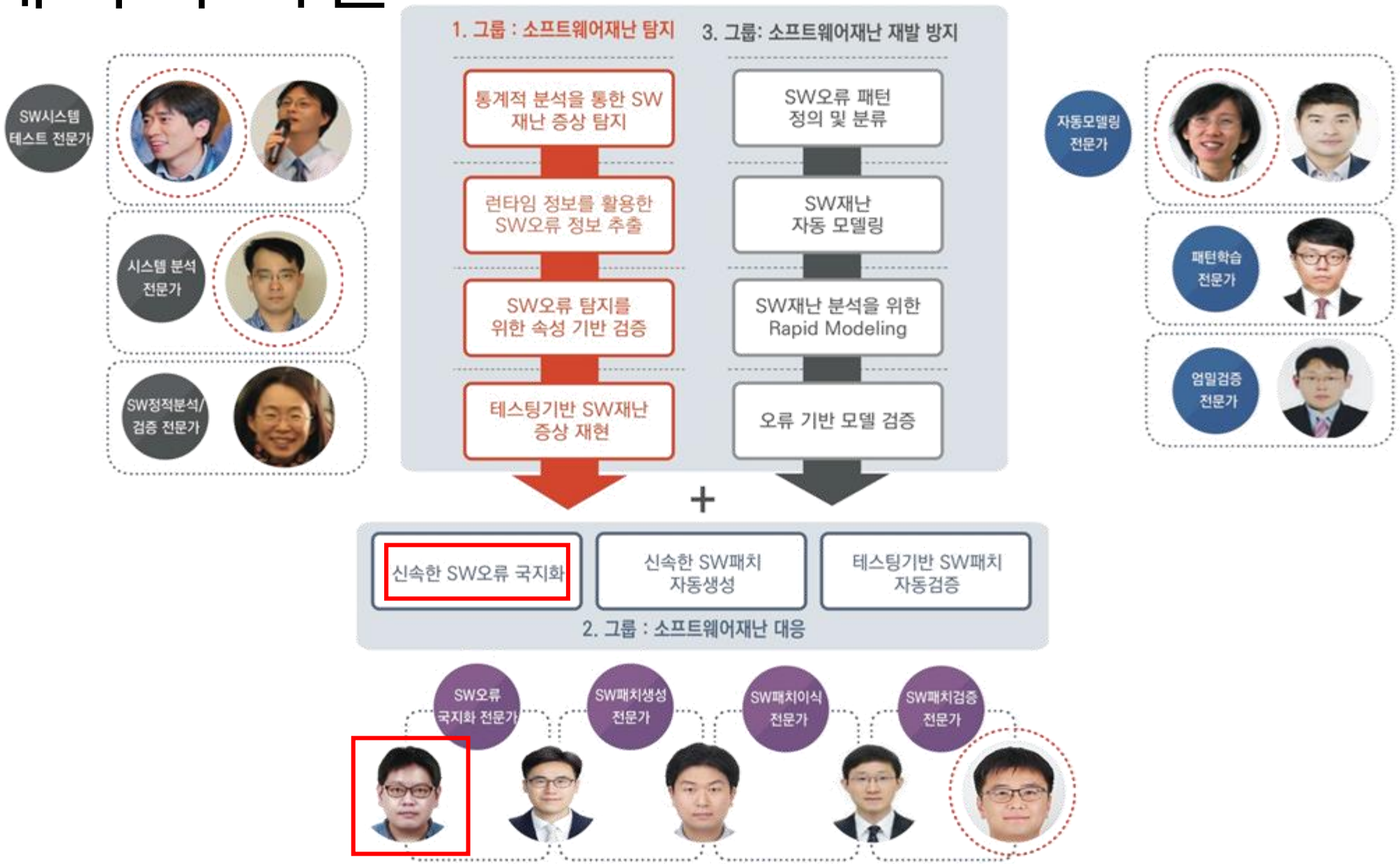
더 빠르고 정확한 오류 위치 찾기

김윤호
한양대학교 컴퓨터소프트웨어학부

센터에서의 역할


SW시스템 테스트 전문가
시스템 분석 전문가
SW정적분석/검증 전문가
1. 그룹 : 소프트웨어재난 탐지
통계적 분석을 통한 SW 재난 증상 탐지
런타임 정보를 활용한 SW오류 정보 추출
SW오류 탐지를 위한 속성 기반 검증
테스팅기반 SW재난 증상 재현
3. 그룹: 소프트웨어재난 재발 방지
SW오류 패턴 정의 및 분류
SW재난 자동 모델링
SW재난 분석을 위한 Rapid Modeling
오류 기반 모델 검증
+
신속한 SW오류 국지화
신속한 SW패치 자동생성
테스팅기반 SW패치 자동검증
2. 그룹 : 소프트웨어재난 대응
SW오류 국지화 전문가
SW패치생성 전문가
SW패치이식 전문가
SW패치검증 전문가
자동모델링 전문가
패턴학습 전문가
엄밀검증 전문가

쉽게 보는 센터에서의 역할

- 소방서 관제팀과 비슷한 일을 하면서 sw 재난에 대응하는 역할

정확하게 찾아줘야하고,

늦으면 안된다!

