국제표준의 정적분석 검증 요건

2012. 05. 30

㈜ 소프트4소프트
CEO/CTO 이헌기
02-553-9464

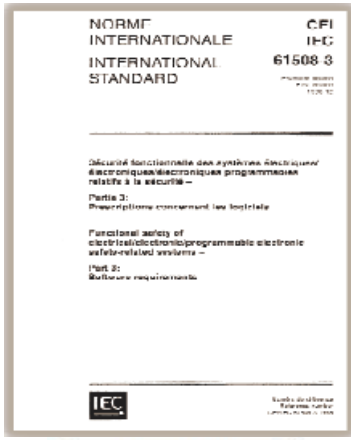
목차

1. 기능 안전성 국제 표준

기능 안전성 이란?

- SW의 오류로 인한 사고를 미연에 방지하기 위해 제정한 기능안전규격
- 새로운 기능의 90%가 SW 사용 (제품단가의 40%가 SW 비용)

Safety Standards Landscape



IEC 61508
1998-2000

안전성 표준

ISO 26262

IEC 62279

DO 178/DO 254

ECSS/CNES

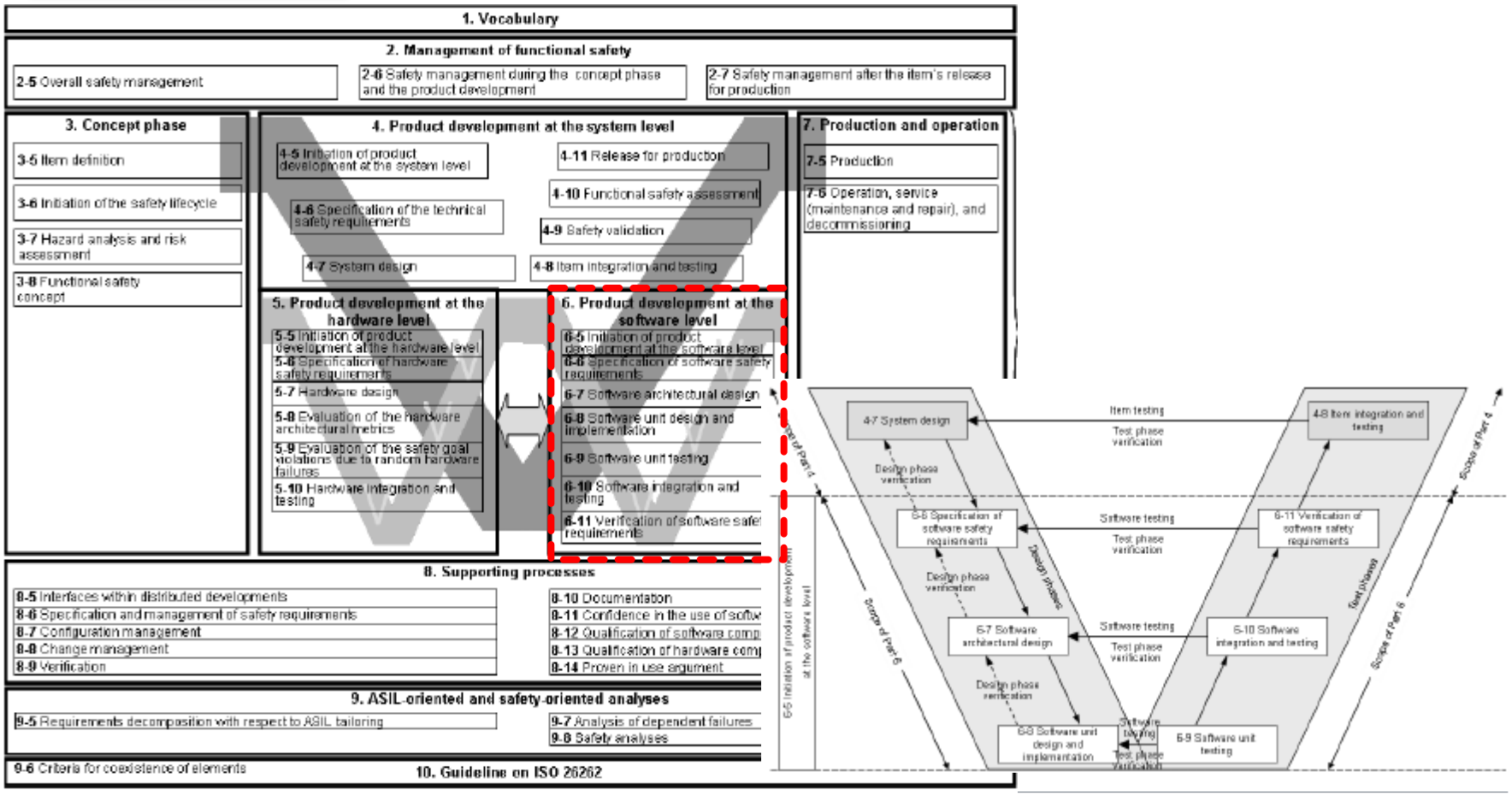
IEC 61226
IEC 60880

2. 국제 표준의 검증과 확인(V&V)

ISO 26262 Part 6

- 자동차내의 전자전기 기기의 기능 안전성 검증 절차

ISO 26262 Core Process



2. 국제 표준의 검증과 확인(V&V)

기능성
신뢰성
사용성
효율성
유지보수성
이식성

ISO 26262 Part 6

- 6.8 Software unit design and implementation 검증 기법 및 설계 원칙

●Walk-through, Inspection

●Semi-formal verification, Formal verification

●Control flow analysis, Data flow analysis, Static code analysis, Semantic code analysis

Design Principles

- 코드단계에서 Software Unit 설계 원칙
- SW Unit에서 기능의 실행의 정확한 순서
- SW Unit간 인터페이스 일치성
- SW Unit에서 또는 사이에서 자료 및 제어 흐름의 정확성
- 적합성
- 견고성 (Ex, Methods to prevent implausible values, execution errors, division by zero, and errors in the data flow and control flow)
- 단순성
- 가독성 및 이해성
- 시험성

Table 8 — Design principles for software unit design and implementation

