

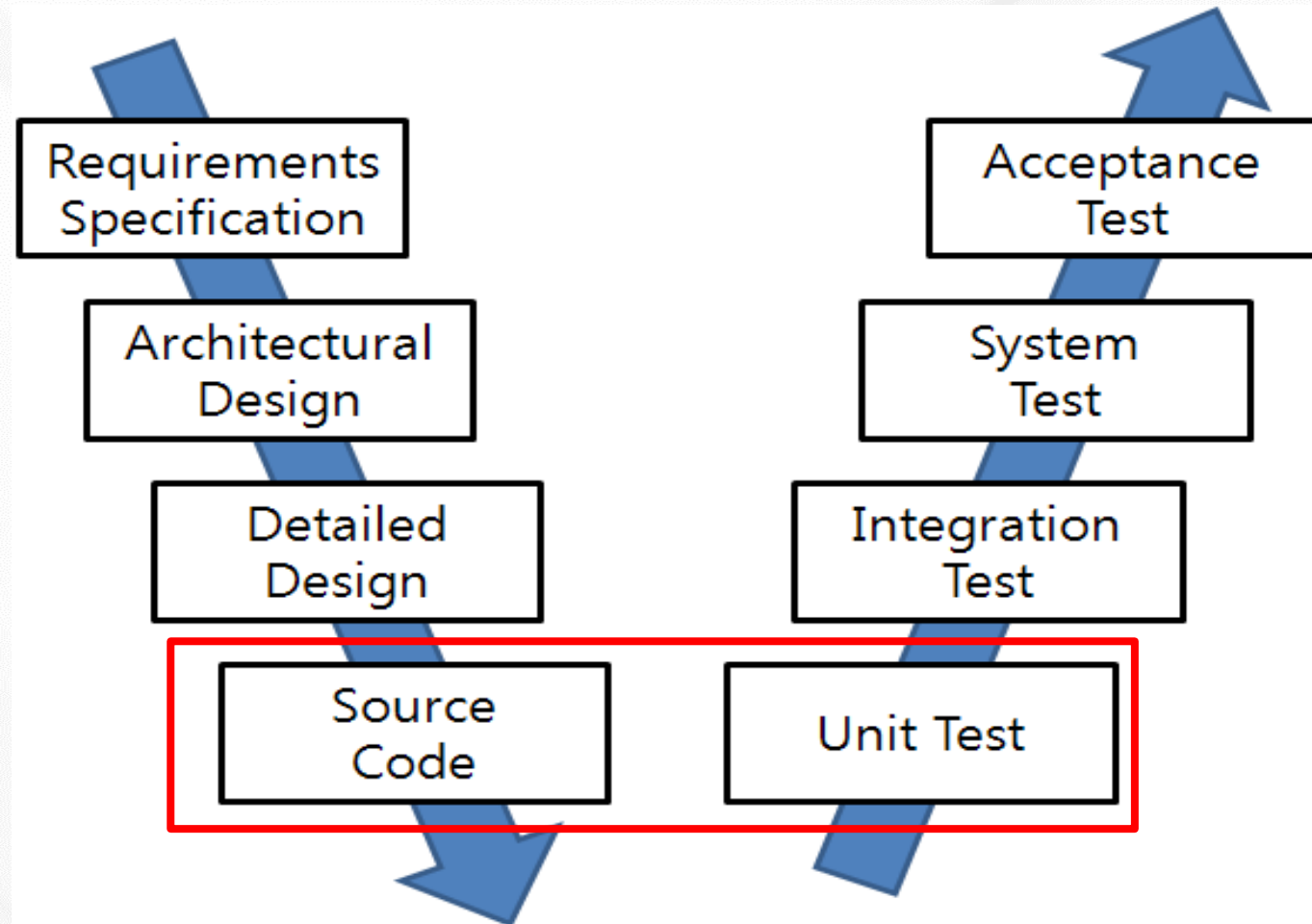
Automated Unit Testing

Moonzoo Kim
moonzoo.kim@gmail.com



SW Development and Testing Model (V model)

Manual
Labor



Abstraction

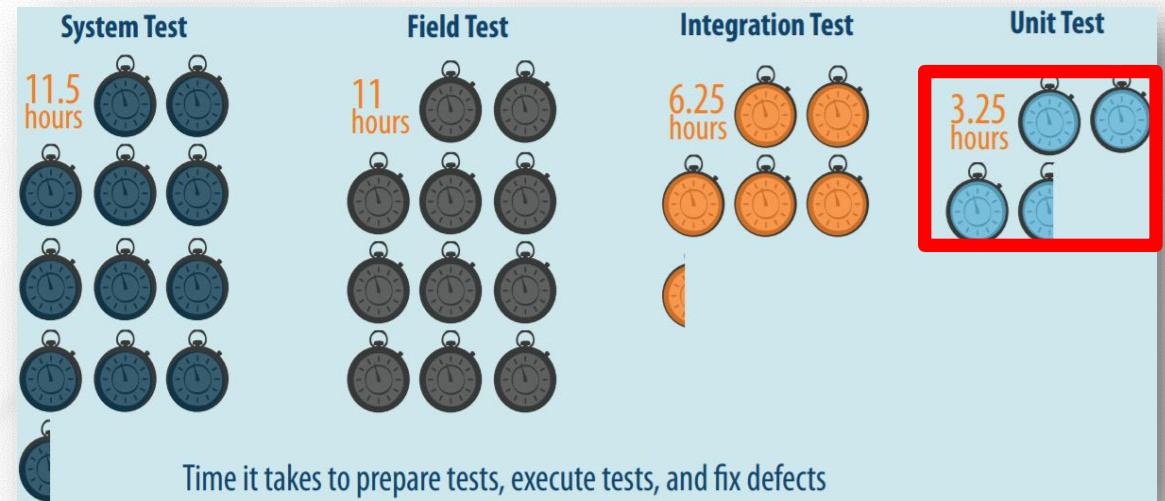
Many Benefits of Unit Testing

- › **Bug correction cost: 7x cheaper** than system tests
 - › \$937 (unit test) vs \$7,136 (system test)

