



Armのセキュリティと機能安全 つながる世界の 安心して安全なソフトウェアのテスト



Delivering Software Quality and Security through
Test, Analysis & Requirements Traceability

- 独語ではどちらも
Sicherheit (ズィツハーハイト)
- 日本語では安心 や セキュリティ
など、安全とは分けて表現される
が、モノがつながる時代に区別する
ことはできない
- セキュリティの脆弱性が、
機能安全上の潜在的な脅威になる



<http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/>

様々な領域で現実的な問題に、、

LDRA

Hackers are alleged to have destroyed a pump used to pipe water to thousands of homes in a US city in Illinois.

Hackers with access to the utility's network are thought to have broken the pump by turning it on and off quickly.

The FBI and Department for Homeland Security (DHS) are investigating the incident as details emerge of what could be a separate second attack.



The alleged attack was made on a system that piped clean water to homes in Illinois

上水道インフラ

<http://www.bbc.co.uk/news/technology-15817335>

Hack attack causes 'massive damage' at steel works

22 December 2016 Technology

プラントの溶鉱炉



The hack attack led to failures in plant equipment and forced the fast shut down of a furnace

<http://www.bbc.co.uk/news/technology-30575104>

JEFFREY M. HARRIS / GETTY IMAGES
SF'S TRANSIT HACK COULD'VE BEEN WAY WORSE—AND CITIES MUST PREPARE

交通機関



<https://www.wired.com/2016/11/sf-s-transit-hack-couldve-way-worse-cities-must-prepare/>

OPINION

Pacemaker hacker says worm could possibly 'commit mass murder'

ペースメーカー

It seems like something is very wrong with the picture when you read the news and it sounds more like a science fiction novel than a newsflash. For

MORE LIKE THIS

Pacemaker hack can deliver deadly 830-volt

http://www.computerworld.com/article/2473402/cybercrime-hacking/pacemaker-hacker-says-worm-could-possibly-commit-mass-murder-.html#tk.drr_mlt

- 仮に機能安全が求められていない製品であっても、つながる世界で他への攻撃の踏み台にされるようではブランド価値の低下を免れない



<https://www.informationweek.com/government/leadership/internet-of-things-8-cost-cutting-ideas-for-government/d/d-id/1113459>

情報セキュリティ早期警戒パートナーシップガイドライン - IPA

- 脆弱性とは、ソフトウェア製品やウェブアプリケーション等において、コンピュータ不正アクセスやコンピュータウイルス等の攻撃により、その機能や性能を損なう原因となり得るセキュリティ上の問題箇所

https://www.ipa.go.jp/security/ciadr/partnership_guide.html

国民のための情報セキュリティサイト - 総務省

- 脆弱性とは、コンピュータのOSやソフトウェアにおいて、プログラムの不具合や設計上のミスが原因となって発生した情報セキュリティ上の欠陥のこと

http://www.soumu.go.jp/main_sosiki/joho_tsusin/security/basic/risk/11.html



Wifi接続されるヘッドユニットへのパスワードを解読して侵入

ヘッドユニットにつながるCANバス上のコントローラのファームウェアを変更 (**権限が不要であった**)

CANバス上のコンポーネントを制御下に

Thinking Safety vs Security

- ・高品質であるとしても必ずしもセキュアなソフトウェアとは言えない
- ・セキュアなソフトウェアは現在知られている あるいは将来の脅威への防御のために高品質でなければならない
- ・高品質のソフトウェアを構築するために必要な同じ開発ライフサイクルの厳格さを要求する => **脆弱性の原因となるプログラムの不具合や設計上のミス**を排除

航空業界標準：LDRA社テストツール



コーディング
スタンダード
チェック

コード品質の
尺度

カバレッジ
解析

データフロー、
コントロールフ
ロー解析

単体テスト
統合テスト

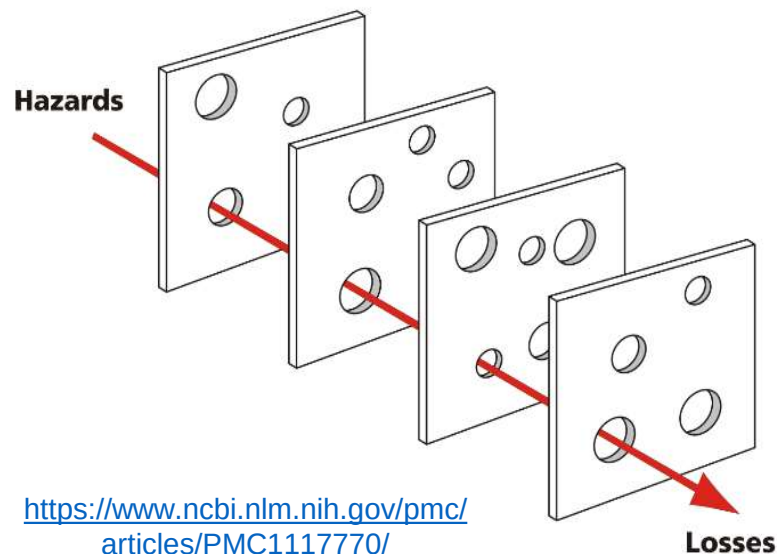
要件トレーサ
ビリティ

テストの管理

認証プロセス
の支援

最も厳密なことで知られる航空業界のDO-178スタンダード認証に標準的に採用されるテストツールの各種機能が防御の一翼を担う

セキュリティへの単一解は無く複数の対策（多重防御）が施される



スイスチーズモデル：事故は多重防護壁の穴をすべて貫通したときに生じる。穴は潜在的にも存在するし、即発的（**攻撃によって**）にも発生する。

事故を防ぐには穴の有無を常に監視し、発見したら直ぐに塞ぐ必要がある

- **セキュアブート**により正しいイメージがロードされる（不正な改ざんを検知）
- **ドメイン分離**でシステムのクリティカルな領域を保護
- **最小限の権限** (Least Privilege) **の設計原則**で脆弱性を最小限に抑える（Multiple Independent Levels of Securityのコアとなる原則）
- **攻撃対象領域を特定して最小限**に抑える
- セキュアコーディング標準
- セキュリティテスト

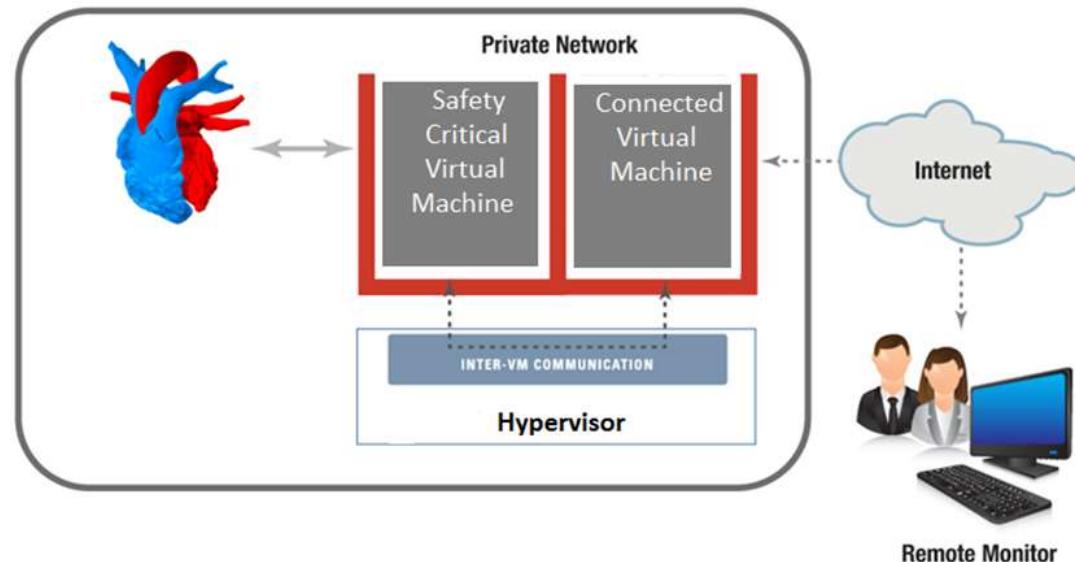
アクセスが困難で脆弱性が最小限に抑えられるように、コード自体が安全であることが重要