	Methods	ASIL A	ASIL B	ASIL C	ASIL D
1a	One entry and one exit point in subprograms and functions[a]	++	++	++	++
1b	No dynamic objects or variables, or else online test during their creation[a, b]	+	+	++	++
1c	Initialization of variables	++	++	++	++
1d	No multiple use of variable names[a]	+	++	++	++
1e	Avoid global variables or else justify their usage[a]	+	+	++	++
1f	Limited use of pointers[a]	o	+	+	++
1g	No implicit type conversions[a, b]	+	++	++	++
1h	No hidden data flow or control flow[c]	+	++	++	++
1i	No unconditional jumps[a, b, c]	++	++	++	++
1j	No recursions	+	+	++	++

No recommendation / Recommended / Highly recommended

[a] Methods 1a, 1b, 1d, 1e, 1f, 1g and 1i may not be applicable for graphical modelling notations used in model-based development.

[b] Methods 1g and 1i are not applicable in assembler programming.

[c] Methods 1h and 1i reduce the potential for modelling data flow and control flow through jumps or global variables.

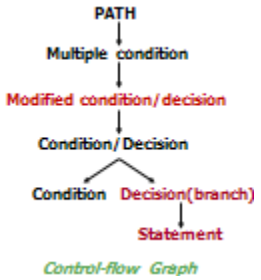
NOTE For the C language, MISRA C[4] covers many of the methods listed in Table 8.