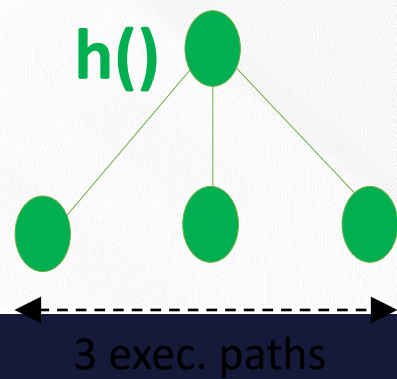
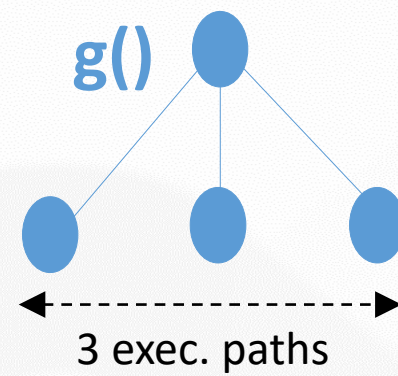
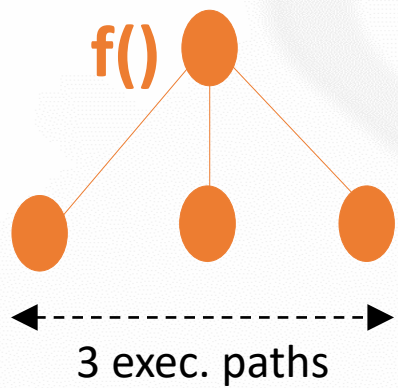


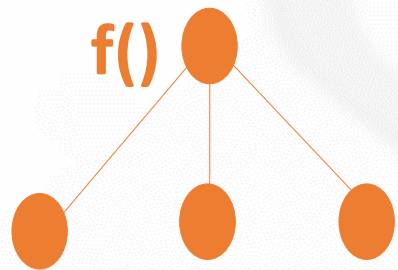
Source: B. Boehm and V. Basil, Software Defect Reduction Top 10 List, IEEE Computer, January 2001

- › **Bug correction time: 3x faster** than system testing
 - › 3.25 hours vs 11.5 hours

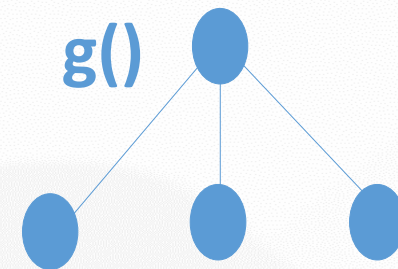


Source: Capers Jones, Applied Software Measurement: Global Analysis of Productivity and Quality