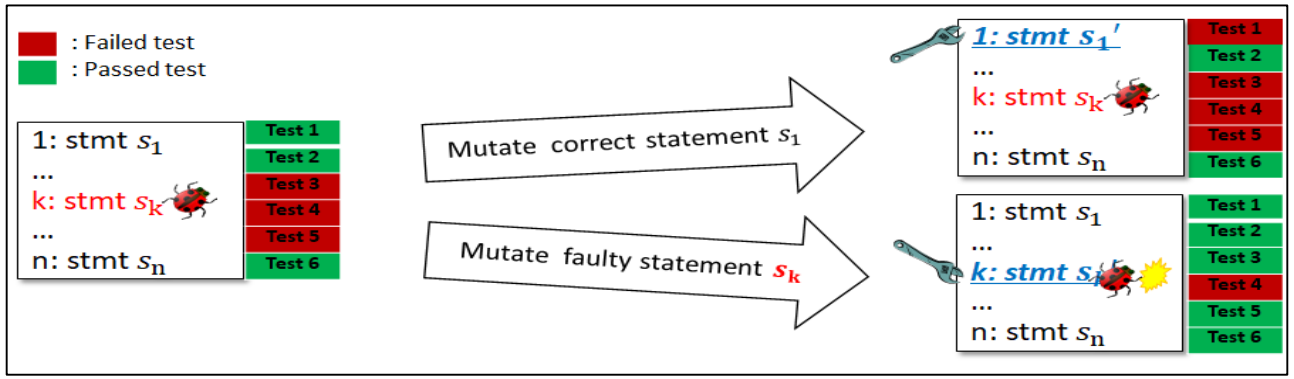
변이 분석을 사용한 정확한 오류 위치 찾기

1. MUSE [ICST 14]
   - 변이 분석을 활용한 정확한 오류 위치 찾기

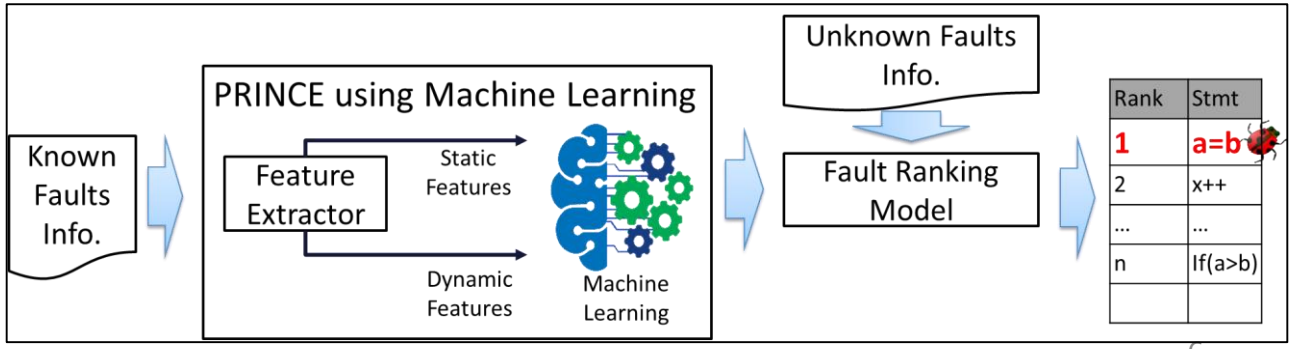
2. MUSEUM [ASE 15, IST 17]
   - 변이 분석을 활용해 다중 언어 프로그램에서 정확한 오류 위치 찾기

3. PRINCE [TOSEM 19]
   - 변이분석에 기계학습까지 더해서 더 정확하게 오류 위치 찾기