2. 국제 표준의 검증과 확인(V&V)

ISO 26262 Part 6

- 6.9 Software unit testing 기법 및 구조적 커버리지 측정

- Methods for software unit testing
- Methods for deriving test cases for software unit testing
- Structural coverage metrics at the software unit level



Unit Testing Demonstrations

- Software Unit 증명
  - 설계 사양 준수
  - HW-SW 인터페이스의 사양 준수
  - 지정된 기능
  - 의도하지 않은 기능의 부재에 대한 신뢰
  - 견 고 성 (Ex, The absence of inaccessible software, the effectiveness of error detection and error handling mechanisms)
  - 기능을 지원하는 데 충분한 리소스

Methods		ASIL A	ASIL B	ASIL C	ASIL D
1a	Requirements-based test[a]	++	++	++	++
1b	Interface test	++	++	++	++
1c	Fault injection test[b]	+	+	+	++
1d	Resource usage test[c]	+	+	+	++
1e	Back-to-back comparison test between model and code, if applicable[d]	+	+	++	++

[a] The software requirements at the unit level are the basis for this requirements-based test.

[b] This includes injection of arbitrary faults (e.g. by corrupting values of variables, by introducing code mutations, or by corrupting values of CPU registers).

[c] Some aspects of the resource usage test can only be evaluated properly when the software unit tests are executed on the target hardware or if the emulator for the target processor supports resource usage tests.

[d] This method requires a model that can simulate the functionality of the software units. Here, the model and code are stimulated in the same way and results compared with each other.

Methods		ASIL A	ASIL B	ASIL C	ASIL D
1a	Statement coverage	++	++	+	+
1b	Branch coverage	+	++	++	++
1c	MC/DC (Modified Condition/Decision Coverage)	+	+	+	++

2. 국제 표준의 검증과 확인(V&V)