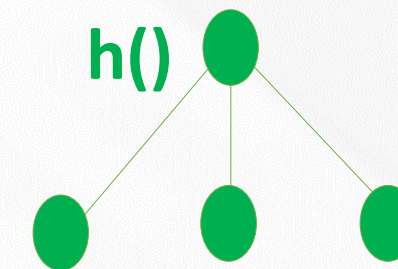




3 exec. paths

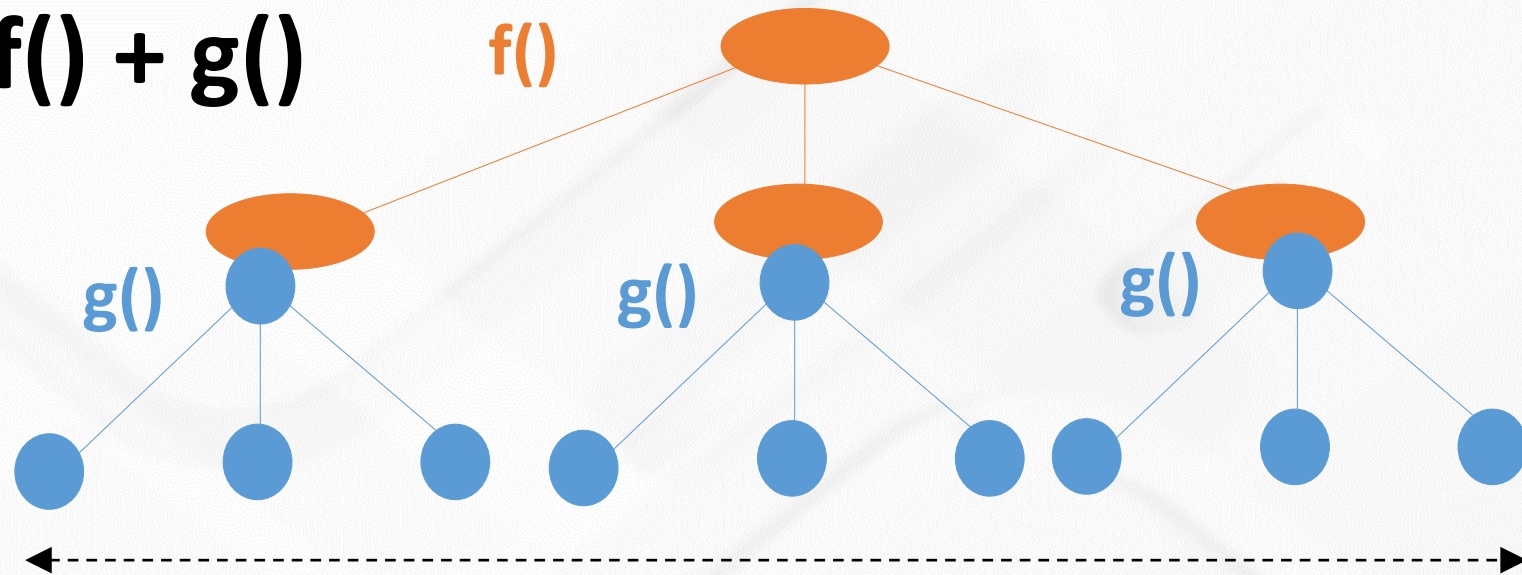


3 exec. paths

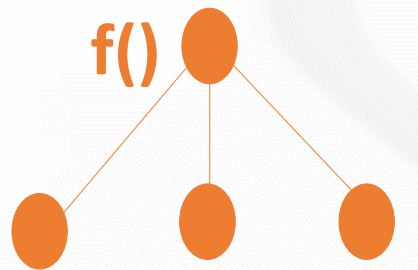


3 exec. paths

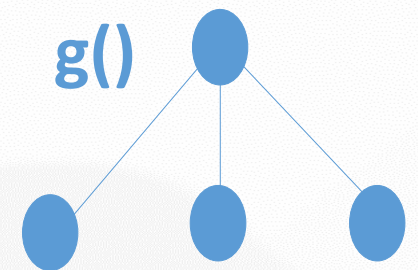
$f() + g()$



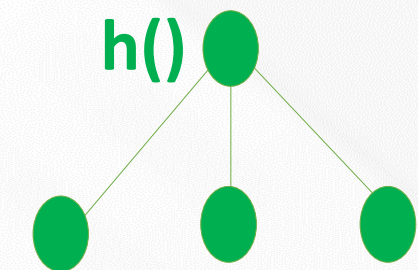
9 ($=3^2$) execution paths



3 exec. paths

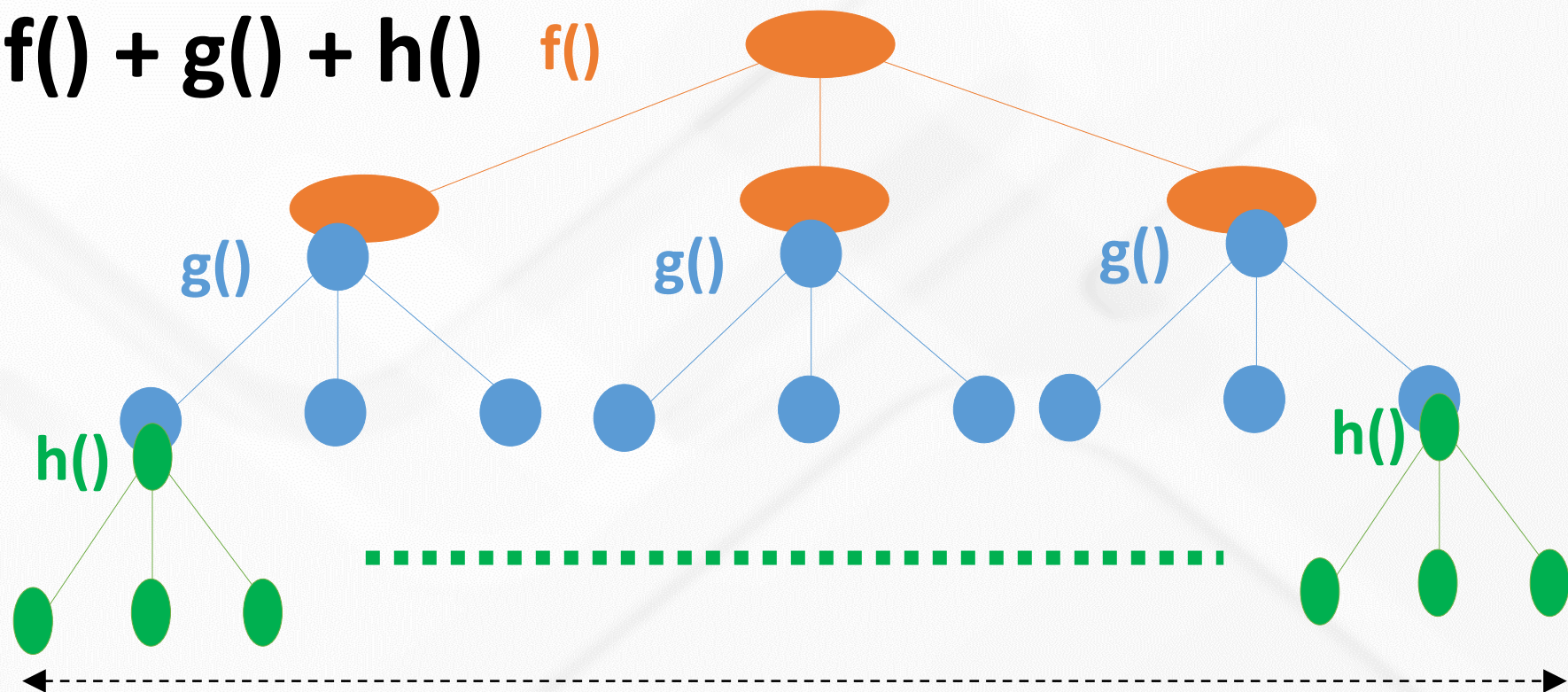


3 exec. paths



3 exec. paths

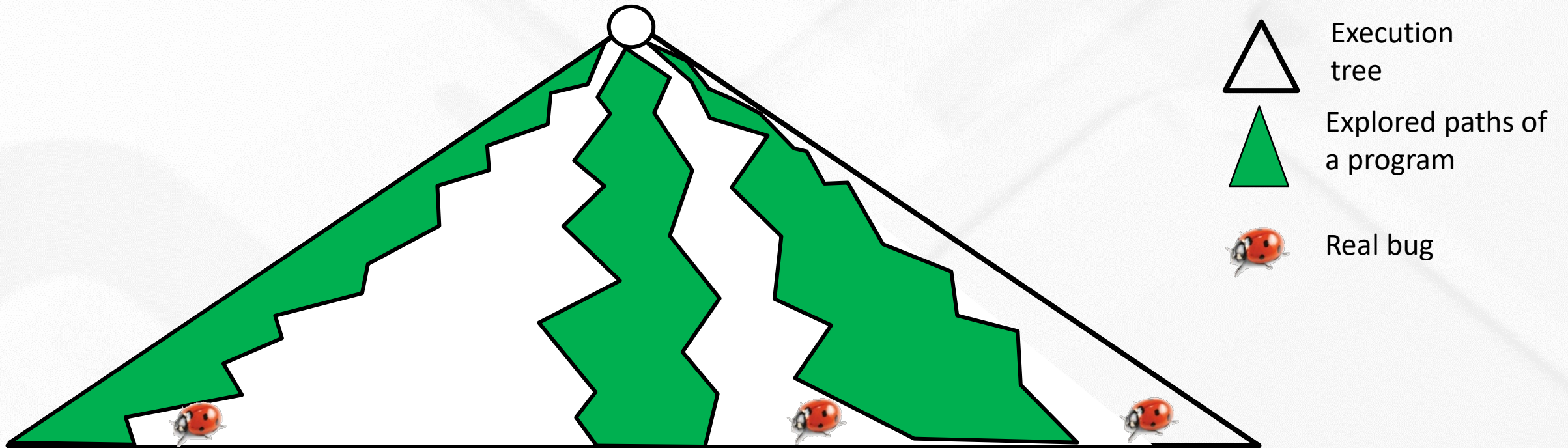
$f() + g() + h()$ $f()$



27 ($= 3^3$) execution paths

Pros and Cons of Auto. Test Gen. in System-level

- › Pros: **No false alarms**