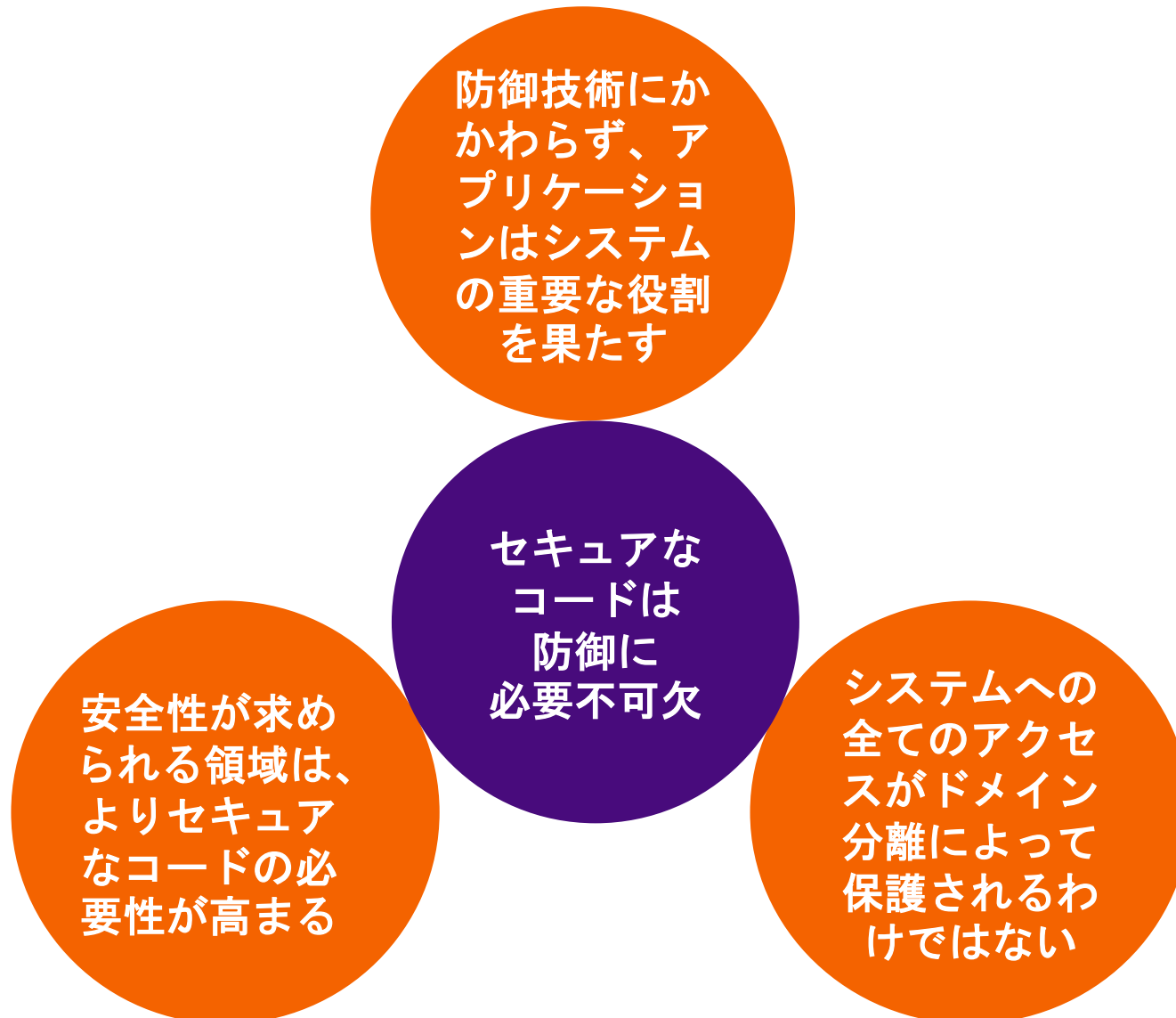
And after release

- 新たに発見される脆弱性に対処するためのレスポンスなメンテナンスプロセス

Separation Technologies are just one slice of “Swiss Cheese”!

- ハイパーバイザーなど仮想化による分離は有効な防御策ではあるもののセキュリティを保証する完全策ではない
 - ハイパーバイザードメインの分離は、仮想化を実装するプロセッサ（IntelのVT-xやArmの仮想化拡張など）に制限される
 - DMAエンジンやGPUなど、システム内の他のバスマスタは、ハイパーバイザによって提供される保護をバイパスできる
 - 分離されるドメイン間の通信にアプリケーションコードが必要





Code Reviews

- Secure Coding

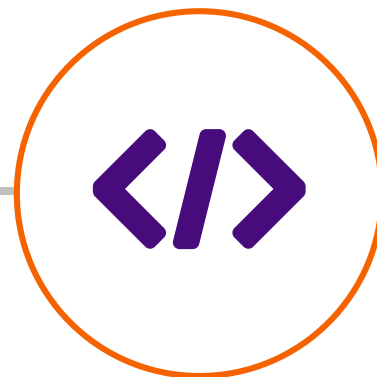
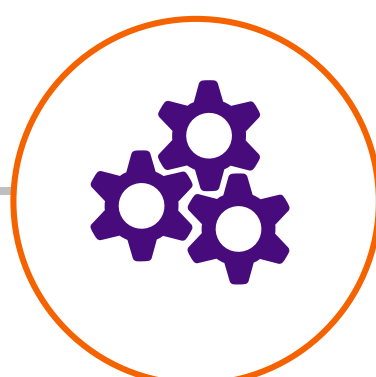


ソフトウェアの欠陥の約80%は、言語の約20%の誤った使用によって引き起こされる

言語の使用を制限することで、問題のあることが分かっているサブセットを避けることができるなら、ソフトウェアの品質は大幅に向上する

処理系定義の動作

コーディングエラー



未規定の動作 / 未定義の動作

ランタイムエラー

■ 共通脆弱性タイプ一覧 <https://cwe.mitre.org/>

National Institute of Security Technologyの調査によると、ソフトウェアの脆弱性の64%はプログラミングエラーに由来する。CWEプロジェクトは、ソフトウェアの欠陥をより深く理解し、その欠陥を特定、修正、防止するための自動ツールを作成することを目的とする。CWEの共通脆弱性リストは、**米国国土安全保障省**の国家サイバーセキュリティ部門が共同で主催するソフトウェア保証戦略イニシアチブの一環として作成される。

共通脆弱性タイプ一覧CWE概説 <https://www.ipa.go.jp/security/vuln/CWE.html>

699 - Development Concepts

- Location - (1)
- Motivation/Intent
- Resource-specific
- Weaknesses Intr
- Weaknesses Intr
- Weaknesses in C
- OWASP Top Te
- OWASP Top Te
- OWASP Top Te
- Improper Ne
- Improper Ne
- Improper Ne
- Improper Ne
- XML Injecti
- OWASP Top Te
- OWASP Top Te
- OWASP Top Te
- OWASP Top Te
- OWASP Top Te
- OWASP Top Te

