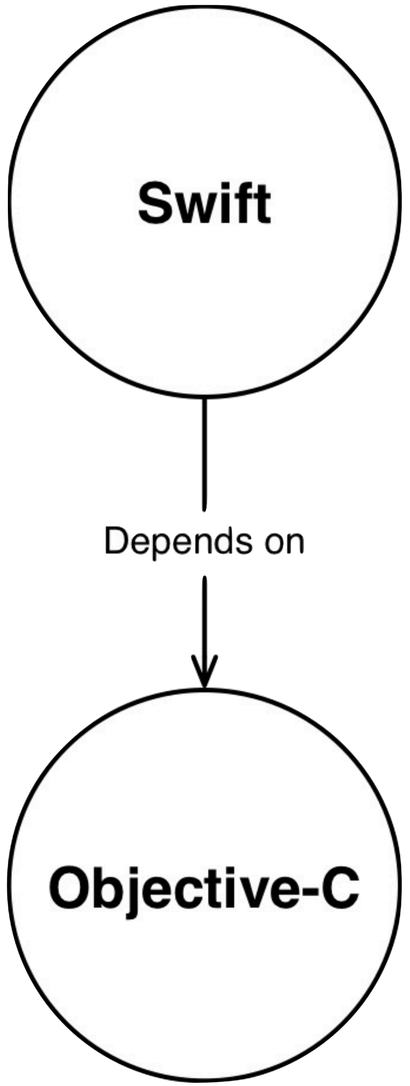
First Roadblock

Creating inter-dependent mixed source swift packages

Creating inter-dependent mixed source swift packages

First Roadblock

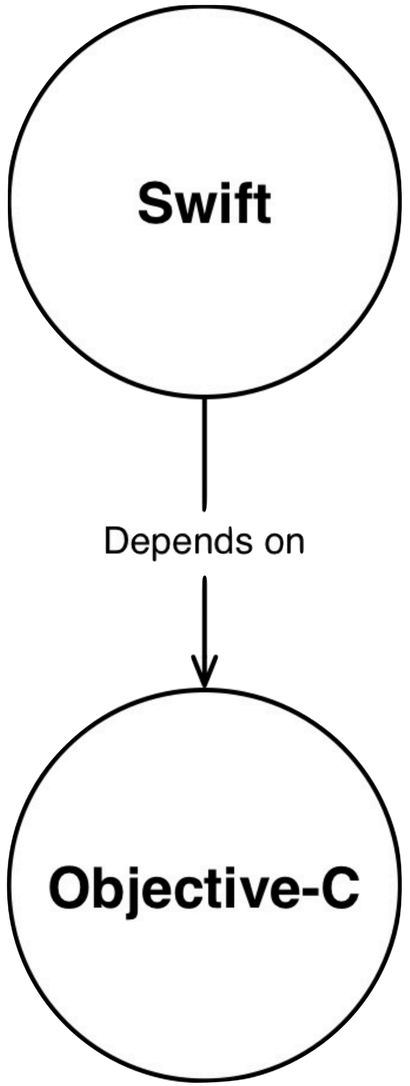
- SPM supports mixed source swift packages like this
- You can only have single language targets
- A target written in one language can depend on targets written in other languages



