Migrating your legacy code to local SPM packages

an iterative approach

Zouhair Mahieddine
iOS engineer at Medium.com
@Zedenem

Modernizing a codebase

Why?

- To make it more robust
- To make future changes faster and easier to implement
- To facilitate collaboration

When?

- Every chance you get
- In as small iterations as possible
- *While continuously releasing to production*

Benefits of local SPM packages

- Independent modules
- Enforced separation of concerns
- Easy to setup and maintain
- Small initial setup: great for an iterative approach

Acknowledgments

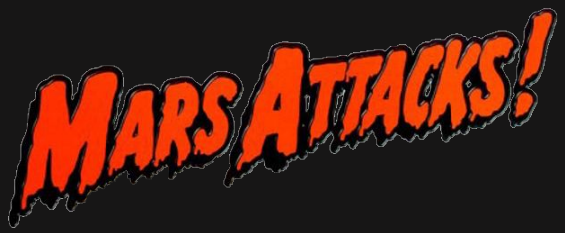
What to expect?

- *some* **slides** (concepts) and *a lot of* **live coding** (practice)
- Follow along: *github/Zedenem/frenchkit* (major steps are tagged)
- Progressively less and less familiar

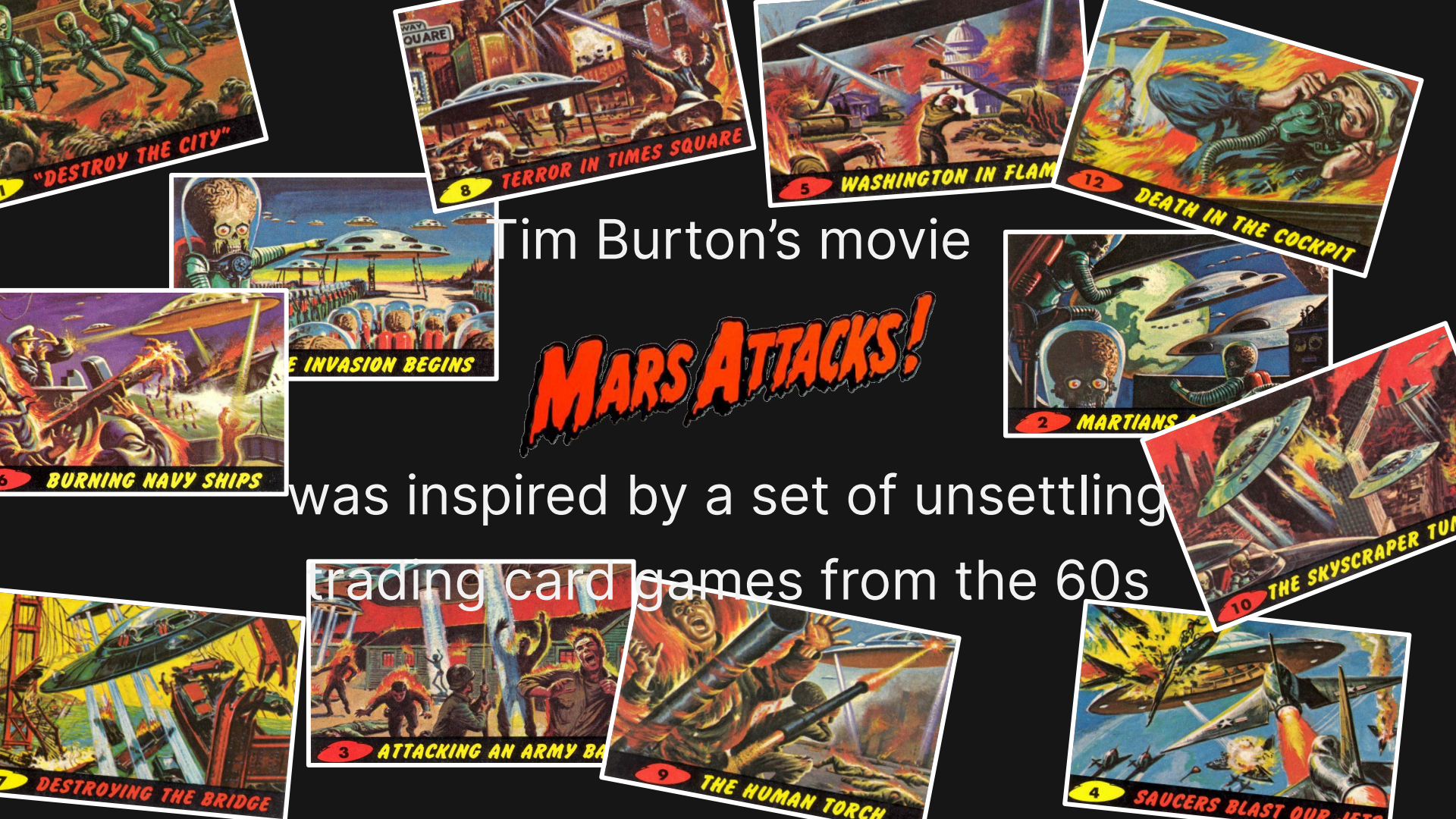
My goal:

