앱 안정성 및 확장성 강화를 위한

안드로이드 아키텍처

-테스트 구현-

[Part 2. 테스트 구현 2](#_Toc116369581)

[1. 좋은 아키텍처를 위한 올바른 테스트 2](#_Toc116369582)

[2. Google은 어떻게 테스트하는가 5](#_Toc116369583)

[3. Testing Code Best Practices 6](#_Toc116369584)

[참고 링크 7](#_Toc116369585)

# Part 2. 테스트 구현

## 1. 좋은 아키텍처를 위한 올바른 테스트

1.1 왜 아키텍처에서 테스트가 중요한가

1.1.1 현실적인 필요성

-아키텍처를 변경하는 작업은 많은 코드 수정을 수반

-모든 작업을 QA에서 담당하면 비효율적

1.1.2 좋은 설계를 촉진

-테스트 케이스를 통해서 API를 변경했을 때의 사용성의 차이를 즉시 알 수 있음

1.1.3 코딩 생산성

-테스트가 FAIL 되지 않는 이상 내가 수정한 코드가 잘 움직인다는 심리적 안정성이 코드에 더 집중할 수 있게 해줌

-테스트를 안하면 개발 기간은 짧아지겠지만 언젠가 갚아야하는 부채로 남음. 이를 고려하면 실제로 테스팅 구현을 함께하는 쪽이 개발 시간이 짧음

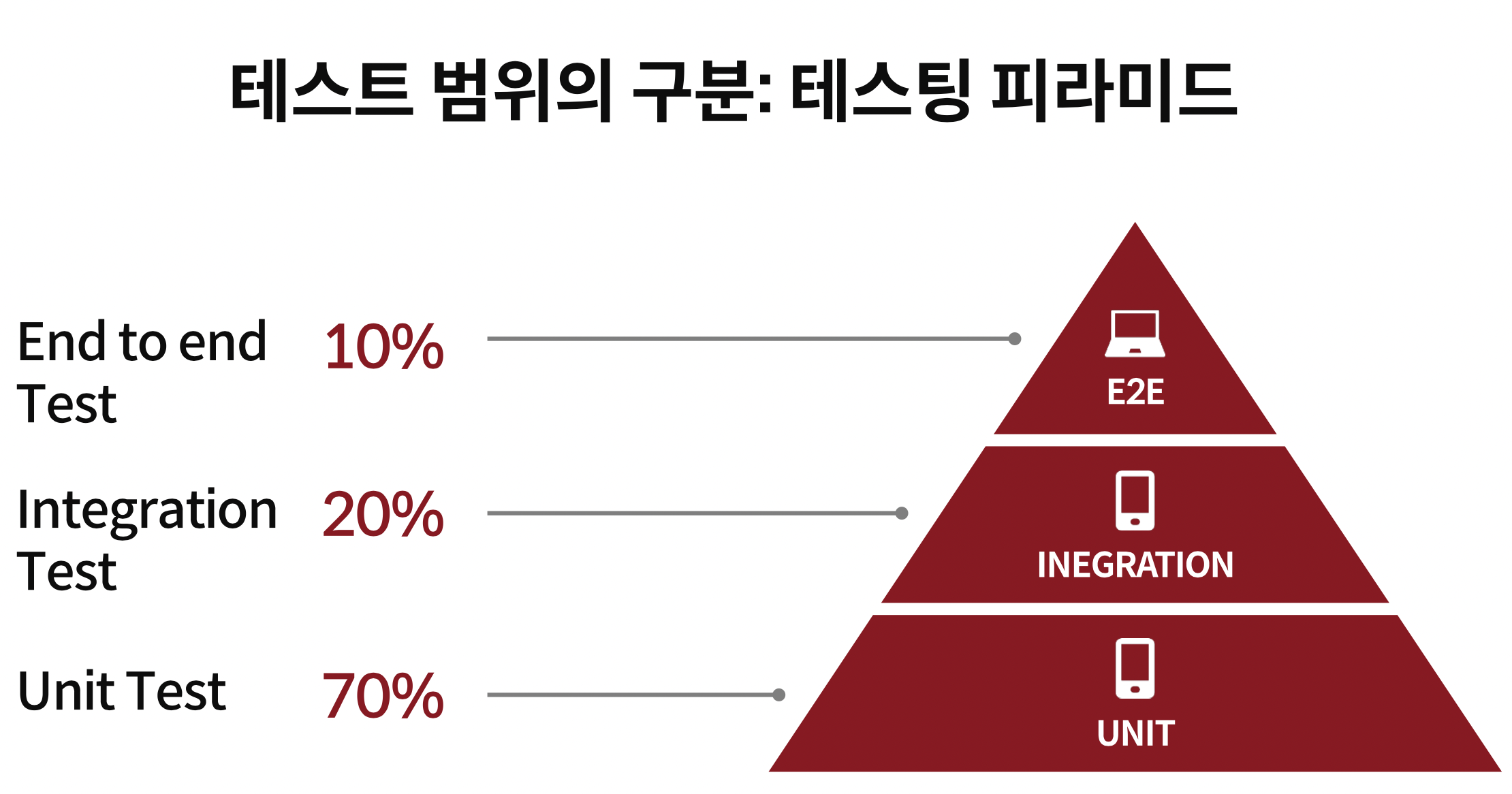
1.1.4 협업을 촉진

-테스트 케이스만 보면 특정 API의 기능과 의도, 올바른 사용법을 단번에 파악할 수 있기 때문에 문서로서의 기능

-코드 담장자가 아니더라도 코드를 수정할 수 있으며, 이 때 원저작자의 의도와 다른 방향으로 구현했다면 테스트도 실패

-더 효율적인 코드 리뷰 가능

1.2 테스트 범위의 구분: 테스팅 피라미드



[그림 2.1 테스팅 피라미드]