Creating inter-dependent mixed source swift packages

First Roadblock

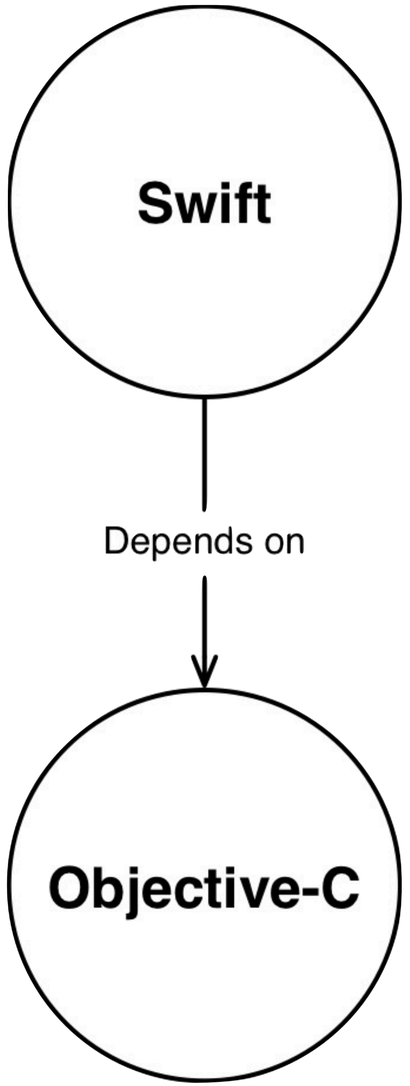
```
.target(
  name: "ApiManagerObjC",
  dependencies: ["Alamofire"],
  path: "ApiManager/ObjC",
  publicHeadersPath: "."
),

.target(
  name: "ApiManager",
  dependencies: ["ZMTApiManagerObjC"],
  path: "ApiManager/Swift"
)
```



Creating inter-dependent mixed source swift packages

First Roadblock

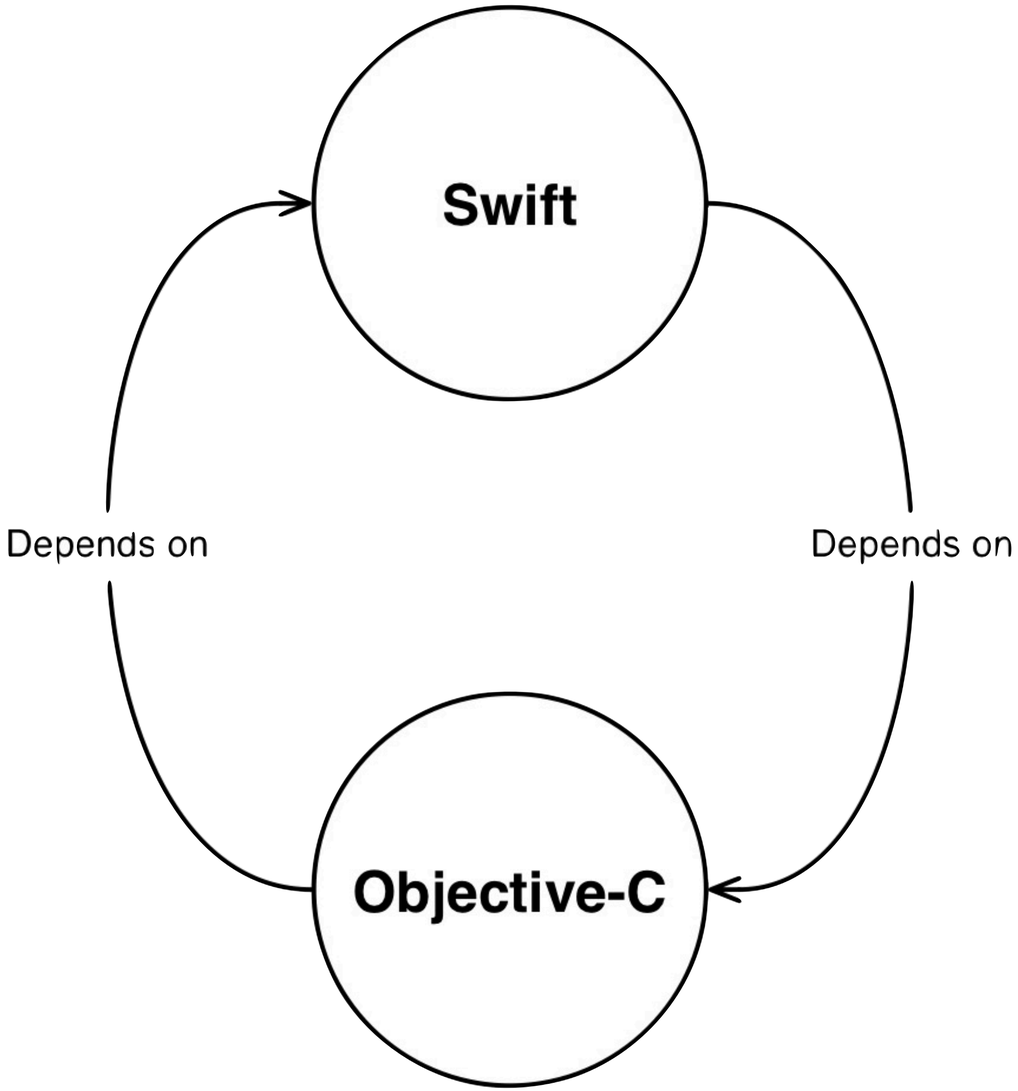
```
.library(
  name: "ApiManager",
  targets: ["ApiManager"]
)
```



