A review of software testing

- P.David Coward

컴퓨터 공학부

200412324

박동일

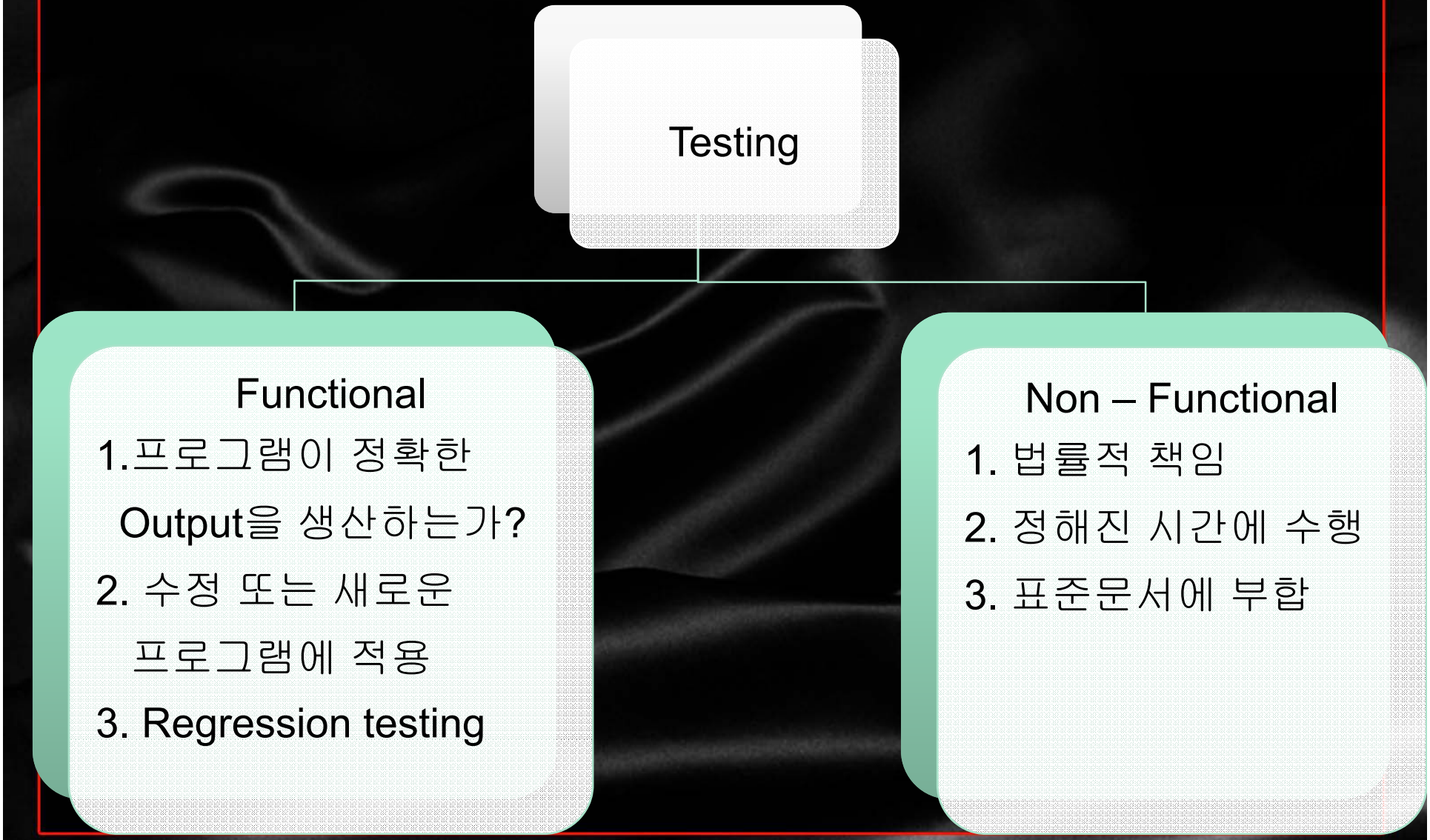
Contents

1. What is Testing
2. Two Types Of The Testing
3. Views to Testing
4. Testing Strategies
5. Static vs. Dynamic analysis
6. Various Testing Techniques

What is Testing ?