- › Cons: Low bug detection power due to **large search space**



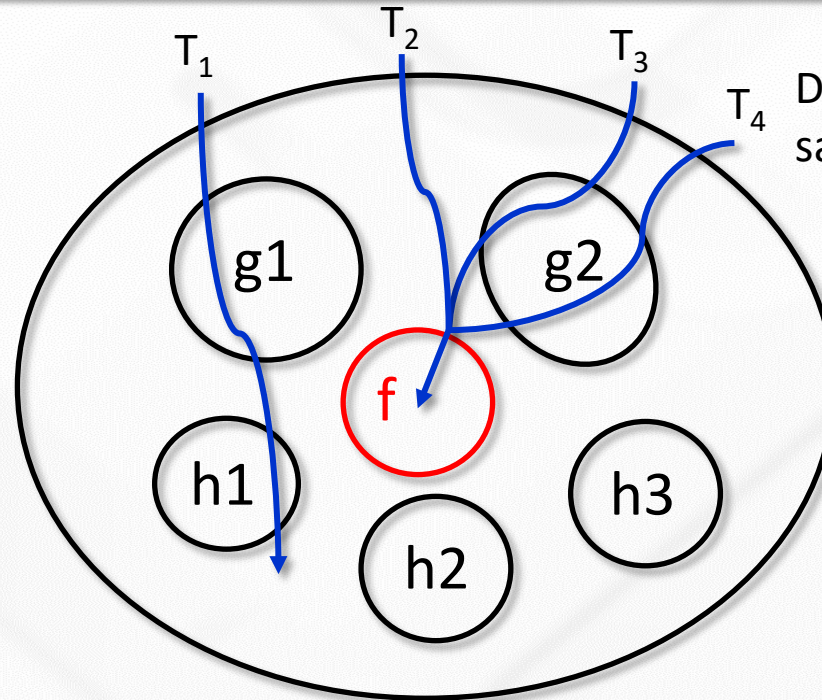
Automated Test Generation in System-Level

Pros

- + Can be easy to generate system TCs due to clear interface specification
- + No false alarm (i.e., no assert violation caused by infeasible execution scenario)

Cons

- Low controllability of each unit
- Large and complex search space to explore in a limited time
- Hard to detect bugs in corner cases



Different system tests T_2 to T_4 exercise the same behavior of the target unit

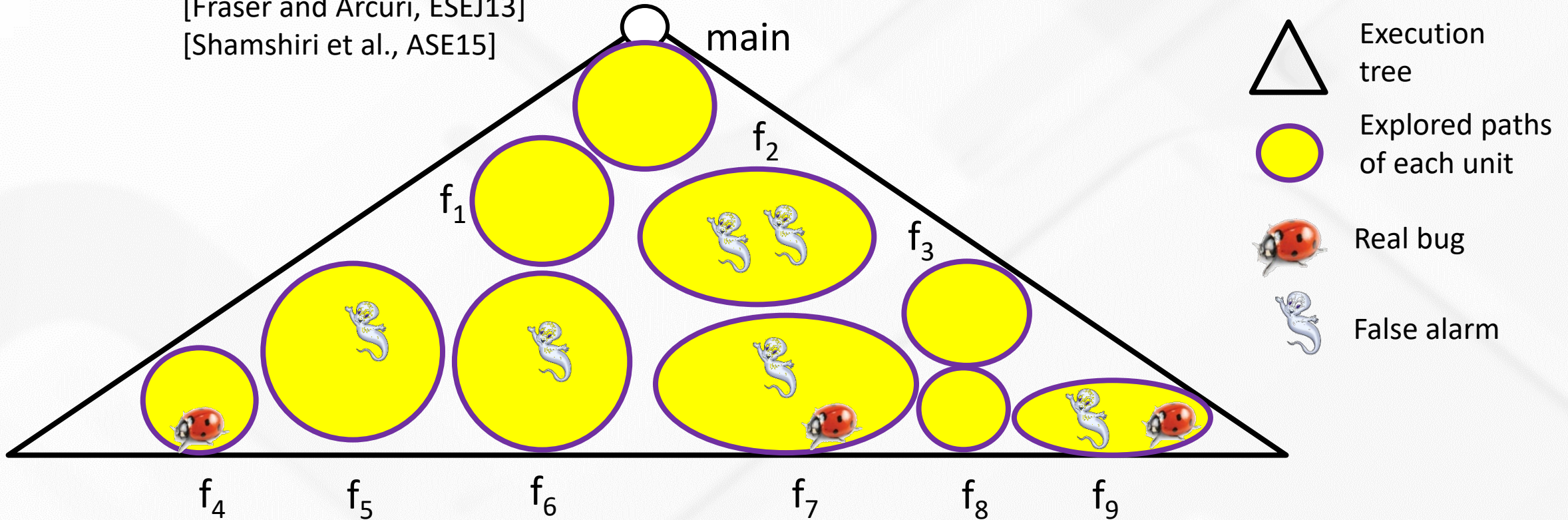
Pros and Cons of Auto. Test Gen. in Unit-level