Creating inter-dependent mixed source swift packages

First Roadblock

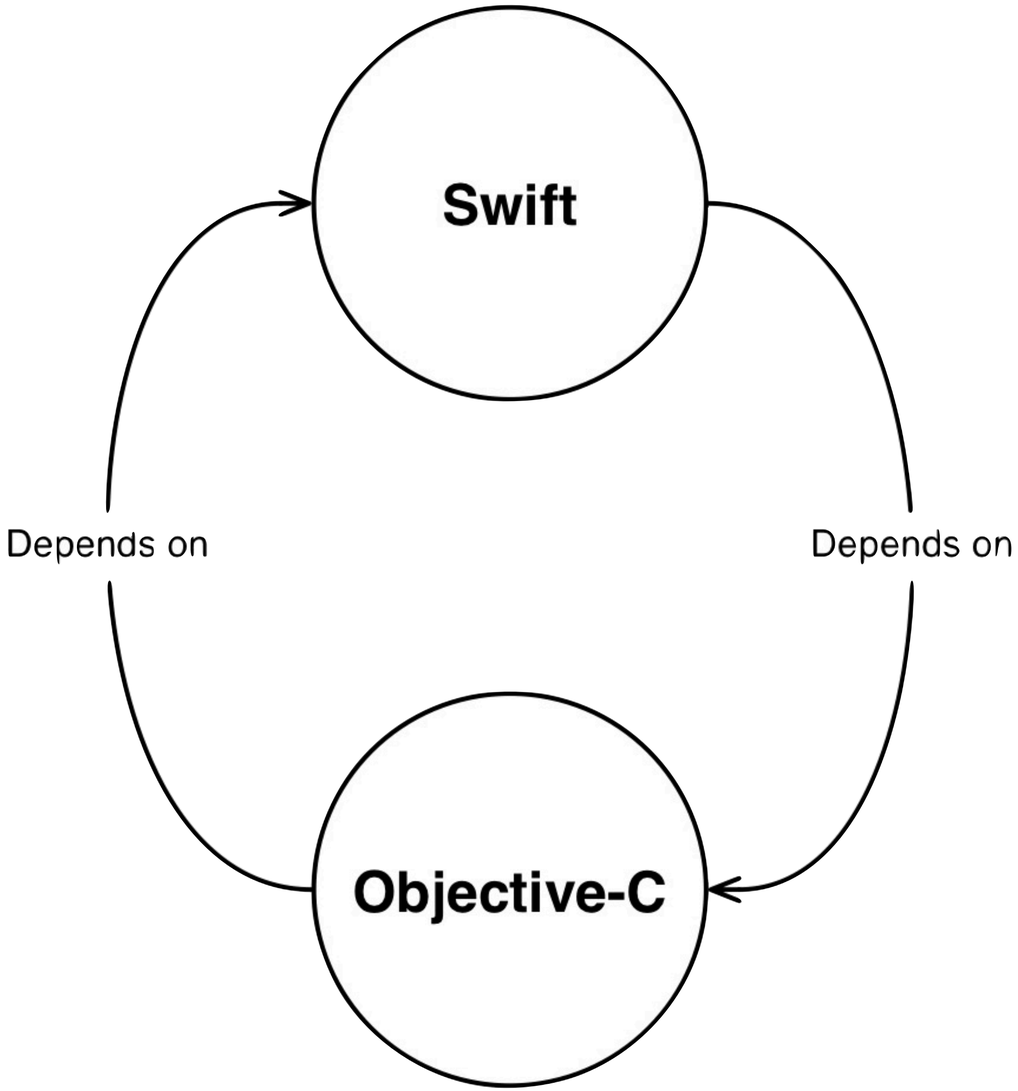
- Most of our kits have cyclical dependencies on Swift & ObjC
- Recommended way is to distribute SPMs with xcframeworks
- Couldn’t find any previously existing solution to our problem