each of us will leave this room having learnt something

Tim Burton’s movie



was inspired by a set of unsettling trading card games from the 60s



Tim Burton's movie

MARS ATTACKS!

was inspired by a set of unsettling

trading card games from the 60s

Starting point

Starting point

You just got handed a project

Starting point

You just got handed a project

You have to

- maintain its release cycle
- implement product features and bug fixes

You want to

- reduce tech debt
- modernize the codebase

Let’s have a look

git tag: 00_start_here

Let’s have a look

- Mix of **Objective-C** and **Swift**
- Mix of **SwiftUI** and **UIKit**
- OK unit tests code coverage

`git tag: 00_start_here`

Strategy

1. Initial setup for minimal iterations
2. See your iterations through *(including tests)*
3. Don’t constrain future iterations

Optional rules

- Don’t reduce test coverage
- ABWS: Always Be Writing Swift
- Keep ObjC code isolated

git tag: 00_start_here

Plan

1. Initial local SPM packages setup
    a. Schemes and test targets
    b. Linking
2. First package: Design system (colors)
    a. Isolate code in a package
    b. Use package code in main project
    c. Bridge to ObjC
3. Second package: API Service
    a. Modernizing while modularizing
    b. Async await
    c. Bridge to ObjC

git tag: 00_start_here

1. Initial local SPM packages setup

- Setting up local packages is (almost) one click away
- XCode takes care of most of the linking
- **Bonus:** packages code rely on only two things:
  - File-system structure
  - `Package.swift`

git tag: `00_start_here`

1. Initial local SPM packages setup

Live coding

git tag: `00_start_here`

2. Design system (colors)

- Super easy to add new independent modules
- Directly accessible from Swift code in the main project and in modules
- Accessible from ObjC code *as long as* it conforms to `@objc` requirements → can pollute modules.
- **Strategy:** explicit bridges to ObjC.

2. Design system (colors)

Live coding

git tag: 01_localPackagesSetupDone

3. API service

- Handle dependencies between modules

git tag: 03_beforeAPIModuleIntro

3. API service

- Handle dependencies between modules

```
import PackageDescription

let package = Package(
  name: "LocalPackages",
  platforms: [.iOS(.v14)],
  products: [
    .library(name: "API", targets: ["API"]),
    .library(name: "DesignSystem", targets: ["DesignSystem"]),
    .library(name: "Model", targets: ["Model"]),
  ],
  dependencies: [],
  targets: [
    .target(name: "API", dependencies: ["Model"]),
    .testTarget(name: "APITests", dependencies: ["API"]),
    .target(name: "DesignSystem", dependencies: []),
    .testTarget(name: "DesignSystemTests", dependencies: ["DesignSystem"]),
    .target(name: "Model", dependencies: []),
    .testTarget(name: "ModelTests", dependencies: ["Model"]),
  ]
)
```

git tag: 03_beforeAPIModuleIntro

3. API service

- Handle dependencies between modules

```
import PackageDescription

let package = Package(
  name: "LocalPackages",
  platforms: [.iOS(.v14)],
  products: [
    .library(name: "API", targets: ["API"]),
    .library(name: "DesignSystem", targets: ["DesignSystem"]),
    .library(name: "Model", targets: ["Model"]),
  ],
  dependencies: [],
  targets: [
    .target(name: "API", dependencies: ["Model"]),
    .target(name: "DesignSystem", dependencies: []),
    .testTarget(name: "DesignSystemTests", dependencies: ["DesignSystem"]),
    .target(name: "Model", dependencies: []),
    .testTarget(name: "ModelTests", dependencies: ["Model"]),
  ]
)
```

git tag: 03_beforeAPIModuleIntro

3. API service

- Opportunity to modernize your code (without much effort)
  - Move to `async/await`
  - Focus on a package while modernizing