1000 - Research Concepts

- Coding Standards Violation - (710)
- Hidden Functionality - (912)
- Improper Fulfillment of API Contract
- Indicator of Poor Code Quality - (396)
- Dead Code - (561)
- Empty Synchronized Block - (585)
- NULL Pointer Dereference - (415)
- Omitted Break Statement in
- Return of Stack Variable Add
- Struts: Unused Validation Fo
- Struts: Validator Without Fo
- Suspicious Comment - (546)
- Unused Variable - (563)
- Use of Function with Inconsi
- Use of Hard-coded, Security-
- Use of Obsolete Functions - (
- Use of Potentially Dangerous
- Reliance on Undefined, Unspec
- Use of Inherently Dangerous Fu

Detectable violation

- Improper Access of Indexable Resource ("Range Error") - (118)
- Improper Check or Handling of Exceptional Conditions - (703)
- Improper Control of a Resource Through its Lifetime - (664)
- Improper Enforcement of Message or Data Structure - (707)
- Incorrect Calculation - (682)
- Divide By Zero - (369)
- Incorrect Calculation of Buffer Size - (131)
- Incorrect Calculation of Multi-Byte String Length - (135)
- Incorrect Pointer Scaling - (468)
- Integer Overflow or Wraparound - (190)
- Integer Underflow (Wrap or Wraparound) - (191)
- Off-by-one Error - (193)
- Use of Pointer Subtraction to Determine Size - (469)
- Use of sizeof() on a Pointer Type - (463)

Statically Detectable Violation

- JPCERT/CCに日本語版公開

<https://www.jpcert.or.jp/sc-rules/>

- CERT's Top 10 Secure Coding Practices

IPAのセキュア・プログラミング講座（P12~）

<https://www.ipa.go.jp/files/000059838.pdf>

- By Robert C. Seacord



DARPA(《米》国防総省国防高等研究事業局)が中心となってカーネギーメロン大学内に設置されたCERT/CC (Computer Emergency Response Team/Coordination Center) により公開



Software Engineering Institute
Carnegie Mellon

MISRA C:2012 Addendum 2

ISO/IEC TS 17961:2013 の「Cセキュアコーディング」に対するMISRA C:2012のカバレッジ

x46 の C SecureルールのどれがMISRA C:2012のガイドラインの対象であるかを示す

MISRA C:2012 Amendment 1

ISO C セキュアガイドラインへのカバレッジを強化する
14の新しいCコーディングルール

特に「信頼されていないデータの使用」というセキュリティ上の脆弱性に付き物の具体的な問題へ取り組む

セキュアでないコーディングの例

https://www.fuji-setsu.co.jp/files/MISRA_C_2012_AMD1.pdf

■ IPA/SECの組み込みソフトウェア開発向けコーディング 作法ガイド ESCR [C言語版] Ver.3.0 の発刊 ～セキュアコーディングへの対応～

Results View				
Available Results ☒ Code Review : demo_set : C - SEC-Cv3 Model ☒				
	Number Viola	Level of Viola	Phase Code	Standard Code
▼ demo_set				
▼ demo1.c				
◆ Global variable should be declared const. : result		Mandatory	78 D	SEC-Cv3 M1.11.1,M1.11.3
◆ Variable should be declared static. : result		Mandatory	27 D	SEC-Cv3 M2.2.2
◆ Basic type declaration used. : unsigned int result (analysed w. MISRA-...		Mandatory	90 S	SEC-Cv3 P2.1.3
◆ Macro not used in translation unit. : FILE		Mandatory	628 S	SEC-Cv3 M1.1.1
◆ #define of keyword. (analysed w. MISRA-C:2012)		Mandatory	626 S	SEC-Cv3 M1.8.2
◆ Included file not protected with #define.		Mandatory	243 S	SEC-Cv3 M4.4.6
▼ Header Files				
▼ demo1.h				
> ◆ Macro parameter not in brackets.	2	Mandatory	78 S	SEC-Cv3 M4.7.1
> ◆ Macro contains unacceptable items.	2	Mandatory	79 S	SEC-Cv3 M1.8.2
◆ Basic type declaration used. : unsigned int (analysed w. MISRA-...		Mandatory	90 S	SEC-Cv3 P2.1.3
◆ Macro not used in translation unit. : MYADD		Mandatory	628 S	SEC-Cv3 M1.1.1
◆ Macro replacement list needs parentheses.		Mandatory	77 S	SEC-Cv3 M4.7.1
> ◆ Use of function like macro.	4	Optional	340 S	SEC-Cv3 E1.1.1,M5.1.3
▼ preferred_cube				
◆ Procedure should be declared static. : preferred_cube		Mandatory	61 D	SEC-Cv3 M2.2.3
▼ preproc_ex				
> ◆ Local variable should be declared const.	6	Mandatory	93 D	SEC-Cv3 M1.11.1,M1.11.3
> ◆ Global denominator not checked within this procedure.	3	Mandatory	131 D	SEC-Cv3 R3.2.1
◆ Procedure should be declared static. : preproc_ex		Mandatory	61 D	SEC-Cv3 M2.2.3
◆ Function return value potentially unused. : sa		Mandatory	91 D	SEC-Cv3 R3.3.1
◆ DU anomaly dead code, var value is unused on all paths. : sa		Mandatory	105 D	SEC-Cv3 M1.1.1,X1.x
◆ DU anomaly, variable value is not used. : sa		Mandatory	70 D	SEC-Cv3 M1.1.1,X1.x
◆ Local or member denominator not checked before use. : preferred...		Mandatory	127 D	SEC-Cv3 R3.2.1
◆ Comment possibly contains code.		Mandatory	302 S	SEC-Cv3 M1.1.2
◆ Function call with no prior declaration. : square		Mandatory	496 S	SEC-Cv3 M4.4.3,R2.8.3

Coding Standards Models



LDRA Report File Editor 9.7.0 © 2017 LDRA Ltd - C:\LDRA_Toolsuite\c\creport.dat

File Edit View Help

Static Complexity Data Flow Cross Ref Info Flow Qual Report Qualsys LOSAJ Hungarian Notation User Def Section 3 Section 4 PDTMCSFVA

Rule Number	Default Strength	Description	CERT-C	CERT-C 2014	CWE	MISRA-C 2012/ADD2	MISRA-C 2012/AMD1	MISRA-C 2012/AMD1/ADD2	SEC-Cv1	SEC-Cv2	secureC
64	M	Array bound exceeded at call.	☒	☒	☒	☒	☒	☒	☒	☒	☒
65	M	continue in ill-formed loop (MR).	☐	☐	☐	☐	☐	☐	☐	☐	☐
66	M	Insufficient array space at call.	☒	☒	☒	☒	☒	☒	☐	☐	☒
67	M	Identifier is typographically ambiguous.	☒	☒	☒	☒	☒	☒	☐	☐	☐

LDRA Report File Editor 9.7.0 © 2017 LDRA Ltd - C:\LDRA_Toolsuite\c\creport.dat

File Edit View Help

Static Complexity Data Flow Cross Ref Info Flow Qual Report Qualsys LOSAJ Hungarian Notation User Def Section 3 Section 4 PDTMCSFVA

Rule Number	Default Strength	Description	CERT-C	CERT-C 2014	CWE	MISRA-C 2012/ADD2	MISRA-C 2012/AMD1	MISRA-C 2012/AMD1/ADD2	SEC-Cv1	SEC-Cv2	secureC
46	M	extern not in nominated include file.	☐	☐	☐	☐	☐	☐	☐	☐	☐
47	M	Array bound exceeded.	☒	☒	☒	☒	☒	☒	☒	☒	☒
48	M	No default case in switch statement.	☒	☒	☐	☒	☒	☒	☒	☒	☒
49	M	Logical conjunctions need brackets.	☒	☒	☒	☒	☒	☒	☒	☐	☐
50	M	Use of shift operator on signed type.	☒	☒	☒	☒	☒	☒	☒	☐	☐