MUSE

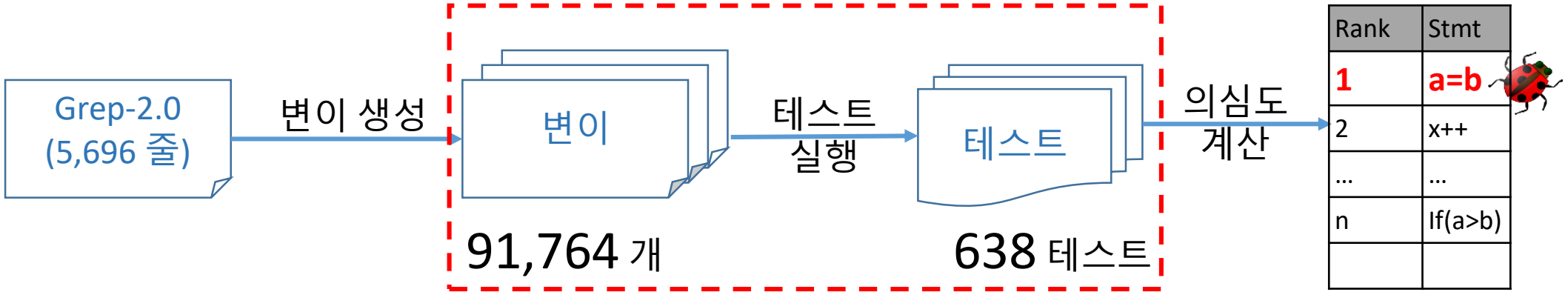


PRINCE



도전과제: 변이 분석이 너무 오래 걸려요

- 변이 분석은 시간 비용이 굉장히 비싸다
- 예: grep-2.0 에서 변이 분석을 사용해 오류 위치를 찾는 경우



- 대략 **~117 시간**

지난화 이야기

- 기계학습과 변이분석을 활용해서 좀 더 빠르게 오류 위치 찾는 방법 발표

(학습에 들어가는 시간 빼고)
10시간만 있으면 오류 위치를 꽤 정확하게 찾을 수 있습니다!