- 프로그램의 품질과 신뢰성을 향상시키기 위해 오류를 찾는 과정.
- 소프트웨어가 요구사항에 일치하는가?
  - **Specification** 정확하지 않다 .
- 적절하게 수행되고 있는가?
- 얼마나 진행 되었는가?
- **Output from test vs. Expected Output**

Two Types of The Testing



Views to Testing

Aims of Testing

Destructive Process

1. faults를 찾는다.
2. Destructive Process 어려움
   소프트웨어는 프로그램을
   확장된 자아로 여긴다.

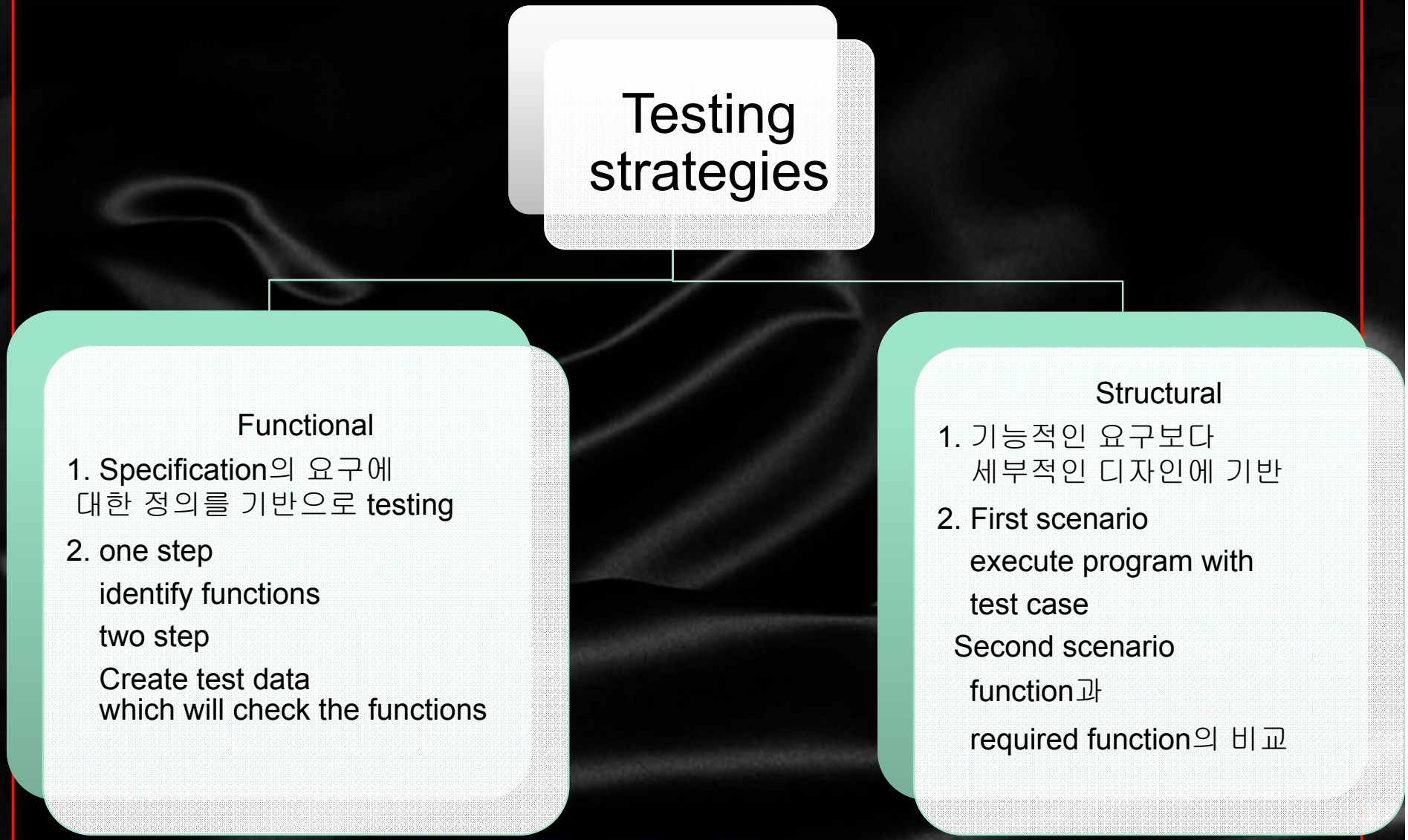
Destructive Process 어려움

Sol) validator 와 creator의
분리

Constructive

1. faults가 없다는 것 을
   증명.
2. Gentle of Testing
3. faults 를 놓칠 수 있다.