- › Pros: High bug detection power for **small search space**
- › Cons: **Many false alarms** due to over-approximated context of a unit

[Gross et al., ISSTA12]

[Fraser and Arcuri, ESEJ13]

[Shamshiri et al., ASE15]



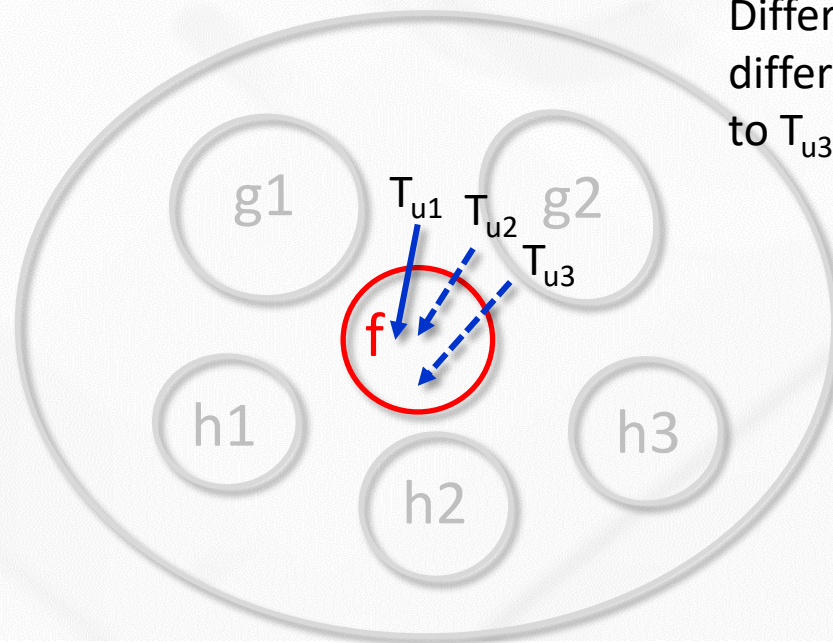
Automated Test Generation in Unit-Level

Pros

- + High controllability of a target unit
- + Smaller search space to explore than system testing
- + High effectiveness for detecting corner cases bugs



Cons

- Hard to write down accurate unit test drivers/stubs due to unclear unit specification
- High false/true alarm ratio



Different unit tests T_{u1} to T_{u3} directly exercise different behaviors of the target unit, but T_{u2} to T_{u3} exercise infeasible paths