LDRA Report File Editor 9.7.0 © 2017 LDRA Ltd - C:\LDRA_Toolsuite\c\creport.dat

File Edit View Help

Static Complexity Data Flow Cross Ref Info Flow Qual Report Qualsys LOSAJ Hungarian Notation User Def Section 3 Section 4 PDTMCSFVA

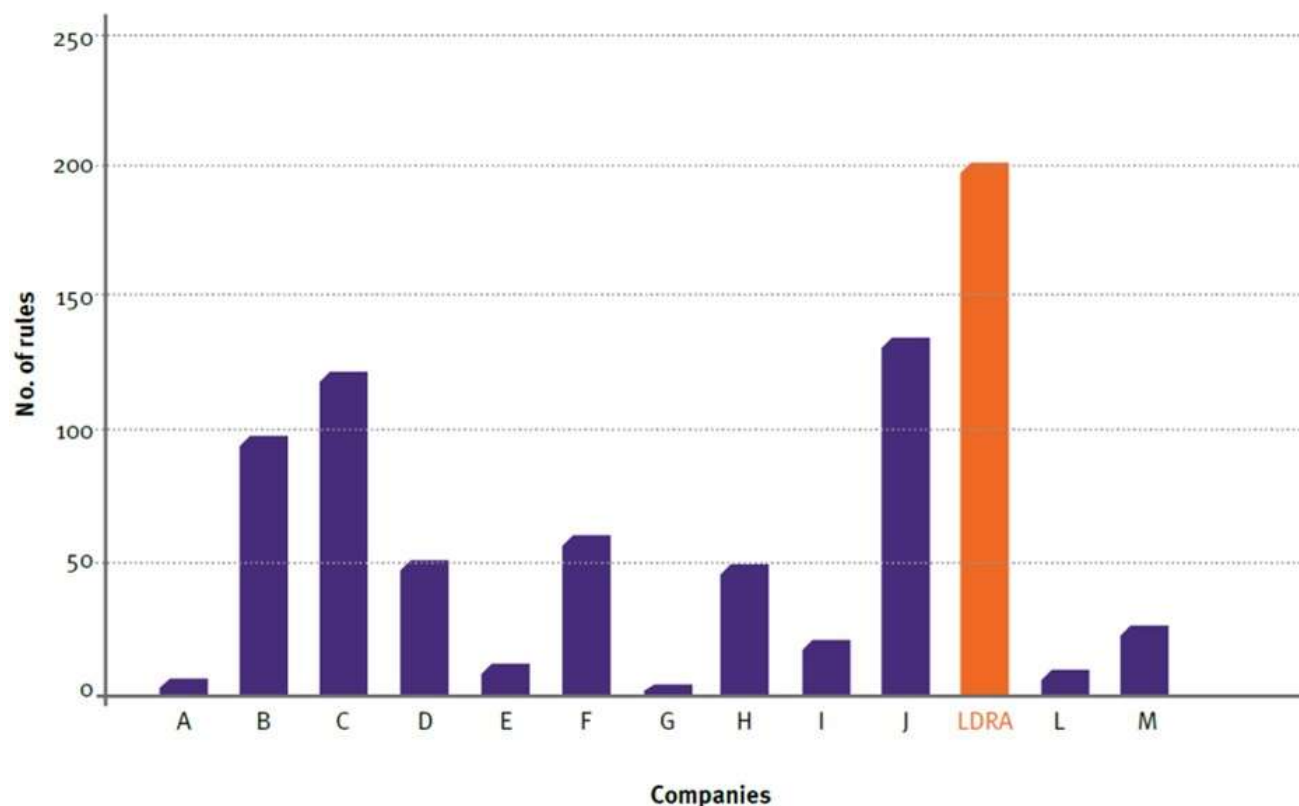
Rule Number	Default Strength	Description	CERT-C	CERT-C 2014	CWE	MISRA-C 2012/ADD2	MISRA-C 2012/AMD1	MISRA-C 2012/AMD1/ADD2	SEC-Cv1	SEC-Cv2	secureC
42	M	Local pointer returned in function result.	☒	☒	☒	☒	☒	☒	☒	☒	☒
43	M	Divide by zero found.	☒	☒	☒	☒	☒	☒	☒	☒	☒
45	M	Pointer not checked for null before use.	☒	☒	☒	☒	☒	☒	☒	☒	☒
47	M	Unused inspect annotation.	☐	☐	☐	☐	☐	☐	☐	☐	☐
48	M	Attempt to write to unopened file.	☒	☒	☒	☒	☒	☒	☐	☐	☐
49	M	File pointer not closed on exit.	☒	☒	☒	☒	☒	☒	☐	☐	☒
50	M	Memory not freed after last reference.	☒	☒	☒	☒	☒	☒	☐	☐	☒

ルール集の編集も容易

17

CERT 対応で他を圧倒！

LDRA



Tool	No. of rules
A	7
B	96
C	122
D	50
E	11
F	63
G	3
H	48
I	21
J	135
LDRA tool suite	201
L	10
M	26

LDRAはシステムワイドにコントロールフローとデータフローを解析できることが優位性のひとつ

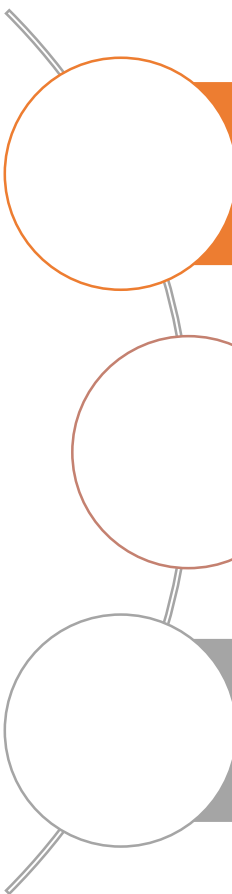
LDRA

19

Code Reviews

- Keep it simple
- Use effective quality assurance





可読性が低下して脆弱性が奥に潜んでしまう

テストは容易で無くなり余計な作業が必要

保守性が低く新たな脆弱性への迅速な対応が困難

コード品質を自動解析



Results View

Quality Review : Mingw_c_cashregister

	Value	Lower Limit	Upper Limit
Mingw_c_cashregister			
Tester.c			
Clarity	64% Metrics Successful		
Maintainability	90% Metrics Successful		
Testability	93% Metrics Successful		
Metric Groupings			
goodbye			
Clarity	50% Metrics Successful		
Maintainability	100% Metrics Successful		
Testability	100% Metrics Successful		
Metric Groupings			
randomShopping			
Clarity	40% Metrics Successful		
Depth of Loop Nesting	1	0	2
Average Length of Basic Blocks	3.57%	1%	6%
Code Comments/Exe. Lines	0 : (Fail)	5	200
Declaration Comments/Exe. Lines	0 : (Fail)	1	100

Metrics for an entire file

Metrics for a single function

Clarity 可読性 100%

Expansion Factor	1.3%
Executable ref. Lines	147
Depth of Loop Nesting	1
Total LCSAJs	46
Unique Operands	105
Average Length of Basic Blocks	4.03%