Testing Strategies



Functional Testing

- 프로그램이 어떻게 기능을 사용하는지 신경 쓰지 않는다.
  - **Black Box Testing**
  - 외부에서 그 기능, 성분 등을 테스트
- 체계적인 디자인 문서화로부터 기능과 데이터의 식별을 위해 규칙이 구성된다.
- 이 규칙은 오류에 대하여 정확히 설명할 수 없다.
- 테스터는 프로그램의 기능적 이해를 바탕으로 **test case**를 제안한다.
- **Oracle**은 각각의 **test case**에 어떤 결과 값이 나올지 정확하게 예측할 수 있는 프로그램이다.

Structural Testing

- **White Box testing**
  - 프로그램의 모든 논리적 경로를 파악하거나 경로상의 복잡도 계산
- 기능적 요구보다 세부적인 디자인에 기반을 둔다.
- **2 scenario**
  - **1. Test case** 와 함께 프로그램을 실행한다.
  - **2.** 프로그램의 기능과 기능적 요구의 일치를 비교한다.
- **Structural testing**은 프로그램의 실행을 포함하고 특별한 **coverage**가 필요하다.
  - **1.** 문장이 한 번이라도 수행되도록 한다.
  - **2.** 프로그램의 모든 경로가 한번이라도 수행되도록 한다.
  - **3.** 논리적 경로가 최소한 한 번이라도 수행되도록 설계한다.

Structural Testing Cont.

- **Problems**
  - **1.** 가능한 경로가 많다.
  - **2.** 실행 불가능한 경로가 존재한다.

Static vs. Dynamic analysis

- **Static**
  - 프로그램을 실행하지 않고 오류를 찾아내는 **test** 방법.
- **Dynamic analysis**
  - 직접 실행 시켜보면서 소프트웨어가 예상 대로 작용하는지 판정
  - **functional** 과 **structural testing**의 다리역할.
  - 처음엔 **functional testing**이 **test case**를 말한다.
    이런 **test case**들의 실행은 **Dynamic analysis**에 의해 감시된다.

Classification of Testing Techniques

	Structural	Functional
Static	Symbolic execution Program proving Anomaly analysis	
Dynamic	Computation testing Domain testing Automatic path-based test Data generation Mutation analysis	Random testing Domain testing Cause- effect Graph Adaptive perturbation testing

Various Testing Techniques

- **Static –Structural**
  - 소프트웨어로부터 만들어진 프로그램을 실제로 실행해보지 않고 분석하는 방법.

- **Symbolic execution**
  - 프로그램을 실제로 실행하지 않는 **testing** 방법
  - 데이터 값은 **symbolic value**로 대체.
  - 출력변수당 하나의 표현을 만든다.
  - **Testing Data**와 **Program Proving**의 중간단계이다.
  - **loops handling**이 가장 어렵다.
  - 가장 일반적인 방법은 **Flow –graph**를 만드는 것이다.

Various Testing Techniques cont.

- **Partition Analysis**
  - **Input data domain**의 서브 도메인을 구별하기 위해 **symbolic execution**을 사용한다.
  - **Input data domain**의 어떤 부분이 구조적이나 기능적 오류가 발견되면 서브도메인에 할당되지 않는다.
  - 이때 **structural** 또는 **Functional(Program or specification)** 의 오류를 발견한다.

Various Testing Techniques cont.

- **Program proving**
  - 일반적인 방법은 **'Inductive Assertion Verification'**
  - 이 방법에서 **Assertion**은 처음과 끝에 놓여있다.
  - 각각의 **Assertion**은 수학적으로 절차들의 기능을 설명
  - **input assertion**이 참이고 **output assertion**도 참이라 확신되면 그 때 주장은 참이라고 한다.
- 방 법
  - 프로그램을 만들어라.
  - 처음과 끝에 수학적인 **assertion**을 넣어라.
  - 시작에서 주어진 **end assertion**을 얻을 수 있는지 결정하라.
  - 시작점에서 주어진 **assertion**과 결과가 같다면 정확하다.

Various Testing Techniques cont.