Creating inter-dependent mixed source swift packages

First Roadblock

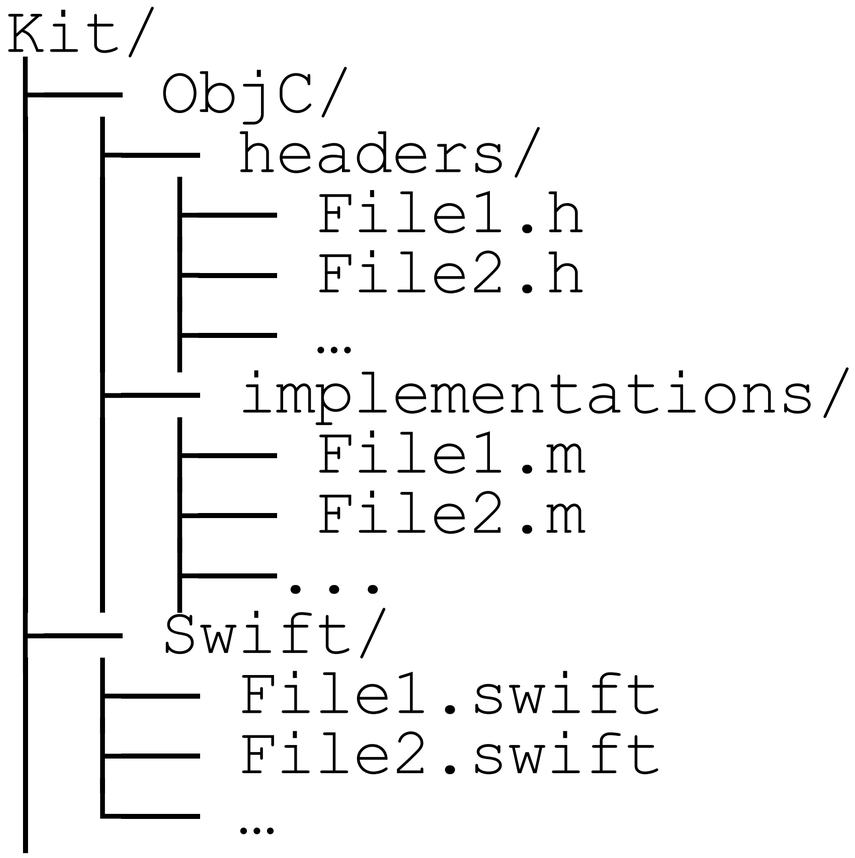
- We looked at how Xcode compiles a inter-dependent mixed source project.
  - Objective-C headers (.h)
  - Swift files
  - Objective-C implementations (.m)



Creating inter-dependent mixed source swift packages

First Roadblock

- Since SPM works with file structures, we had to restructure our kits as follows.
- Root folder contains
  - ObjC folder which contains headers and implementations folder for .h and .m files resepectively
  - Swift folder for swift source files