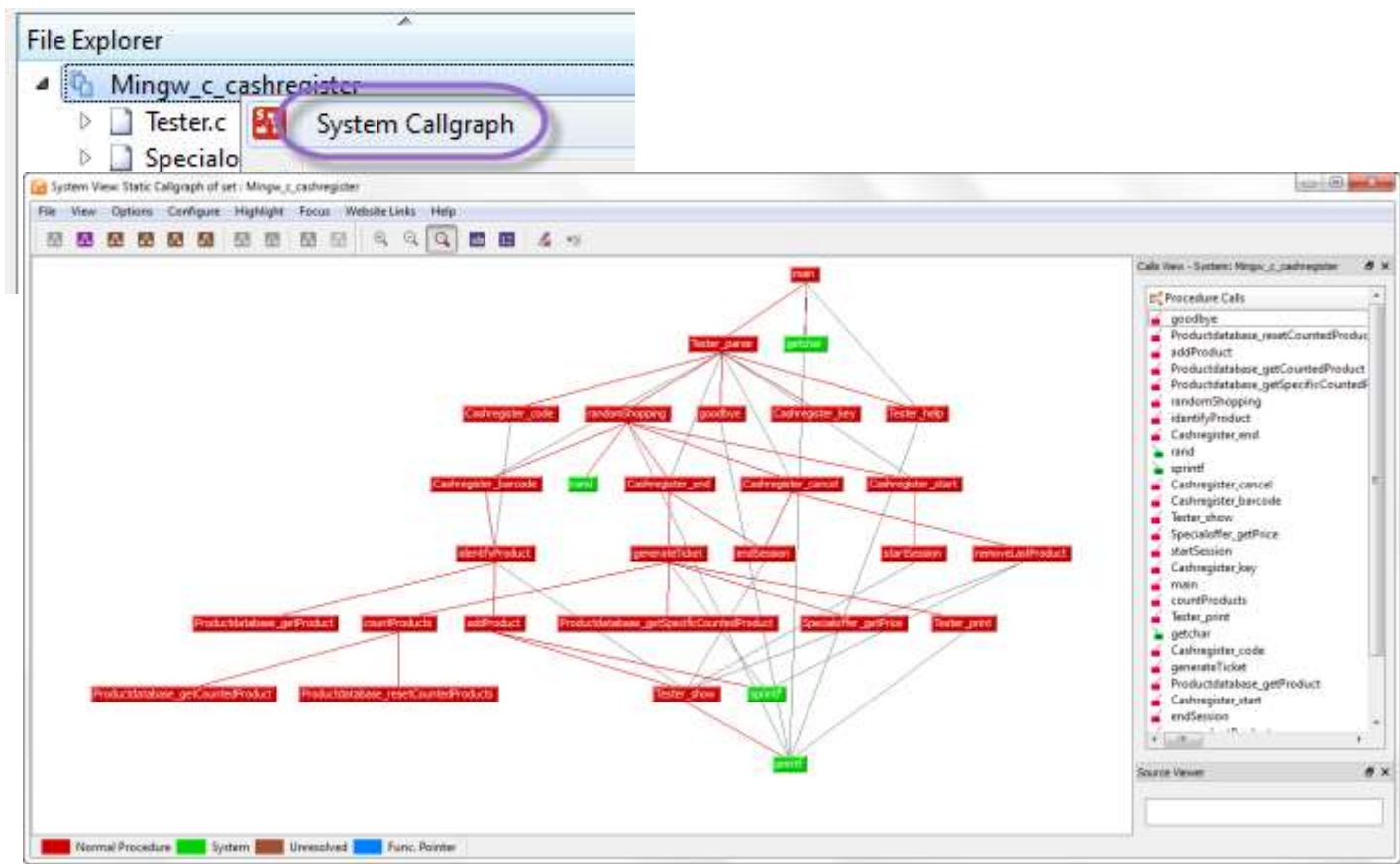
Maintainability メンテナンス性 100%

Number of Procedures	7
Unreachable Branches	0
Unreachable Lines	0
Maximum LCSAJ Density	4
Unreachable LCSAJs	0
Total LCSAJs	46
Vocabulary	119
Cyclomatic Complexity	9
Knots	23
Essential Cyclomatic Complexity	1
Essential Knots	0

Testability テスト容易性 100%

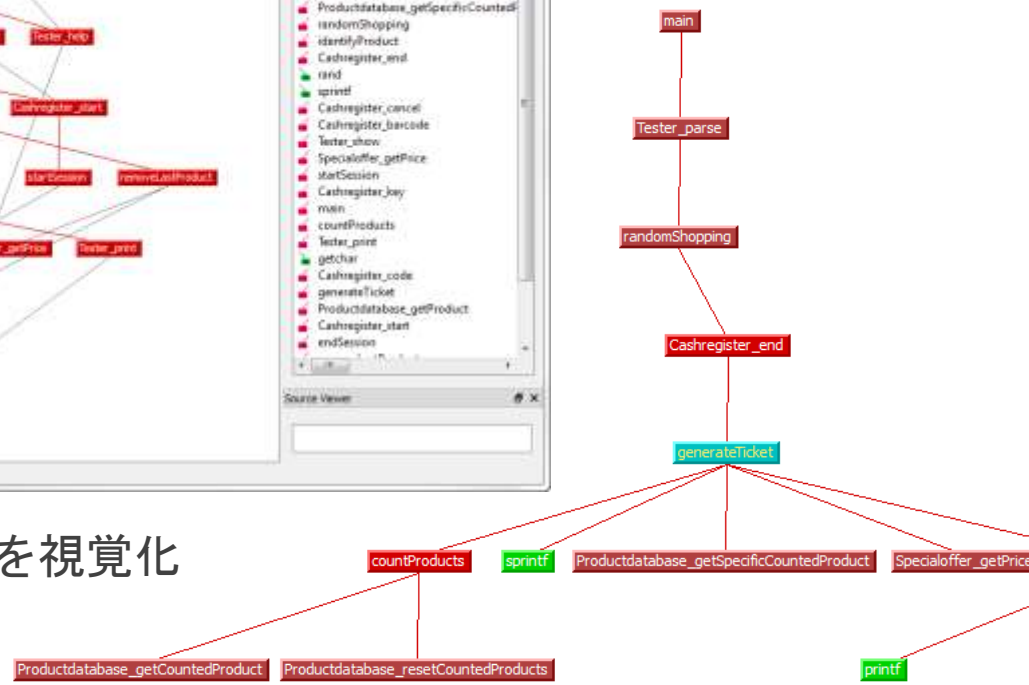
Fan Out	7
File Fan in	0
Number of Procedures	7
Procedure Exit Points	1
Number of Loops	5
Unreachable Branches	0
Unreachable Lines	0
Maximum LCSAJ Density	4
Unreachable LCSAJs	0
Total LCSAJs	46
Total Operands	286
Number of Basic Blocks	36
Executable reformatted Lines	145
Cyclomatic Complexity	9
Knots	23

コントロールカップリング



- Tester_
 - Specia
 - startS
 - Cashre
 - main
 - countP
 - Tester_
 - getcha
 - Cashre
 - genera
 - Produc
 - Cashre
- Control Coupling Graph
 - Extended Control Coupling Graph
 - Highlight Parents
 - Highlight Ancestors
 - Highlight Children
 - Highlight Descendents
 - Exclude System Calls
 - Restore Original Graph