산업별 SW 융합 코딩 표준

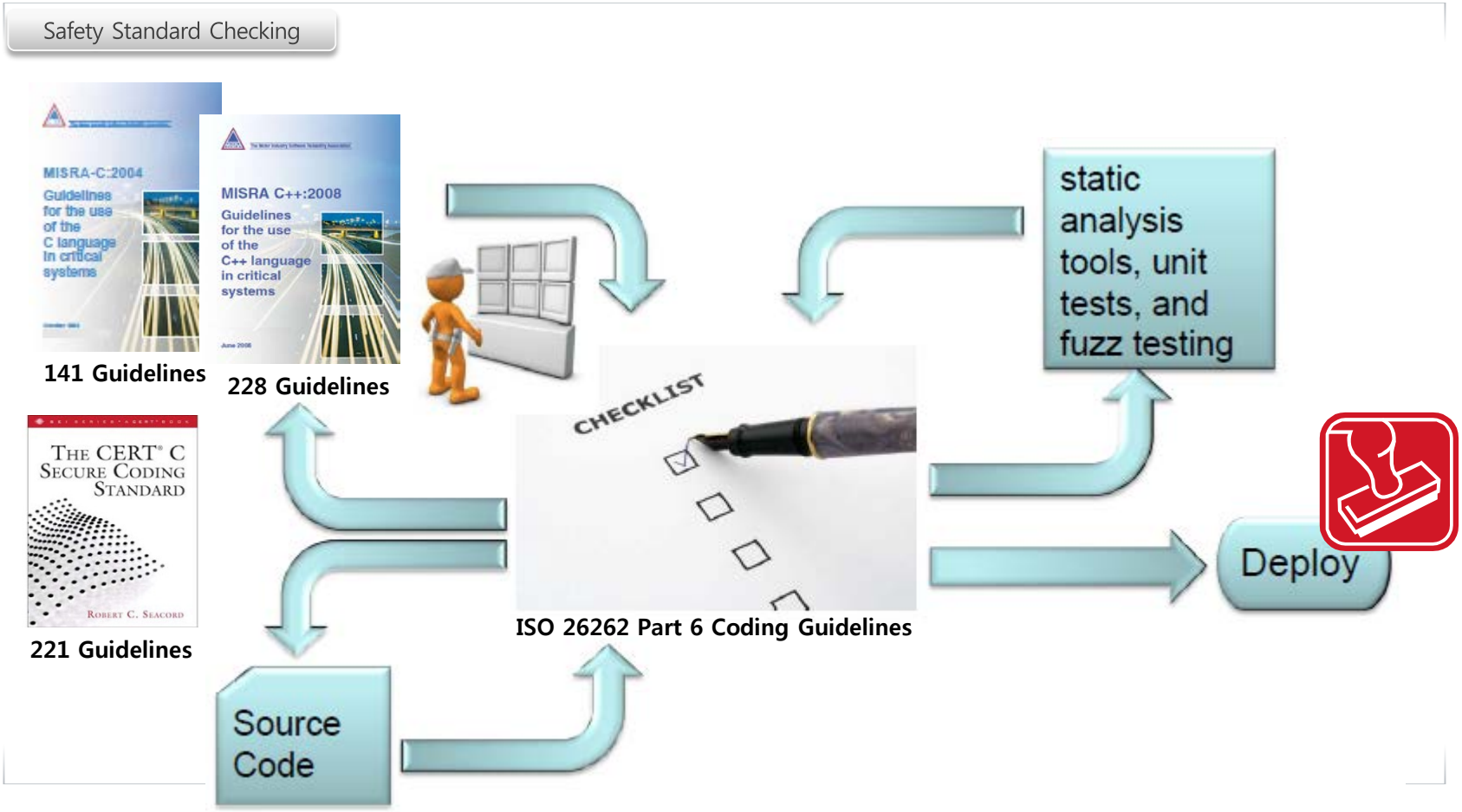
Embedded Coding Standards

C Standards			
MISRA-C:2004	Automotive	141 Rules	ISO 26262 DO-178B DO-278 IEC 62304 IEC 61508
HIS	Automotive	127 Rules	
JPL	Aerospace	23 Rules	
CERT C	Security	221 Rules	
C++ Standards			
MISRA-C:2004	Automotive	228 Rules	ISO 26262 DO-178B DO-278 IEC 62304 IEC 62279 IEC 61508
High Integrity C++	Automotive	198 Rules	
JSF++ AV	Aerospace	231 Rules	
BSS C/C++	Aerospace	124 Rules	
CERT C++	Security	101 Rules	
Java Standards			
BSS Java	Aerospace	223 Rules	ISO 26262 DO-178B DO-278 IEC 61508
JPL	Aerospace	53 Rules	
CERT Java	Security	156 Rules	

2. 국제 표준의 검증과 확인(V&V)

Software unit design and implementation 검증 표준

- MISRA-C:2004(C2)/1998(C1)는 ISO 26262 코딩표준
- MISRA-C:2012(MISRA C3) 공개 예정



2. 국제 표준의 검증과 확인(V&V)

MISRA-C: 2004 예제

- MISRA-C는 ISO 26262 코딩표준(Table8: Design principles for software unit design and implementation)

MISRA-C:2004 Example Rule

- Rule 9.1(초기화하지 않은 변수)
  - All automatic variables shall have been assigned a value before being used.
  - Rational: a variable which is used before it has been assigned a value will likely cause data flow anomalies

- Data Anomaly(변수의 읽기 또는 쓰기에 따라 변수 결함 구분)
  - UR Anomaly
  - DD Anomaly
  - DU Anomaly
  - Dead Variable

Anomaly 유형	설명

Sample Code	1 int t, x, y, z; // Dead Code(t)
	2
	3 x = 1; // DD Anomaly(x)
	4 if(y>0) // UR Anomaly(y)
	5 x=2;
	6 z=x+1; // DU Anomaly(z)

UR Anomaly	A variable not written but read - This is a **bug** and leads to an error - ex) 4 line
DD Anomaly	A value written twice to a variable - This is ominous but don't have to be a bug - ex) 3 line (6 line)
DU Anomaly	A variable written but not read - It can be removed to improve the architecture - ex) 6 line
Dead Code	A variable is neither written nor read - It can be removed to improve the readability - ex) 1 line