git tag: 04_APIModuleIntroducedWithBundledResources

3. API service

- Opportunity to modernize your code (without much effort)
  - Move to `async/await`
  - Focus on a package while modernizing

```swift
@objc protocol APIServicing {
  func objc_fetchTopRated(page: Int, completion: @escaping (TopRatedResponse?, NSError?) -> Void)
}

@objc class APIService: NSObject, APIServicing {
  @objc func objc_fetchTopRated(page: Int, completion: @escaping (TopRatedResponse?, NSError?) -> Void) {
    fetchTopRated(page: page) { result in
      switch result {
      case let .success(topRatedResponse): completion(topRatedResponse, nil)
      case let.failure(error): completion(nil, error as NSError)
      }
    }
  }

  func fetchTopRated(page: Int, completion: @escaping (Result<TopRatedResponse, Error>) -> Void) {
    completion(.init {
      guard let url = Bundle.main.url(forResource: "topRated_\(page)", withExtension: "json") else {
        throw APIServiceError.invalidRequest
      }
      let data = try Data(contentsOf: url)
      let decoder = JSONDecoder()
      decoder.keyDecodingStrategy = TopRatedResponse.keyDecodingStrategy
      return try decoder.decode(TopRatedResponse.self, from: data)
    })
  }
}
```

```swift
public protocol APIServicing {
  func fetchTopRated(page: Int) async throws -> TopRatedResponse
}

public class APIService: APIServicing {
  public func fetchTopRated(page: Int) async throws -> TopRatedResponse {
    guard let url = Bundle.module.url(forResource: "topRated_\(page)", withExtension: "json") else {
      throw APIServiceError.invalidRequest
    }
    let data = try Data(contentsOf: url)
    let decoder = JSONDecoder()
    decoder.keyDecodingStrategy = TopRatedResponse.keyDecodingStrategy
    return try decoder.decode(TopRatedResponse.self, from: data)
  }
}
```

git tag: 04_APIModuleIntroducedWithBundledResources

3. API service

- Access bundled files in a package

git tag: 04_APIModuleIntroducedWithBundledResources

3. API service

- Access bundled files in a package

```
.target(name: "API",
        dependencies: ["Model"],
        resources: [
          .process("FakeData/search_cat.json"),
          .process("FakeData/search_dog.json"),
          .process("FakeData/topRated_1.json"),
          .process("FakeData/topRated_2.json"),
          .process("FakeData/topRated_3.json"),
        ]),
```

git tag: 04_APIModuleIntroducedWithBundledResources

3. API service

- Access bundled files in a package

```
.target(name: "API",
        dependencies: ["Model"],
        resources: [
          .process("FakeData/search_cat.json"),
          .process("FakeData/search_dog.json"),
          .process("FakeData/topRated_1.json"),
          .process("FakeData/topRated_2.json"),
          .process("FakeData/topRated_3.json"),
        ]),
```

```
guard let url = Bundle.module.url(forResource: "topRated_\(page)", withExtension: "json") else {
  throw APIServiceError.invalidRequest
}
```

git tag: 04_APIModuleIntroducedWithBundledResources

3. API service

- Access bundled files in a package

```
.target(name: "API",
        dependencies: ["Model"],
        resources: [
          .process("FakeData/search_cat.json"),
          .process("FakeData/search_dog.json"),
          .process("FakeData/topRated_1.json"),
          .process("FakeData/topRated_2.json"),
          .process("FakeData/topRated_3.json"),
        ]),
```

```
guard let urBundle.module.url(urce: "topRated_\(page)", withExtension: "json") else {
  throw APIServiceError.invalidRequest
}
```

git tag: 04_APIModuleIntroducedWithBundledResources

3. API service

- Handle dependencies between modules
- Opportunity to modernize your code (without much effort)
  - Move to `async/await`
  - Focus on a package while modernizing
- Access bundled files in a package
- Expose to ObjC

git tag: 03_beforeAPIModuleIntro

3. API service

Live coding

git tag: 03_beforeAPIModuleIntro

3. API service

- Handle dependencies between modules
- Opportunity to modernize your code (without much effort)
  - Move to `async/await`
  - Focus on a package while modernizing
- Access bundled files in a package
- Expose to ObjC
- A word on cyclic linking issues

git tag: `05_apiModuleDone`

What we learned

1. How to setup local SPM packages with dedicated schemes and tests targets
2. How to isolate code in a package and only expose its public API
3. How to bridge to Objective-C when necessary
4. async/await bridging to Objective-C
5. A strategy for iterative codebase modernization

git tag: 06_CyclicDependenciesQuickfix

Migrating your legacy code to local SPM packages

an iterative approach

Thank you!

Zouhair Mahieddine
iOS engineer at Medium.com
@Zedenem

Migrating your legacy code to local SPM packages

an iterative approach

Q&A

Zouhair Mahieddine
iOS engineer at Medium.com
@Zedenem