Legend

	A feasible execution
	An infeasible execution

Approximate Input Space for Symbolic Unit Testing

Over-approximated
symbolic inputs

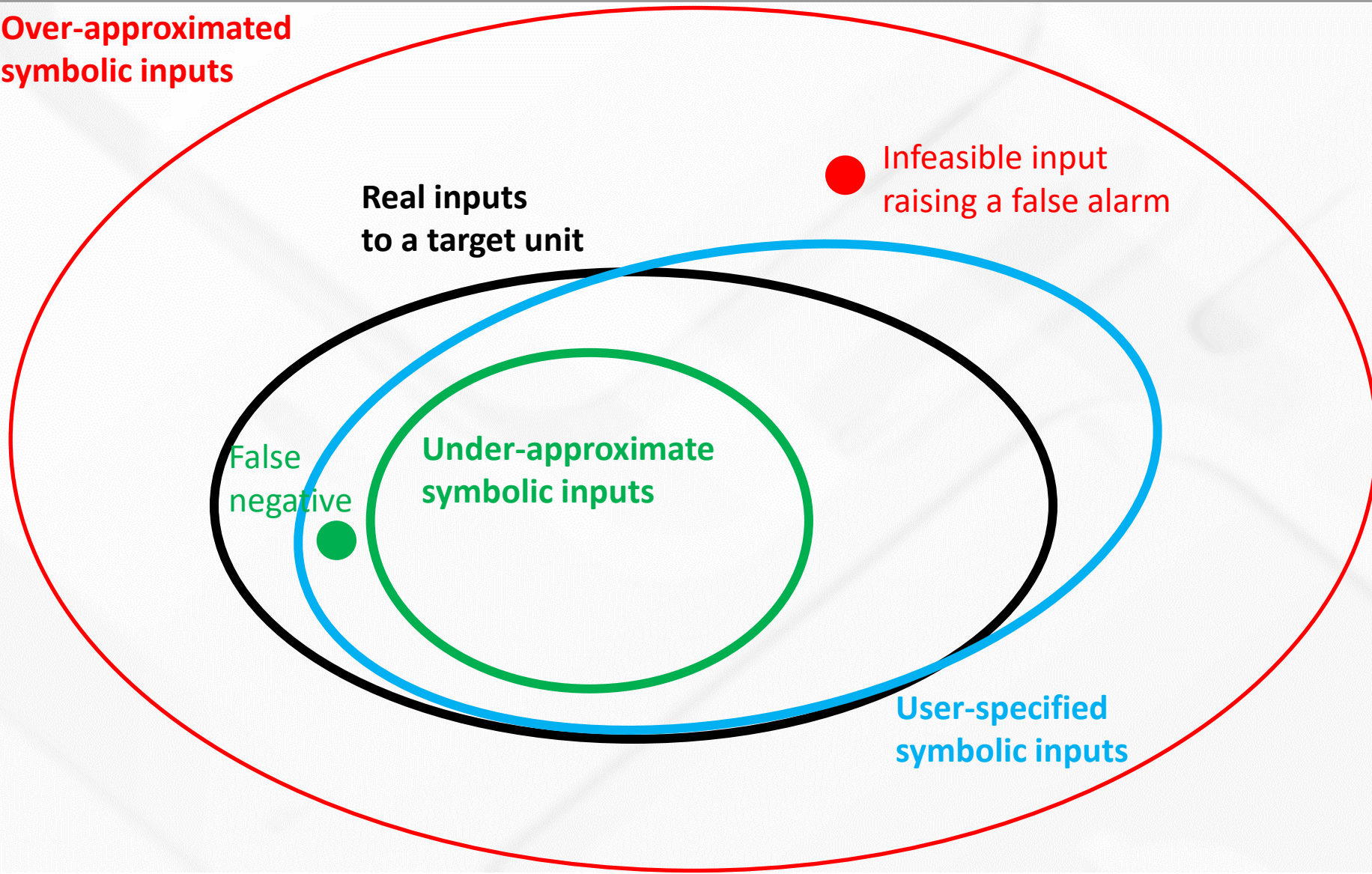
Real inputs
to a target unit

● Infeasible input
raising a false alarm

False
negative
●

Under-approximate
symbolic inputs

User-specified
symbolic inputs



Related Work on Automated Unit Testing

	Bug detection ability	False/True alarm ratio	Target languages
Function input generation [PLDI 05][FSE 05][EMSOFT 06][TAP 08][ISSTA 08][SEC 15]	High	High	Procedural or OO languages
Method-sequence generation [ICSE 07] [ICST 10][FSE 11] [ICSE 13]	High	Medium	Object-oriented languages
Capture system tests to generate unit tests [TSE 09] [STTT 09][ISSTA 10]	Low	Low	Object-oriented languages
Automated Unit Test Generation with Realistic Unit Context	High (86.7% of target bugs in SIR and SPEC2006)	Low (2.4 false alarms per one true alarm)	Procedural languages

Constructing Unit Test Driver/Stubs

› Example of an automatically generated unit-test driver

```
01:typedef struct Node_  
02:  char c;  
03:  struct Node_ *next;  
04:} Node;  
05:Node *head;  
06:// Target unit-under-test  
07:void add_last(char v){  
08:  // add a new node containing v  
09:  // to the end of the linked list  
10:  ...}  
11:// Test driver for the target unit  
12:void test_add_last(){  
13:  char v1;  
14:  head = malloc(sizeof(Node));  
15:  SYM_char(head->c);  
16:  head->next = NULL;  
17:  SYM_char(v1);  
18:  add_last(v1); }
```

} Set global
variables

] Set parameter

Unit Test Driver

Generate symbolic inputs
for global variables and
parameters



Call target function

Trade-off between Bug Detection Ability and Accuracy

Bug Detection **Ability**
(aiming low false
negative)

Recall



Bug Detection **Accuracy**
(aiming low false
positive)

Precision

Cutting-edge Accuracy of Unit Testing



CONBRIO [Kim'18]:
4.5:1 w/ 91% of
bugs detected



- ▶ Goal: To detect bugs in Samsung smartphones
- ▶ Project period: '10~'14
- ▶ Project funding: 400,000 USD
- ▶ Results:
 - ▶ Developed Concolic unit-testing tool **CONBOL**
 - ▶ Detected **many crashes** in 4 MLOC smartphone SW

2014

Automated Unit Testing of Large Industrial Embedded Software using Concolic Testing


Yunho Kim	Yonil Kim, Taeksoo Kim, Gunwoo Lee, Yoonkyu Jang	Moonsoo Kim
CS Dept. KAIST	Samsung Electronics	CS Dept. KAIST
South Korea	South Korea	South Korea
Email: kimyunho@kaist.ac.kr	Email: [yonil.kim, taeksoo.kim, gunwoo.lee, yoonkyu.jang]	Email: moonsoo@cs.kaist.ac.kr

[illegible]

To address the aforementioned challenges, we have developed CONCORDE and SYMBOLIC (CONCORDE) toolchain that generates symbolic unit testing drivers/strats and performs concolic testing on a host PC automatically. Conventional unit testing requires a programmer to write test cases and then perform unit testing and they can make unit testing drivers/strats correctly and specify requirement properties based on their knowledge of the target units [27]. However, this development is often not true in industrial projects since developers have little time to write test cases and a tight project schedule (for example, smartphone products are often developed in less than six months due to heavy market competition). CONCORDE handles the aforementioned tasks (i.e., generate test cases and perform unit testing) automatically. CONCORDE has additional benefits for testing embedded software by applying concolic testing at unit level since unit testing has less dependency on the target hardware platform, unlike debugging and integration testing [28].

CONCORDE targets *small* (i.e., small point-to-point devices), *critical* (i.e., safety-critical), and *high-risk* (i.e., high-risk) systems, because

testing to embedded software due to the following challenges: these crash bugs can cause entire software to fail and, thus,

[illegible][illegible][illegible]

An aerial photograph of Zurich, Switzerland, showing the city built on islands and along the shores of Lake Zurich. The city features a mix of historic architecture, including a prominent church with a tall spire, and modern buildings. The lake is visible in the background, surrounded by mountains.

[illegible]

target
oncologic
shadows
utilize
and, too
needs and
difficulties

ances to
diseases
diseases
and
management
to
such as
port code
main main
negotiate a
rationalization,
hubs.