2. 국제 표준의 검증과 확인(V&V)

CERT C 예제

- CMU SEI CERT(Computer Emergency Response Team) Secure Coding Standards
- CERT는 C 프로그래밍 언어에서의 안전한 코딩에 대한 가이드라인 제공

CERT C Example Rule

- MEM31-C (동적으로 할당된 메모리는 한 번만 해제)
  - Freeing memory multiple times has similar consequences to accessing memory after it is freed.
  - Rational: To eliminate double-free vulnerabilities, it is necessary to guarantee that dynamic memory is freed exactly one time
- Memory Anomaly
  - Memory leak
  - Double free
  - Use after free
  - Free null pointer

Sample Code

```
char* ptr = (char*)malloc (SIZE);

if (abrt) {
   free(ptr);
}
...
free(ptr); /* potential double-free */
```

2. 국제 표준의 검증과 확인(V&V)

IEC 62279 코딩 가이드라인 (철도 RAMS 인증)

- 열차제어시스템 SW C코딩지침서

IEC 62279 – C Coding Guideline

IEC 62279 Table	IEC 62279 Coding Technique	IEC 62279 Rule
Table A.3 – Software Architecture	Defensive Programming	Variable & Parameter Check: Type
		Variable Check: Dimension & Range
		Parameter as Read-only
	Fault Detection & Diagnosis	Arithmetic errors
		Pointer arithmetic
		Array bound errors
		Null pointer dereferencing
Table A.4 – Software Design and Implementation	Modular Approach	a well defined function
	Strongly Typed Programming Language	the number and the type of arguments
		well-structured program of control structures
Table A.5 – Verification and Testing	Metrics	Graph Theoretic Complexity (Cyclomatic Complexity)
		Number of ways to activate a module (Cohesion)
		Software Science:program length (Halsted Metrics)
		Number of Entries and exit per Module (Coupling)
Table A.12 - Design and Coding Standards	Coding Style Guide	formatting and naming conventions
	Limited Use of Recursion	Recursive Call
	No Unconditional Jumps	Goto
Table A.19 - Static Analysis	Boundary Value Analysis	zero divisor
		ASCII characters
	Control Flow Analysis (Anlomaly)	Inaccessible code
		Knotted code
	Data Flow Analysis (Anlomaly)	Variables that are read before they are written
		Variables that are written more than once without being
		Variables that are written but never read
Table A.20 - Modular Approach	Module Size Limited	
	Parameter Number Limit	
	One Entry/One Exit Point in Subroutines and Functions	

3. 정적 분석 도구

정적 분석이란?

- 소스코드를 실행하지 않고 코드 결함을 찾을 수 있는 방법
- 조기 결함 발견으로 테스트 비용 및 시간 절감

●테스팅 이전에 50-90% 가량의 코드결함을 제거함으로써, 개발 공정의 10%~30%, 테스팅 비용 및 공정의 5-10배, 그리고 유지보수 비용의 2/3를 절감 효과가 있음 **(Software Inspection, Tom Glib & Dorothy Graham)**

Static Analysis vs. Testing Scope

	Static Analysis	Testing	Code Verification
Scope	Style violations		Syntactic Code Analysis
	Code inconsistencies		
	Portability issues		
	Security problems	Security problems	Semantic Code Analysis
	Run-time errors	Run-time errors	
		Violations of requirements	
		Performance problems	
Path Coverage	All Paths considered	High-density path coverage impossible, Coverage only as good as the test suite	
Results	Problems that might occur (=warning)	Problems that did manifest (=error)	
	May include false positives	No false positives (except in extreme circumstances)	