어림도 없지! 아무리 길어도 **1시간** 안에 대응까지 끝내지 못하면 **안.됩.니.다.**

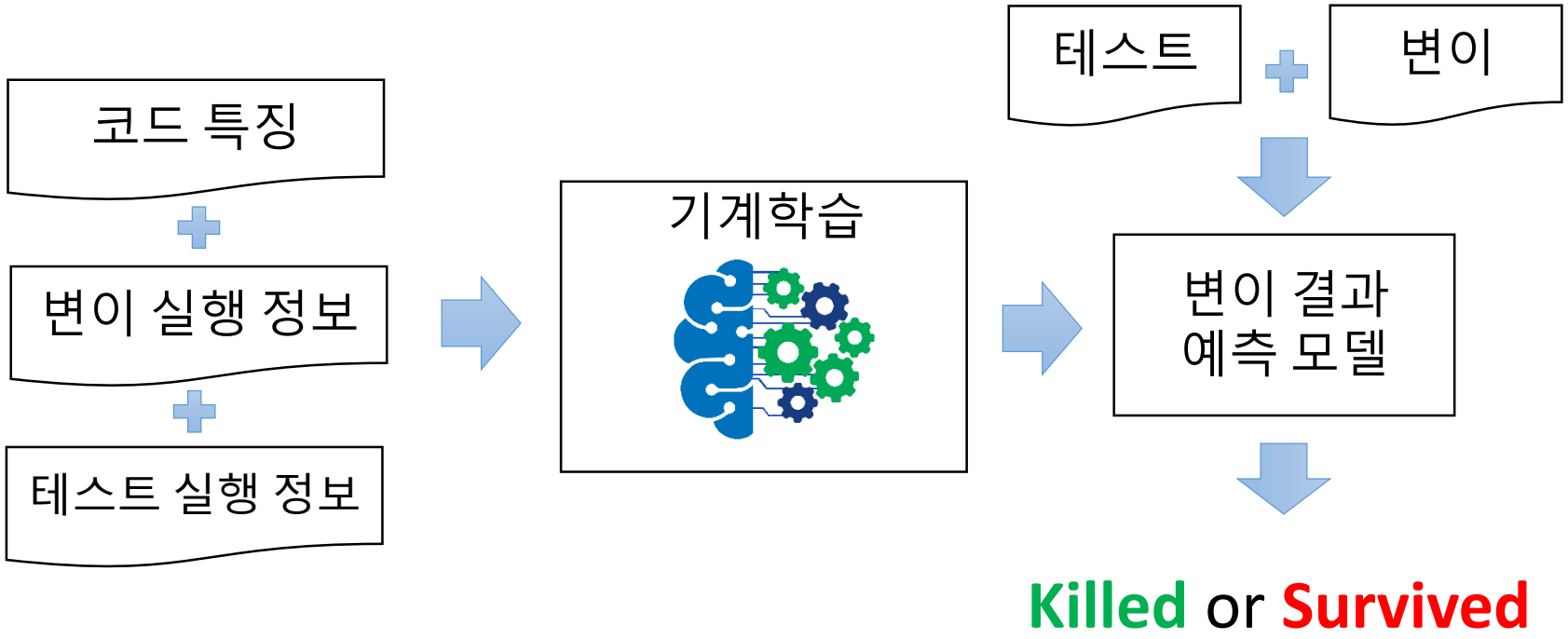
네... 알겠습니다.

어떻게 시간 비용을 더 줄일 수 있을까?

1. 변이 실행 너무 비싼데 그냥 과감하게 포기해보자
   - 포기하기엔 너무 아깝다
   - 변이 분석을 사용하면 커버리지만 사용할 때 보다 오류 위치를 4배 정도 더 정확하게 찾는다

2. 변이 실행 결과를 실행하지 않고 예측해보자
   - 덜 정확해도 되니까 과거 변이 실행 결과를 보고 미래를 예측해보자
   - Predictive Mutation Testing

Predictive Mutation Testing (PMT)



실험 설정

- 5개 SIR 벤치마크 프로그램의 75개 버그 대상
  - Bash와 Vim은 너무 커서 일단 제외

프로그램	버그 수	코드 줄 수	테스트 수
flex	19	7254	567
grep	18	5696	809
gzip	16	3040	208
make	19	9820	1006
sed	3	3980	360
Average	15.0	5879.8	275.8

- PMT 구현
  - 특징: 변이 연산자 종류, 변이된 코드의 총 실행 횟수, 주어진 테스트가 변이된 코드를 실행한 횟수
  - 학습 알고리즘: Random Forest