- **Anomaly Analysis**
  - 1단계 : 컴파일러에 의한 문법적인 체크
  - 2단계 : 프로그램언어로써 변칙적인 것을 찾는다.
    - **Island code**의 존재는 실행될 수 없는 코드
    - 배열과 관련된 문제
    - 변수초기화의 실패
    - 사용 불가능한 변수와 라벨
    - 루프의 안,밖으로의 점프

Dynamic Functional Test Cases

- **Dynamic-Functional**
  - **Test case** 를 실행한다.
  - 소프트웨어의 디자인은 고려하지 않는다.

- **Domain- testing**
  - 요구사항의 비공식적인 분류에 바탕을 둔 테스트
  - 데이터 또는 **function**은 도메인 파티션의 기초를 제공
  - 테스트 케이스를 실행하고, **expectation**과의 비교를 통해 에러를 찾는다.

Dynamic Functional Test Cases Cont.

- **Random testing**
  - 코드나 **specification**의 참조 없이 데이터 값 생산
  - **Random number generation**이 사용됨
  - 프로그램의 전체 영역을 보장할 수 없다.
  - 작은 부분만을 조사하는 것이 핵심이다.

Dynamic Functional Test Cases Cont.

- **Adaptive Perturbation testing**
  - **test case**의 효율성을 평가하는데 기반.
  - **test case**의 효율성을 증가시키는 목적으로 사용.
  - **S/W**안에 개발자가 넣은 실행가능 한 **assertion** 사용
    assertion : statement about the reasonableness of values of variables.
  - **violated assertions**이 더 이상 증가 하지 않을때 까지 진행된다
  - 루틴이 최적화 값을 찾아 버려진 값을 교체하는 수의 **assertion**이 최대가 되면 최적화 된 것이다.

Dynamic Functional Test Cases Cont.

- **Cause – Effect Graphing**
  - 출력은 연산으로 사용된 입력 값의 조합으로부터 온다.**(AND, OR)**
  - 적은 수의 **testing case**에 유용함

  1) 5단계
    - **specification**을 일할수 있게 나눠라.
    - 원인과 결과를 명확히 하라.
    - 원인과 결과의 의미를 그래프로 나타내라.
    - 불가능한 조합과 결과를 보여주기 위해 주석을 써라.
    - 그래프를 제한적인 입력의 **decision table** 로 바꿔라.

Dynamic Structural Test Cases

- **Dynamic - Structural**
  - 소프트웨어는 **test case** 안에서 실행된다
  - **test case**는 소프트웨어 분석에 의해 만들어진다.

- **Domain and Computational Testing**
  - **Test case**의 선택에 기반을 두고 있다.
  - **Assignment statements**는 **Computation paths**로 사용된다.

Dynamic Structural Test Cases Cont.

- **Domain testing**
  - **Domain testing**은 **test data**를 선택하고 각각의 도메인 경로의 **boundary**에 가까이 간다.
  - 케이스가 잘 못된 경로를 따라 갔을 때의 에러는 **domain error**
- **Computation Testing**
  - 계산에러의 발견에 초점을 둔다.
  - 정확한 경로를 따라 갔을 때 결과의 에러는 **Computation error**

Dynamic Structural Test Cases Cont.

- **Paths considered**
  - **Path computation**
    - 수적인 표현으로 구성된다.
    - 각각의 **output**변수는 특별한 경로의 **input**변수와 상수로 환산된다.
  - **Path condition**
    - 제한된 경로의 결합이다.

Dynamic Structural Test Cases Cont.

- **Path domain**
  - **path condition**을 만족하는 **input** 변수로 구성된다.
- **Empty path**
  - 실행 불가능 하고 실행 할 수 없다.

Dynamic Structural Test Cases Cont.