3. 정적 분석 도구

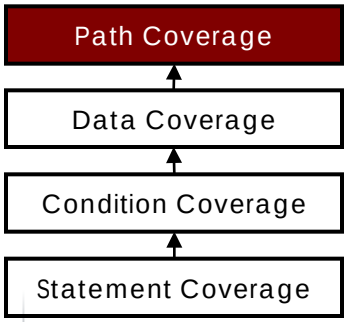
Static Analysis Tool

- **Original Static Analysis Tools**
  - **Syntactic Analysis (Data Flow Analysis) – 1세대**
    . Coding Standard Checks
    . High False-Positive & False Negative – less precise
    . (Speed) Faster - **Coffee Break**

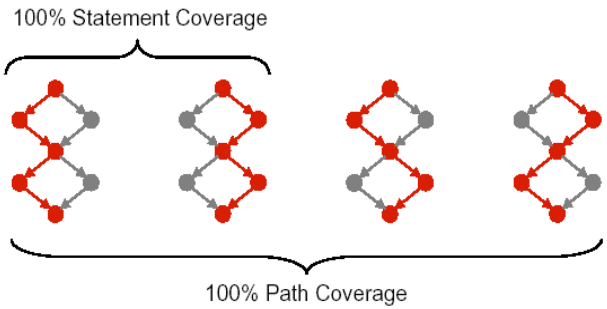
(Fatal Error)
arrays out of bounds,
buffer(integer) overflow,
division by zero,
null pointer dereference,
uninitialized Variable,
memory(resource, file) leak,
etc.

- **Advanced Static Analysis Tools – 현재 RESORT**
  - **Deep Semantic Analysis (Path Execution) – 3세대**
    . Run-time Error & Code Vulnerability Detection with Value Tracking
    . Low False-Positive – more precise
    . (Speed) Slower - **Overnight**