develop-
network
effective
critical
target
strong
as based
on this
where
to right
to other
market
to (con-
dition)
training
main local
public use

renewals,
renewal
and, thus,

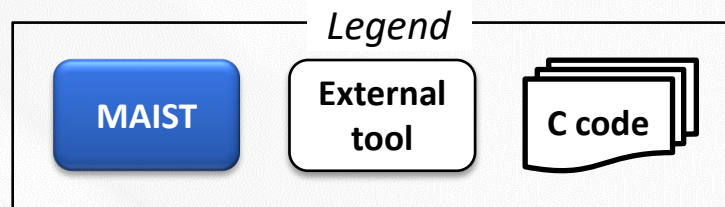
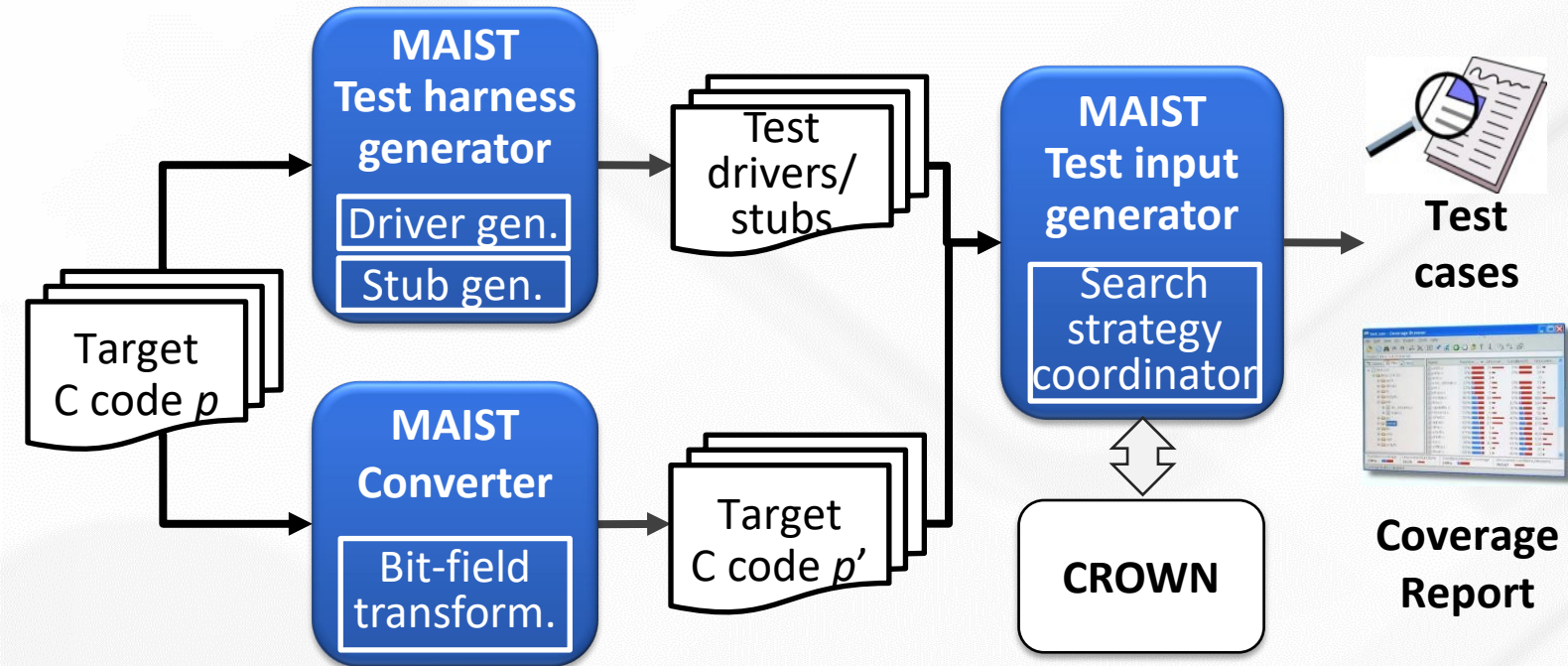


- FSE '2011

- ICSE 2012, ICST 2012

- ASE 2013

Concolic Unit Testing Project w/ Hyundai and Mobis for 5 years



- CROWN features**
- Bit-level accuracy
 - Easy-to-extend
 - Low memory overhead

■ 현대모비스 인공지능 도입 사례

AI 시스템	목적	도입 효과
마이스트 (Mobis AI Software Testing)	소프트웨어 검증 자동화	<div> 통합형 차체제어장치(TBU) 투입 인력 53% 감소 </div> <div> 써라운드 뷰 모니터링(SVM) 투입 인력 70% 감소 </div>
마이봇 (Mobis AI Robot)	소프트웨어 개발문서 검색	답리닝 기반, 개발문서 20만 건 관리

<http://m.yna.co.kr/kr/contents/?cid=AKR20180720158800003&mobile>

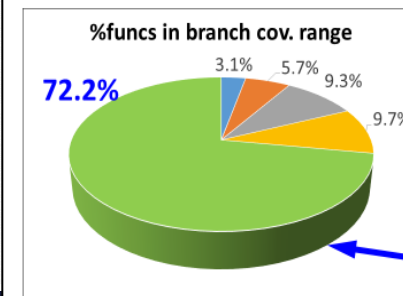
KAIST

Concolic Testing for High Test Coverage and Reduced Human Effort in Automotive Industry

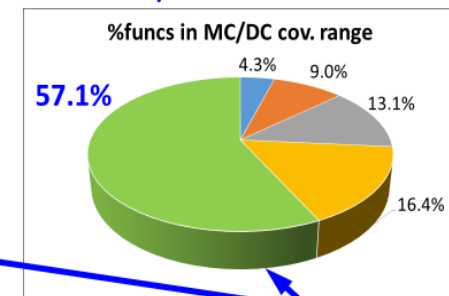
Page 16

RQ1: MAIST Achieved **90.5% Branch** and **77.8% MC/DC Cov.**

100% branch cov. of 72.2% of funcs



100% MC/DC cov. of 57.1% of funcs



■ [20%,40%) ■ [40%,60%) ■ [60%,80%) ■ [80%,100%) ■ 100%

* Running 20 hours on 12 CPU cores (3.0GHz)

Practical Benefit of Automated Unit Testing

System testing is **expensive** and **less effective**

- › **Full vehicle HW** and **human drivers** are required
- › Driving a car with **various physical environments** spends **a lot of time**
- › **Hard-to-achieve high test coverage** due to low controllability

VS

Unit testing is **cheap** and **more effective**

› Full vehicle HW and human drivers are **NOT necessary**

› No need to drive a car

› **Easy-to-achieve high coverage** due to high controllability



Solution for Huge Economic & Social Cost due to SW Bugs

Labor-intensive Manual Testing
Large SW Testing Cost and Time
Low Bug Detection Ability
Low Product Quality