Creating inter-dependent mixed source swift packages

First Roadblock

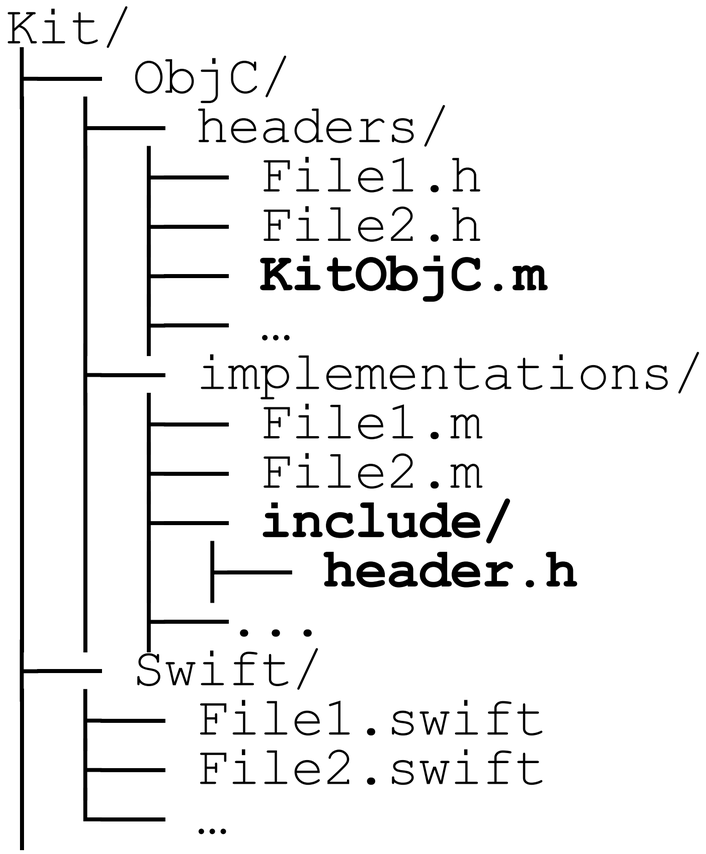
- We also had to update our Package Description
  - First target only has .h files
  - Second target depends on the first target and has only swift files
  - Final target depends on second target and has only .m files

```
.target(
  name: "ApiManagerObjC",
  dependencies: ["Alamofire"],
  path: "ObjC/headers",
  exclude: [“implementations”],
  publicHeadersPath: "."
),

.target(
  name: "ApiManagerSwift",
  dependencies: [“ZMTApiManagerObjC"],
  path: "Swift"
),

.target(
  name: "ApiManager",
  dependencies: ["ZMTApiManagerSwift"],
  path: "ObjC/implementations"
)
```

Creating inter-dependent mixed source swift packages

First Roadblock

- This almost worked, but we got this error-
  “ApiManagerObjc.o” file was not found
- The reason for this error is that an Objective-C target requires at least one .m file to generate a .o file.
- So we updated our file structure as shown.



First Roadblock

Creating inter-dependent mixed source swift packages

Second Roadblock

Resolving Forward Declarations

Resolving Forward Declarations

Second Roadblock

- Some properties/functions defined in Objective-C header files were not accessible by our Swift target.
- The issue was we had declared properties of types defined in Swift in those header files by using forward declaration.

```
@class SwiftObject;
@property (nonatomic, strong)
SwiftObject *property;
- (void)setObject:
(SwiftObject*)object;
```

Resolving Forward Declarations

Second Roadblock

- Forward declarations get resolved by the implementation of the header file
- These properties/functions were accessible as expected in the implementation target
- The issue was only limited to these properties/functions being accessed in the **Swift** target

```
@class SwiftObject;
@property (nonatomic, strong)
SwiftObject *property;
- (void)setObject:
(SwiftObject*)object;
```