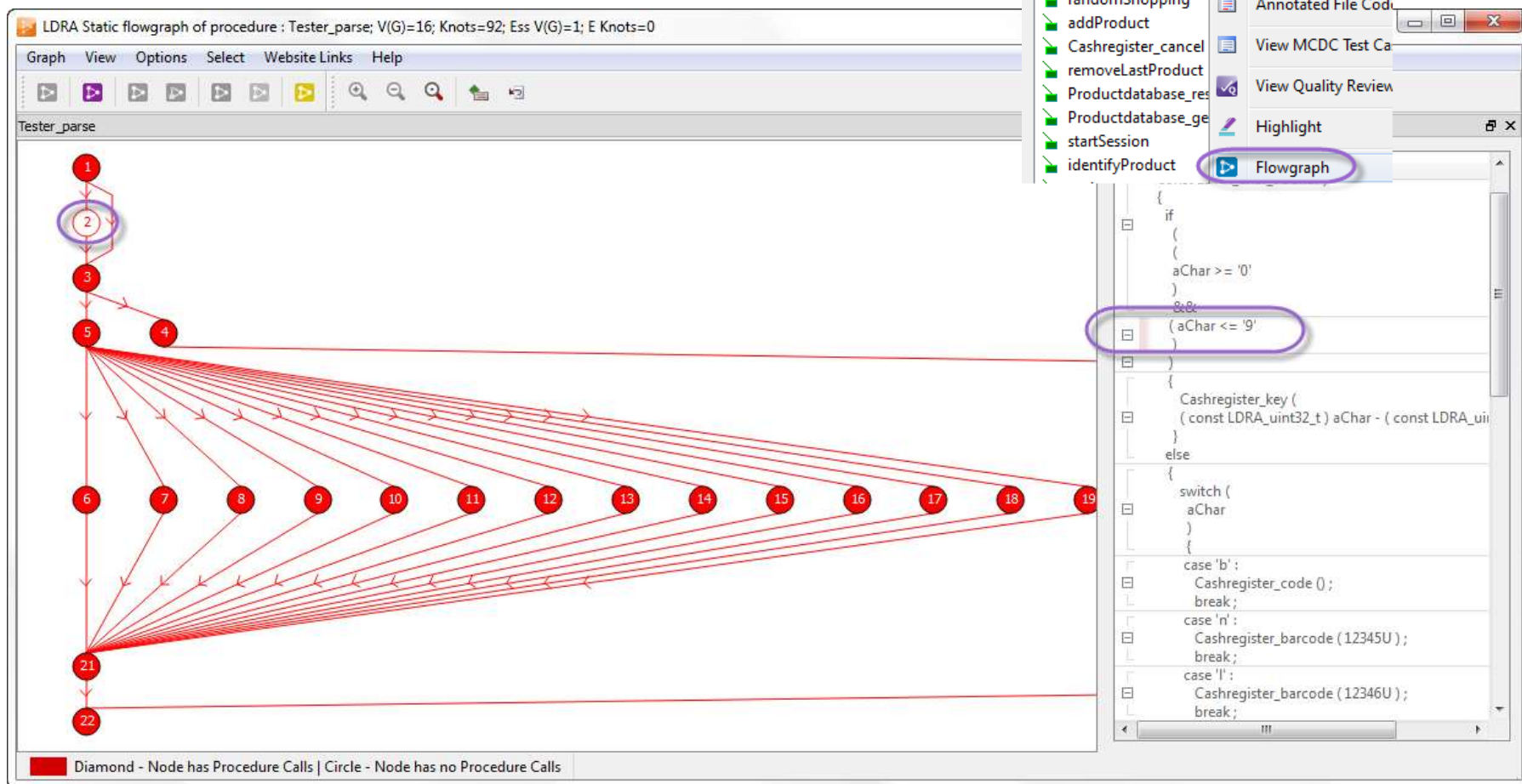
System Callgraph 関数間の相互関係を視覚化



Extended Control Coupling Graph 特定関数にフォーカス

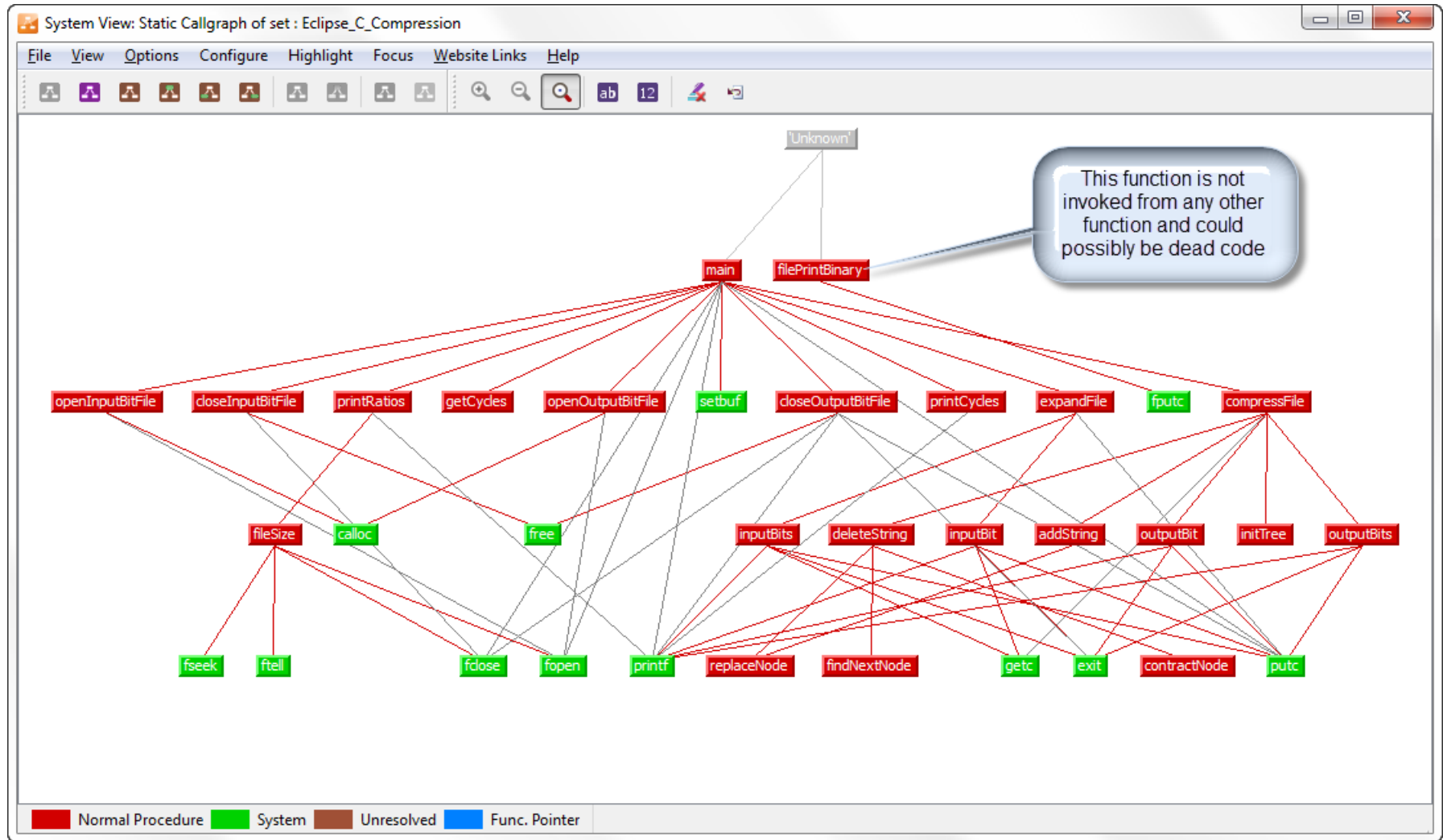
フローグラフ

- フローグラフ内のブロックやブランチに該当するコード領域をハイライト表示



Static Callgraph – デッドコードの検出

LDRA

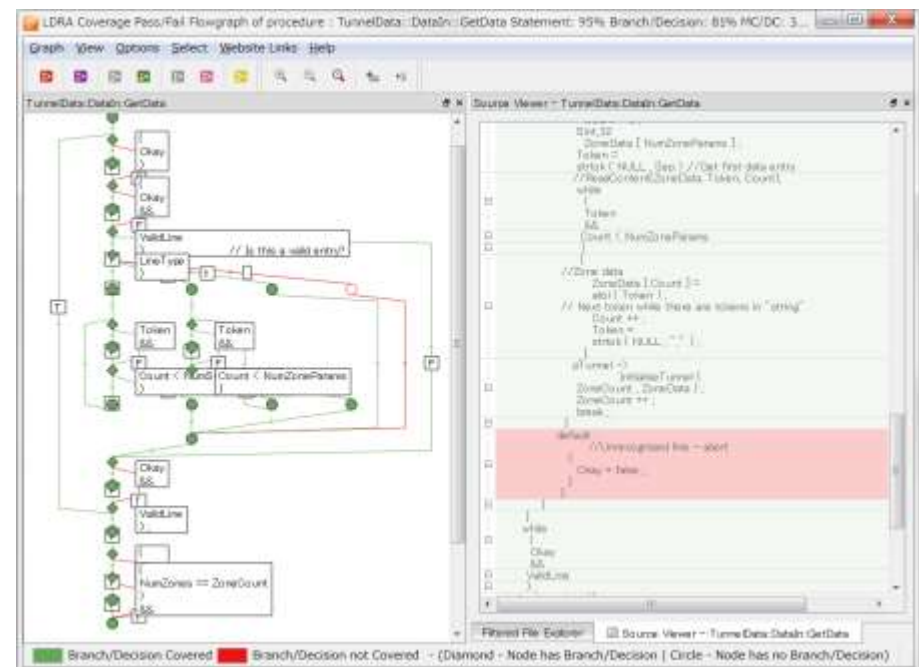


Test Coverage



テストのカバレッジ

- セキュアコーディングの実践により「防御的」ソフトウェアを得られたとして、それが動作することをテストする必要がある
- そのテストの十分性は？
- 安全性レベルに応じた各種カバレッジレベルに対応
 - Entry points coverage
 - Statement coverage
 - Branch decision coverage
 - Modified Condition / Decision Coverage (MC/DC)
 - Data coupling and control coupling coverage
 - Object code coverage
 - Linear Code Sequence And Jump coverage (LCSAJ)



**Common Weakness Enumeration**
A Community-Developed List of Software Weakness Types

CWE and SANS Institute
TOP 25
MOST DANGEROUS
SOFTWARE
ERRORS

Home > CWE List > CWE: Individual Dictionary Definition (2.11)

ID Lookup:

Home | About | CWE List | Scoring | Community | News | Search

CWE-912: Hidden Functionality

Weakness ID: 912
Abstraction: Class

Status: Incomplete

Presentation Filter:

Description

Description Summary
The software contains functionality that is not documented, not part of the specification, and not accessible through an interface or command sequence that is obvious to the software's users or administrators.

Extended Description