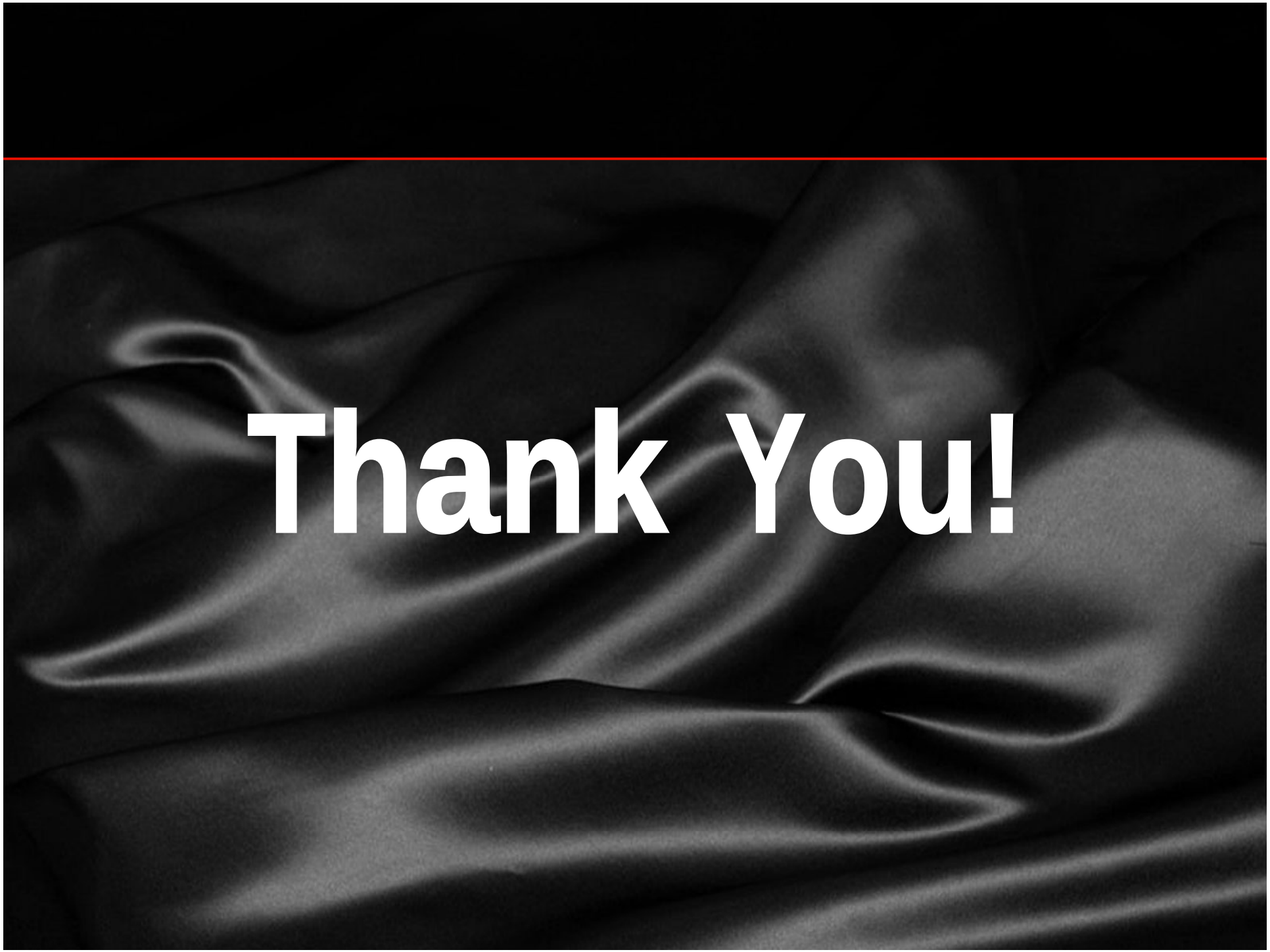
- **Automatic test data generation**
  - **Path generation** 과 **predicate solving path**를 연속적으로 사용.
    이것은 **High coverage**에서 테스트케이스 생성
  - **High coverage**의 **program**은 **specification**과 일치 하지 않음.
    기능적인 부분의 누락이 있을 수 있다.
  - 이 **testing**가 목적을 달성하기 위해서는 **formal specification** 필요.

Dynamic Structural Test Cases Cont.

- **Mutation Analysis**
  - 질적인 **test data**와 관련있다.
  - 프로그램은 **test data**를 **test** 하기 위해 사용.
  - **Original** 프로그램 과 **mutant** 프로그램은 같은 **test data**를 사용한다.
    - **Output**이 다르다면 **mutant**는 **dead.** **(**사용 되지 않는다**.)**
    - **Output**이 같다면 문제가 변화된다. **(mutant**는 **live** 라고 함**)**
  - **Dead** 와 **live**의 비율을 기준으로 측정.

Summary

- **Testing**의 목적은 소프트웨어의 신뢰를 얻는 것이다.
- 많은 **Testing**기법이 있다.
  - 목적에 맞게 사용
- **Test case**가 에러를 찾지 못하면 정확하다고 할 수 있다
- 넓은 범위의 **Testing techniques**의 사용은 정확한 소프트웨어를 만든다.


Thank You!