Resolving Forward Declarations

Second Roadblock

- We had to remove the breaking forward declarations
- Change forward declared types to first available ObjC type (or NSObject)
- Write extensions to preserve type-safety in Swift
- Typecast in ObjC methods to actual types

LegacyObject.h

```
@property (nonatomic, strong) NSObject
*propertyObj;
- (void)setObject:(NSObject*)object;
```

LegacyObject+.swift

```
import KitObjC;
@objc public extension LegacyObject {
  var property: SwiftObject? {
    get { propertyObj as? SwiftObject }
    set { propertyObj = newValue }
  }
}
```

LegacyObject.m

```
@import KitObjC;
@import KitSwift;
- (void)setObject:(NSObject*)object {
  SwiftObject* obj = (SwiftObject*)object;
  // use obj instead of object
}
```

Second Roadblock

Resolving Forward Declarations

Rinse & repeat the process for all 21 kits

Import the packages in the app

Build & Run

Test, test, test

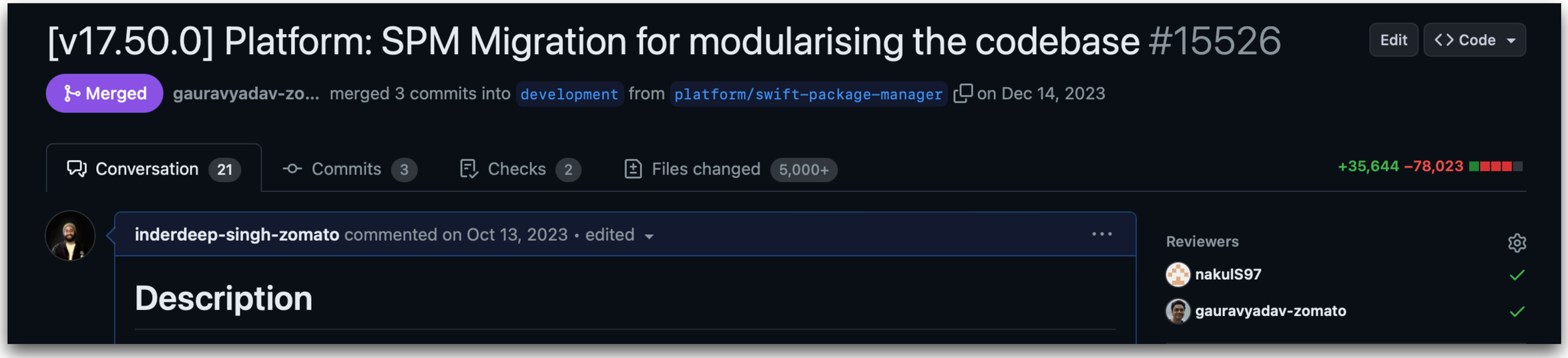
Pre-release hurdles

I can do this all day (week, month 🥹)

- NSKeyedArchiver & NSKeyedUnarchiver crashes
- Issues in retrieving assets from B
- Objective C code not getting indexed in Pods
- App size increased to 5x
- Live Activity crashing
- Syncing with development branch

Merging the Pull Request

GitHub crashed while trying to review



App shipped to customers! 🚀

Starting v17.50.0 in January 2024

0

New crashes reported due to migration

99.99%

Crash-free sessions

(0% change post migration)

Post release

Last few steps to victory

- Migrated all newly created packages to a new **iOS Mono Repo**
- Linked the new repo to app’s repo using Git submodule
- Wrote GitHub Actions to ensure the resiliency of the system
- Migrated all of our sister apps to the new kits

Realisations

Swift Package Manager rocks 🚀

- Upgrading to new versions of Xcode is hassle-free
- Macros massively help ease day-to-day development
- Creating & linking new kits is super-easy!

Dive Deeper

Check out our Apple approved blog post ✅

Thank you!

Gate is open for questions now

Inder Deep Singh

SDE II, Zomato

LinkedIn

Let's Connect