Hidden functionality can take many forms, such as intentionally malicious code, "Easter Eggs" that contain extraneous functionality such as games, developer-friendly shortcuts that reduce maintenance or support costs such as hard-coded accounts, etc. From a security perspective, even when the functionality is not intentionally malicious or damaging, it can increase the software's attack surface and expose additional weaknesses beyond what is already exposed by the intended functionality. Even if it is not easily accessible, the hidden functionality could be useful for attacks that modify the control flow of the application.

Common Consequences

Scope	Effect
Other	Technical Impact: <i>Varies by context; Alter execution logic</i>
Integrity	

Potential Mitigations

Phase: Installation
Always verify the integrity of the software that is being installed.

Phase: Testing
Conduct a code coverage analysis using live testing, then closely inspect any code that is not covered.

Relationships

Nature	Type	ID	Name
ChildOf	❌	505	Intentionally
ChildOf	✅	710	Coding Stan
ParentOf	✅	506	Embedded Malicious Code
ParentOf	✅	514	Covert Channel

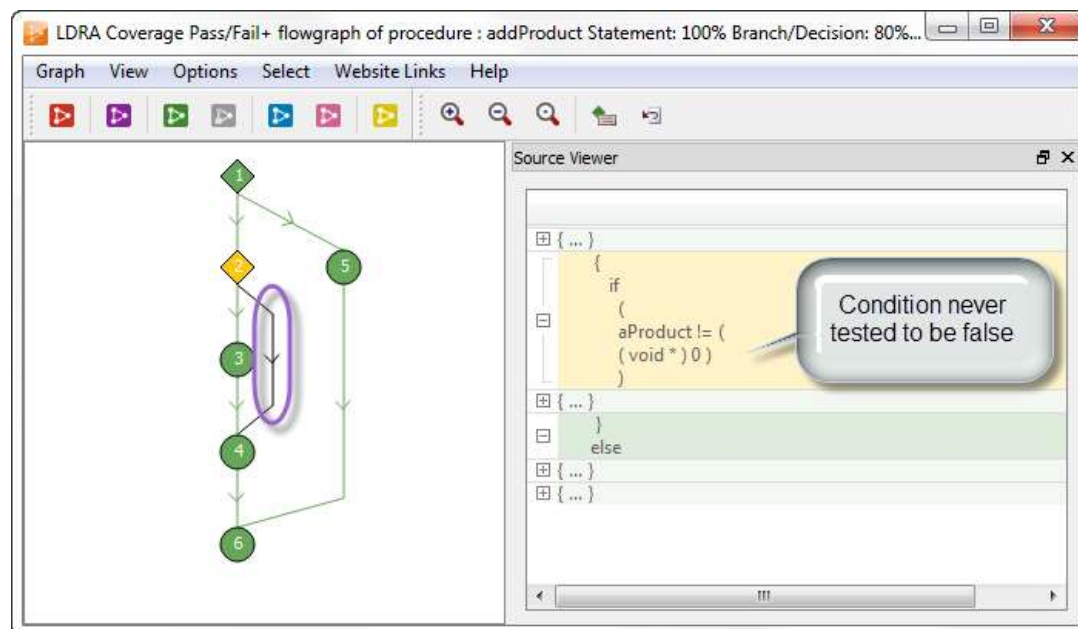
699
1000
1000
1000

隠れた機能は、アプリケーションの攻撃面を増加させ、意図した機能を弱体化させかねない

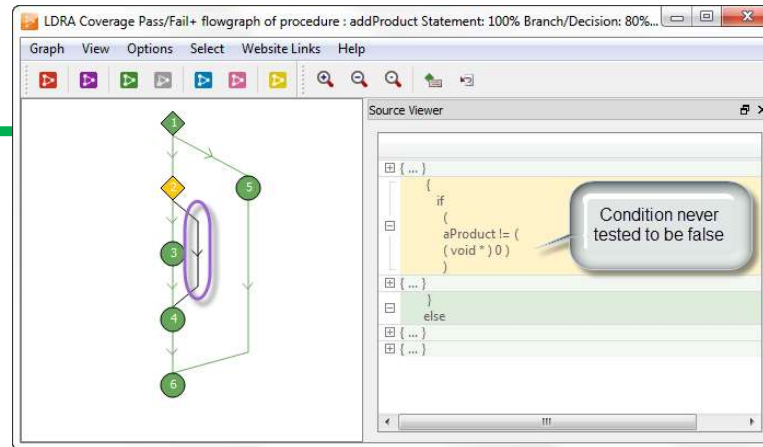
対策として、構造化カバレッジにより、実行されていないソースコードを検出することによって隠された機能を突き止めることができる

<https://cwe.mitre.org/data/definitions/912.html>
<http://jvndb.jvn.jp/ja/contents/2015/JVND-2015-006032.html>

- 一般に、例外的な処理をするコードは十分にテストされていない可能性が高く、めったにない条件下で、安全性やセキュリティの問題を引き起こすことになる
- 例えばポインタがNULLでない場合など、ディフェンシブなプログラムはシステムテストでは確認できない
- この例で100%カバレッジを満たすには単体テストによる評価が必要