1차시도 - 실패

- PMT 를 적용한 MUSE가 오류 구문을 정확하게 찾지 못함
  - 정확도 메트릭 Expense: 버그를 찾기 위해 살펴봐야 하는 코드 구문 비율
- 커버리지 정보만 사용한 Op2보다 오히려 더 부정확함

Expense(%)	MUSE	Op2	MUSE with PMT
flex	13.7	19.4	25.5
grep	1.3	2.6	17.6
gzip	5.3	6.7	19.3
make	3.9	15.5	34.5
sed	1.2	11.4	26.3
평균	**5.1**	**11.1**	**24.6**

1차시도 실패 이유 분석

- Java 프로그램을 대상으로 개발된 PMT가 C 프로그램에서 잘 안되는 거 아닌가?
  - 변이 실행 결과 예측 정확도는 크게 나쁘진 않음
  - 10-fold 교차검증 평균 precision, recall 0.88 이상

- 성공한 테스트로만 학습한 PMT 모델로 실패하는 테스트의 결과를 예측하기 힘들다
  - MUSE 에서 필요한 정보: 코드 위치 l의 코드를 변이시키면
    - 성공한 테스트 TC의 결과는 어떻게 바뀌겠는가?
    - **실패**한 테스트 TC의 결과는 어떻게 바뀌겠는가?

2차시도: 실패한 테스트를 고려한 PMT 모델

- PMT 모델을 학습할 때 실패한 테스트도 사용하자

- 실패한 테스트는 어떻게 구할까?
  - 버그를 삽입해서(=변이를 해서) 실패한 테스트를 만들자

- 문제점: 학습 데이터 생성이 너무 비쌈
  - (테스트 실행 시간) * (생성된 변이 수)도 버거운데
  - (테스트 실행 시간) * **(생성된 변이 수)^2** 가 걸린다

- 임시방편: 한 줄당 한 개씩만 버그를 삽입해보자

실험 결과 - 정확도

- MUSE with PMT 보다 정확도 1.75배 향상
- MUSE 보다는 약 40% 정확도 낮음

Expense(%)	MUSE	Op2	MUSE with PMT	MUSE with PMT^2
flex	13.7	19.4	15.5	14.5
grep	1.3	2.6	7.6	4.1
gzip	5.3	6.7	9.3	7.7
make	3.9	15.5	14.5	6.3
sed	1.2	11.4	16.3	3.4
평균	**5.1**	**11.1**	**12.6**	**7.2**

실험 결과 – 실행 시간

- MUSE with PMT2: 1 버그당 평균 약 25분 소요
- 모델 학습용 데이터 생성 시간: 3.3GHz, 100노드 (400코어) 사용시 한 프로그램당 1~2일

시간(h)	MUSE	Op2	MUSE with PMT	MUSE with PMT^2
flex	9038.5	0.1	0.3	0.4
grep	10786.1	0.1	0.5	0.7
gzip	6222.4	0.1	0.4	0.5
make	10097.2	0.2	0.7	0.9
sed	7088.4	0.1	0.6	0.8
전체 시간	**43232.6**	**0.6**	**2.5**	**3.3**
버그 1개당 소요시간	**576.4**	**0.01**	**0.03**	**0.04**

향후 계획

- 자연어 채널을 사용한 기막히게 좋은 최신 PMT 기술 도입
- 학습 시간을 줄여보자
- 비싼 돈 들여서 학습했으면 최대한 우려먹자
  - 1.0 버전에서 학습한 모델이 있으면 1.15 버전까진 써먹을 수 있어야 남는 장사
  - Cross-version 성능 측정 실험 및 파인 튜닝 방법 연구
- 더 좋은 학습 방법을 찾아보자
  - Random Forest가 과연 최선인가?

이번 워크샵에서는?

1. 그동안 열심히 연구했던 내용을 최대한 자랑
2. 미흡한 부분에 대한 솔직한 토의
3. 더 나은 연구자가 되기 위한 동기 부여
4. 공동체 의식 키우기

STAAR