1.2.1 E2E

-크롬 브라우저를 띄운 다음, 내가 만든 검색 페이지로 들어가서 검색을 해보고 검색한 내용이 제대로 나오는지 화면상에서 확인하거나 직접 회원가입을 해보고 회원가입 후 로그인 되는지 직접 브라우저 상에 값을 입력해서 테스트 하는 방법

1.2.2 Intergrating test

-최소 두 개 이상의 클래스 또는 서브 시스템의 결합을 테스트하는 방법

1.2.3 Unit test

-코드로 코드를 테스트  
-Unit Testing이 가장 쉬우며 효과가 좋음  
-빠르고 비용이 상대적으로 낮기 때문에 개발할 때 필수적으로 작성해야 함.

ex) 프론트엔드 : jest(가장 많이 사용), enzyme / 백엔드: pytest, unittest(장고에서 기본적으로 사용)

1.2.4 TDD(Test Driven Development)

-테스트 주도 개발을 의미하며, 반복 테스트를 이용한 소프트웨어 방법론으로, 작은 단위의 테스트 케이스를 작성하고 이를 통과하는 코드를 추가하는 단계를 반복하여 구현

1.3 안드로이드 테스트의 종류

1.3.1 Local Test

-시뮬레이터나 디바이스가 아닌 JVM위에서 동작

-Junit 라이브러리 사용

cf. Local Test with androidx.test

-내부적으로 Robolectric 라이브러리(에뮬레이션 된 Context, Application, Activity)를 사용하는 JVM 테스트

-보통은 상당 수 unit test가 여기서 구현

1.3.2 Instrumented Test

-시뮬레이터나 디바이스에서 동작

-Espresso 라이브러리 사용

-안정성/호환성/성능 테스트는 디바이스에서 하는 걸 권장하며, 단위 테스트는 JVM에서 실행해야 함. 그리고 되지 않는 것은 시뮬레이터에서 실시하며, 디바이스에서 테스트를 하기 전에 먼저 이런 것들을 고려해볼 것

1.4. 테스트 해야하는 것

-테스트는 실제 일어날 수 있는 에러를 예방하기 위한 것으로 의미 있는 테스트가 되어야 함

1.4.1 Edge case

-알고리즘이 처리하는 데이터 값이 알고리즘의 특성에 따른 일정한 범위를 넘을 경우에 발생하는 문제 케이스

ex) 분모가 0이되는 데이터를 인위적으로 넣어 테스트 진행

1.4.2 Corner case

-여러 가지 변수와 환경의 복합적인 상호작용으로 발생하는 문제 케이스

ex) 특정 데이터를 입력했을 때 디바이스A에서는 정상동작하지만 디바이스B에서는 오류가 발생하는 경우

같은 장치에서라도 시간이나 다른 환경에 따라 오류가 발생하기도하고 정상작동하기도 하는 경우

1.4.3 그 외

-모든 가능한 네트워크 에러 ex)404

-잘못된 데이터 에러ex) 포맷이 틀린 JSON 문자열

-저장소 오버플로

-중요 객체가 재생성 되는 상황 ex)설정 변경, 다크 모드 적용, 화면 회전에 따른 라이프사이클 초기화 등

1.5. 테스트 코드 구현에 익숙하지 않은 경우

1.5.1 작은, 독립적인 부분부터 시작

-독립 함수(ex. 유틸리티 내부 계산 함수 등)에 대한 JVM 단위 테스트

1.5.2 큰 부분부터 시작

-핵심 시나리오에 대한 E2E 테스트. 단 테스트 요소들을 잘게 쪼갬

ex) 로그인 유즈케이스 -> 1.정상값 입력(->1.1저장 1.2전송 1.3UI갱신 등) 2.비정상값 입력

1.6. 외부 의존성은 어떻게 해결하는가?

-외부 의존성 예시 : SQLite, REST/gRPC call

-1순위: Real Code, 의존성 관계에 있는 진짜 코드를 사용

-2순위: Fake, 라이브러리에 의해 제공되는 표준 fake를 사용

-3순위 Mock/Spy/Stub, 위의 방법이 불가능 할 때

cf. 의존성 주입 라이브러리 Hilt : <https://developer.android.com/training/dependency-injection/hilt-testing>

## 2. Google은 어떻게 테스트하는가

## 3. Testing Code Best Practices

# 참고 링크