**Solution: AI-based Automated
Concolic Testing Technique**

movie link <https://bit.ly/3NS6RrQ>

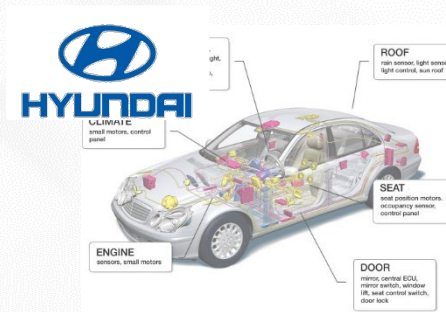
Existing Problems

Developed Solutions



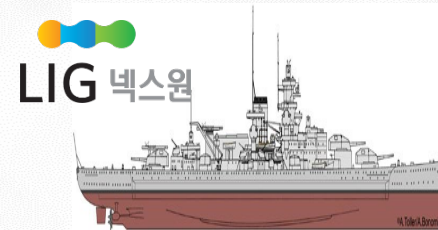
**'10-14 Project w/
Samsung Electronics**

Detected dozens of
crash bugs in the
comm. firmware



**'15~'20 Project w/
Hyundai/Mobis**

Achieved 90% branch
cov. and reduced 80%
of labor cost by using
auto. testing tech.



**'18 Project w/
LIGnex1**

Detected several SW
bugs in the 10
programs in the
battleships

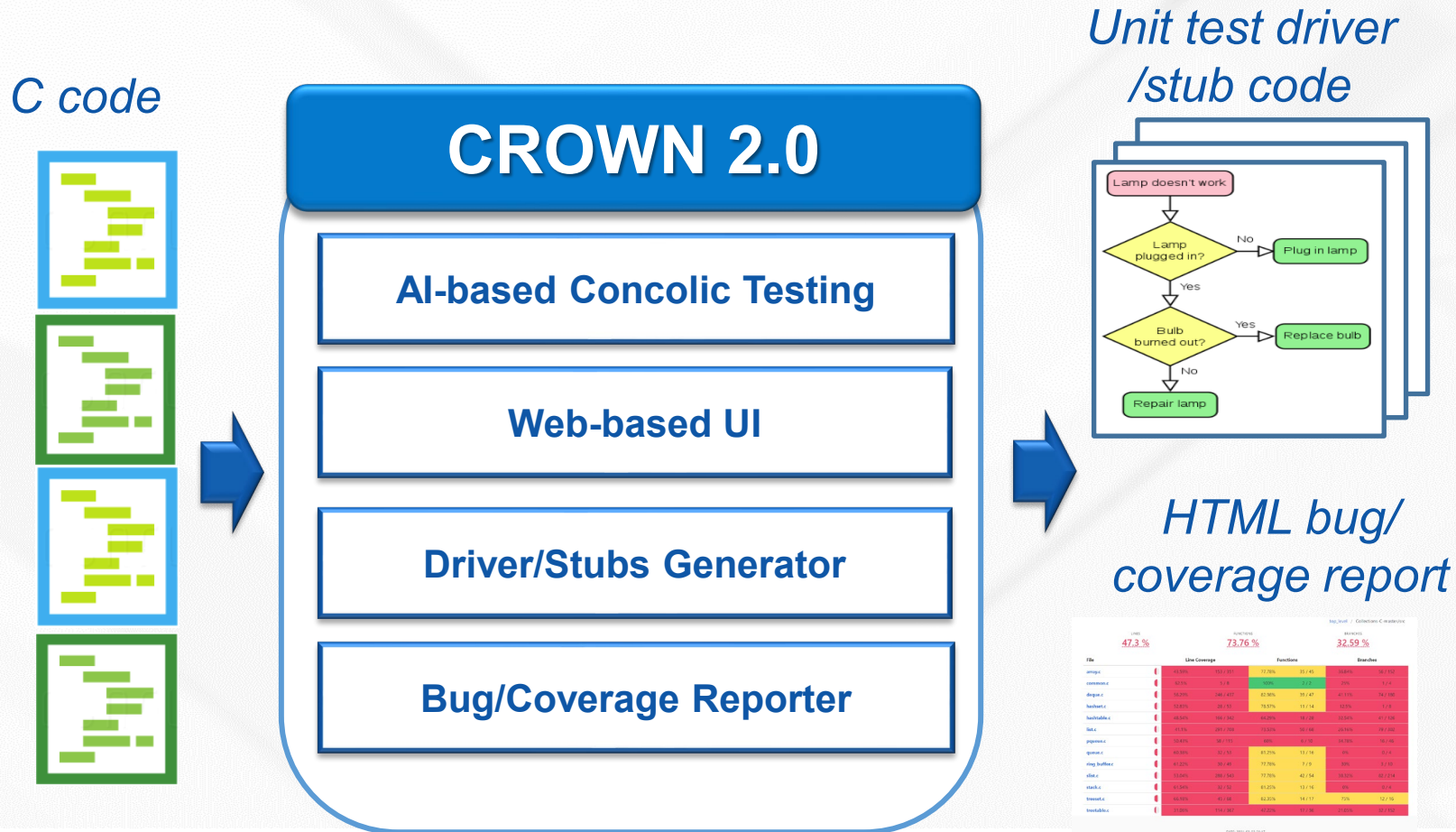


**'20 Project w/ Natl.
Security Research Inst.**

Detected SW bugs in
the software in the
security equipment

CROWN 2.0: Comercial Automated. Unit Testing Tool

CROWN 2.0 is a fully automated software test solution that significantly increases bug detection power and reduces testing cost for embedded C programs



Product Features

- **Automatically build stub and driver code**
- **Automatic test case generation based on AI-based Concolic testing**
- **Code-coverage report and analysis**
- **Test execution playback to help in debugging crash bugs**

Highlights

- **Eliminate need to write test code manually**
- **100% automated test code generation**
- **High-quality unit test generation**
- **Integrated code-coverage analysis**