- 코드 검증 기술 분류 및 상용 도구 검증 기술 유형
  - 코드검증기술 차이로 제품기능 차이 발생



코드검증 기술

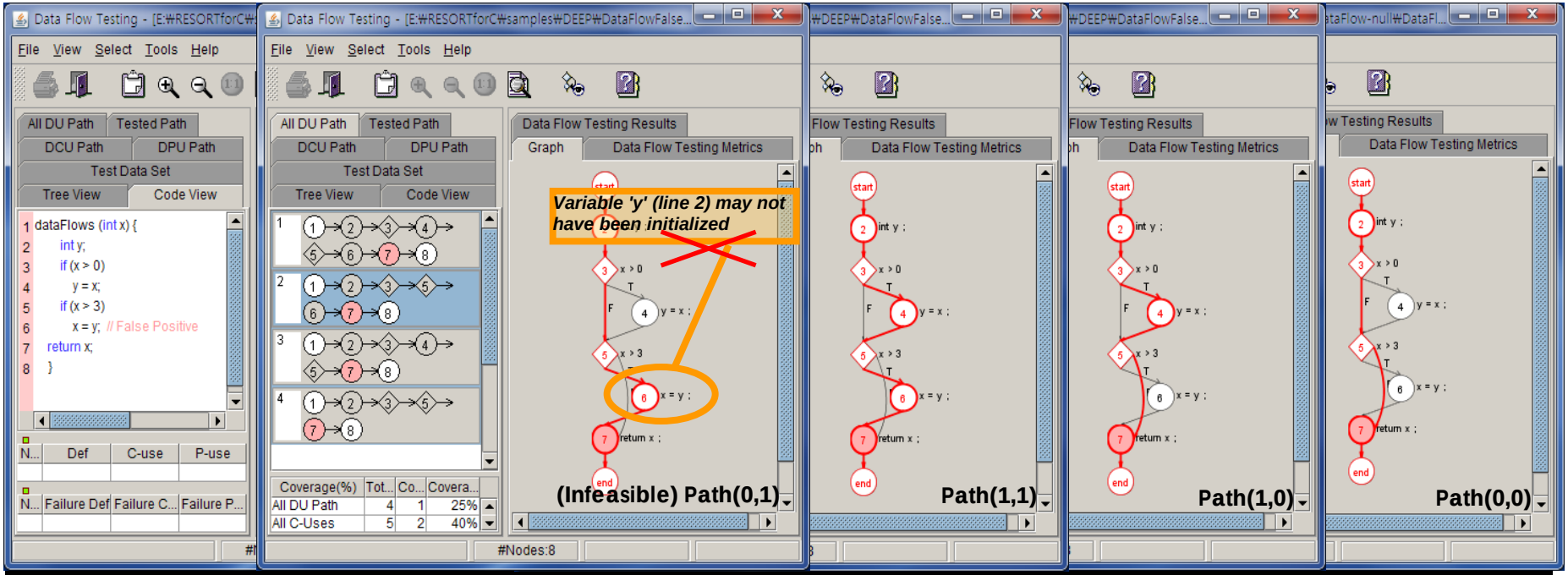


Verification Technology	Commercial Tool
Deep Semantic Analysis (Path Coverage) (외산도구 및 RESORT의 기술)	Path Simulation, Boolean Satisfiability
	X-Tier Dataflow
	Path Sensitive, Symbolic Execution
Syntactic Analysis (Condition Coverage)	Syntax, Flow Analysis (PMD, FindBug, etc.)

3. 정적 분석 도구

Static Analysis Tool

- **Deep Semantic Analysis**의 **Path Sensitive Data Flow Analysis** 기술
  - **1**세대 **Syntactic/Semantic(data flow analysis)** 검증 기술에서 코드 결함으로 보고된 오탐**(False Positive)** 해결 **– Uninitialized Variable**



Coverage	Code Inspection Results
Statement/ Condition Coverage	(Statement) int type의 변수 x, y의 값을 결정하고 수행하기 때문에 한번에 모든 경로를 테스트하지 않게 된다. 예를 들어 조건절 x,>0이 false, x>3이 true 값이라면, 위의 변수 y는 초기화 되지 않은 코드결함으로 보고할 수 있는 오탐을 발생한다: Path(0,1). (Condition) 개별 조건에 따라 True, False의 테스트가 가능하지만, 조건의 조합이 테스트되었는지는 확인할 수 없다.
Path Coverage	경로의 조합으로 입력될 수 있는 모든 전달 값의 범위를 계산한다. 즉 다음과 같이 경로를 계산하여 결과를 도출한다: •Feasible paths: (1,1), (1,0), (0,0); //Works fine, Infeasible Path: (0.1)

4. 결론

정적 분석 도구의 동향

특징	설명
통합 멀티 정적 분석 도구	• 현재까지, 정적 분석도구 유형은 크게 4가지 단독 도구의 시장으로 형성되고 있음: - (1) 코드 품질, (2) 코드 취약점, (3) Run-time Error, (4) 임베디드 표준 • 최근 추세는 통합 멀티 정적 분석 도구 출현 (1) 사용성 장점: 사용자는 하나의 정적 분석 도구 사용으로 결함 탐지의 시간 및 비용의 큰 절감 (2) 관리성 장점: 관리자는 통합 정적 분석의 룰 관리 및 소스 결함 분석 용이
통합 자동화 환경 구축	• 코드 품질 검증 및 보증 활동의 자동화를 위한 코드 품질 프로세스의 통합 환경을 구축하는 추세임, 즉 정적 분석 도구, 결함 관리 도구, 형상관리 도구를 통합하여 코드 품질 프로세스 구축하여 운영함 (1) 개발/운영 프로세스의 비용 및 시간 절감 (2) 정형화된 품질 검증 및 품질 보증 프로세스로 코드 품질 향상

[통합 정적 분석 도구와 품질 프로세스 자동화 환경]

RESORT는 국산 정적 분석 도구 분야의 선도 기업으로 12년 동안 코드 검증 기법의 전문 R&D를 하고 있으며,
IT 서비스, 은행, 증권, 보험, 전자, 의료, 통신, 군수, 그리고 국책연구소 등 다양한 분야에서 사용되고 있습니다.

감사합니다

Soft4Soft

RESORT Code Analysis