시스템 컴포넌트인 테스트

- 아키텍처 관점에선 모든 테스트는 동일
- 테스트
  - 태생적으로 의존성 규칙을 따름
  - 의존성은 항상 테스트 대상이 되는 코드로 향함
  - 테스트는 시스템의 컴포넌트를 향해, 항상 원의 안쪽으로 의존
  - 독립적으로 배포 가능
  - 테스트는 시스템 컴포넌트 중에서 가장 고립되어 있으며, 어떤 사용자나 시스템도 테스트에 의존하지 않는다

테스트를 고려한 설계`

- 테스트는 극단적인 고립성과 배포되지 않는다는 특성 때문에 개발자들이 종종 테스트가 시스템의 설계 범위 밖에 있다고 여기지만, 세트스가 시스템의 설계와 잘 통합되지 않으면 테스트는 깨지기 쉽고, 시스템은 뻣뻣해져 변경하기 어려워짐
- 깨지기 쉬운 테스트 문제(fragile tests problem)
  - 시스템의 공통 컴포넌트가 변경되면, 수백, 수천개의 테스트가 망가질 수 있다
  - 시스템을 뻣뻣하게 만든다는 부작용
  - 이 문제를 해결하기 위해 테스트를 고려해 설계해야한다
    - **변동성이 있는 것에 의존하지 말라**

테스트 API

- 테스트 API
  - 테스트가 모든 업무 규칙을 검증하는 데 사용할 수 있도록 특화된 API를 생성하기
  - (DB와 같은) 값비싼 자원은 건너뛰고, 시스템을 테스트 가능한 특정 상태로 강제하는 강력한 힘들 지녀야함
  - 사용자 인터페이스가 사용하는 interactor & 인터페이스 어댑터(interface adapter)들의 상위 집합이 되어야함
  - 테스트를 애플리케이션으로부터 분리할 목적으로 사용

구조적 결합

- 구조적 결합은 테스트 결합 중에서 가장 강하며, 가장 은밀하게 퍼져 가는 유형
- 시간이 지날수록 테스트는 계속해서 더 구체적이고 더 특화된 형태로 변화되고, 반대로 상용 코드는 더 추상적이고 더 범용적인 형태로 변화

- 예시
  - 모든 상용 클래스에 테스트 클래스가 각각 존재
  - 모든 상용 메서드에 테스트 메서드 집합이 각각 존재하는 경우
  - **상용 클래스나 메서드 중 하나라도 변경되면 딸려 있는 다수의 테스트가 변경되어야 함**
    - 깨지기 쉬운 테스트가 됨

보안

- 테스트 API가 지닌 강력한 힘을 운영 시스템에 배포하면 위험에 처할 수 있다
- 위험을 피하기 위해 테스트 API 자체와 테스트 API 중 위험한 부분의 구현부는 독립적으로 배포할 수 있는 컴포넌트로 분리해야함