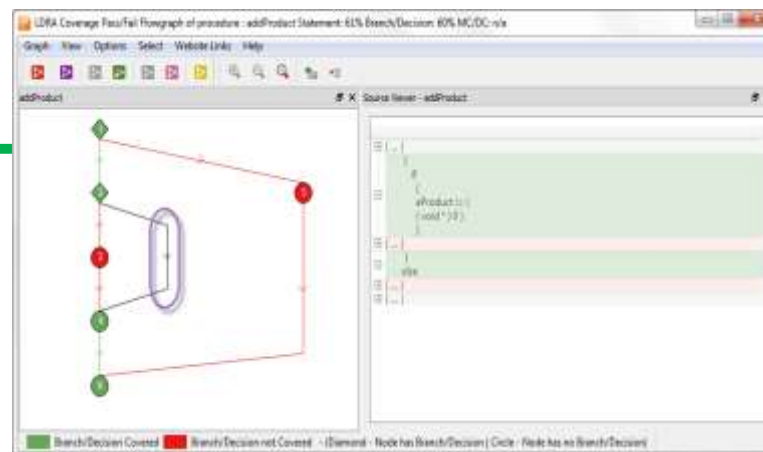
システムレベルテストで満たされないパスを



HLR_0001
HLR_0002
.....
HLR_000n

HLT_0001
HLT_0001
.....
HLT_000m

単体テストで満たす



LLR_0001
LLR_0002
.....
LLR_000n

LLT_0001
LLT_0001
.....
LLT_000m

- ✦ Table B.2.7d - Structural test coverage (conditions, MC/DC) 100 % - Partial - 2 artifacts
 - 📁 Code coverage report fulfilled by 1 item
 - 📄 Mingw_c_cashregister.frm.htm
 - 📁 MC/DC test plan report fulfilled by 1 item
 - 📄 MCDC_Planner_Report.txt

Branch Condition List

CONDITION NUMBER	LINE NUMBER REF. (SOURCE)	CONDITION TEXT
C1	3731 (1556)	repLen >= 2
C2	3734 (1557)	(repLen + 1 >= mainLen)
C3	3739 (1558)	repLen + 2 >= mainLen
C4	3741 (1558)	mainDist >= (1 << 9)
C5	3745 (1559)	repLen + 3 >= mainLen
C6	3747 (1559)	mainDist >= (1 << 15)

Full Truth Table For Decision

INDEX	C1	C2	C3	C4	C5	C6	EXPECTED OUTCOME	MC/DC INDEPENDENT PAIRS WAVE
1	F	T	T	T	T	T	T	
2	F	T	T	T	T	F	T	
3	F	T	T	T	F	T	T	
4	F	T	T	T	F	F	T	C1
5	F	T	T	F	T	T	T	C1
6	F	T	T	F	T	F	T	C1, C2
7	F	T	T	F	F	T	T	C1, C2
8	F	T	T	F	F	F	T	C1, C2
9	F	T	F	T	T	T	T	C1, C2
10	F	T	F	T	T	F	T	C1, C2
11	F	T	F	T	F	T	T	C1, C2
12	F	T	F	T	F	F	T	C1, C2
13	F	T	F	F	T	T	T	C1, C2
14	F	T	F	F	T	F	T	C1, C2
15	F	T	F	F	F	T	T	C1, C2
16	F	T	F	F	F	F	T	C1, C2
17	F	T	F	T	T	T	T	C1, C2
18	F	T	F	T	T	F	T	C1, C2
19	F	T	F	T	F	T	T	C1, C2
20	F	T	F	T	F	F	T	C1, C2
21	F	T	F	F	T	T	T	C1, C2
22	F	T	F	F	T	F	T	C1, C2
23	F	T	F	F	F	T	T	C1, C2
24	F	T	F	F	F	F	T	C1, C2
25	F	T	F	T	T	T	T	C1, C2
26	F	T	F	T	T	F	T	C1, C2
27	F	T	F	T	F	T	T	C1, C2
28	F	T	F	T	F	F	T	C1, C2
29	F	T	F	F	T	T	T	C1, C2
30	F	T	F	F	T	F	T	C1, C2
31	F	T	F	F	F	T	T	C1, C2
32	F	T	F	F	F	F	T	C1, C2
33	F	T	T	T	T	T	T	C1, C2
34	F	T	T	T	T	F	T	C1, C2

MC/DC 100%に必要なテストケースをレポート

Test for Security Requirements

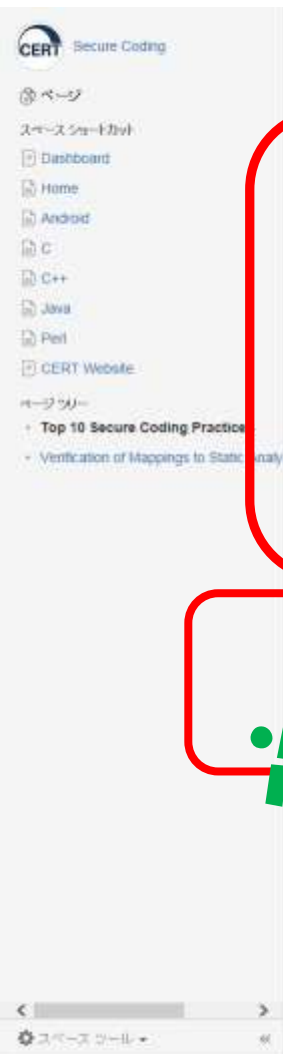


すべての機能、安全およびセキュリティの要件が
正しく実装され検証されることを保証するメカニズム

Security Requirements
Safety Requirements

要件にはないコードの存在が欠陥や脆弱性の原因になる
ことは多い。利用されないなら削除されるべき

双方向のトレーサビリティにより、すべての要件を満た
すためのコードが存在することが保証される



ページ / SEI CERT Coding Standards

Top 10 Secure Coding Practices

Robert Seacord 2011年 3月 2011年 Robert Seacord (Manager) 10番目更新

Top 10 Secure Coding Practices

1. **Validate input.** Validate input from all untrusted data sources. Proper input validation includes validating environmental variables, and user-controlled files [Seacord 05].
2. **Heed compiler warnings.** Compile code using the highest warning level available to catch additional security flaws.
3. **Architect and design for security policies.** Create a software architecture and design that partitions the system into distinct intercommunicating subsystems, each with an appropriate privilege level.
4. **Keep it simple.** Keep the design as simple and small as possible [Saltzer 74]. Safer designs are those that increase dramatically as security mechanisms become more complex.
5. **Default deny.** Base access decisions on permission rather than exclusion. This is the opposite of the "allowlist" approach.
6. **Adhere to the principle of least privilege.** Every process should execute with the minimum privileges necessary to perform its function. If an attacker has to execute arbitrary code with elevated privileges [Saltzer 74], Saltzer's principle of least privilege is violated.
7. **Sanitize data sent to other systems.** Sanitize all data passed to complex subsystems or external systems through the use of SQL, command, or other input made. Because the calling process understands the context, it is responsible for sanitizing the data.
8. **Practice defense in depth.** Manage risk with multiple defensive strategies, so the consequences of a successful exploit. For example, combining secure programming practices with secure coding standards.
9. **Use effective quality assurance techniques.** Good quality assurance techniques include static analysis, code reviews, and testing. Independent security reviews can lead to more secure code.
10. **Adopt a secure coding standard.** Develop and/or apply a secure coding standard.

Bonus Secure Coding Practices

1. **Define security requirements.** Identify and document security requirements early in the development life cycle and make sure that subsequent development artifacts are evaluated for compliance with those requirements. When security requirements are not defined, the resulting system cannot be effectively evaluated.
2. **Model threats.** Identify the threats to which the software will be subjected. Threat modeling involves identifying key assets, decomposing the application, identifying and categorizing the threats to each asset or component, and then developing threat mitigation strategies that are implemented in designs, code, and test cases [Swiderski 04].

I found the following photograph on the Web, and I'm still trying to figure out who owns the rights to it. If you know, please comment below.



1. 入力を検証する
2. コンパイラの警告に注意を払う
3. セキュリティポリシーのための設計
4. シンプルを維持する
5. 拒否をデフォルトにする
6. 最小権限の原則に従う
7. 外部に渡すデータを無害化する
8. 多重防御を行う
9. 効果的な品質保証テクニックを使う
10. セキュアコーディング標準を採用する

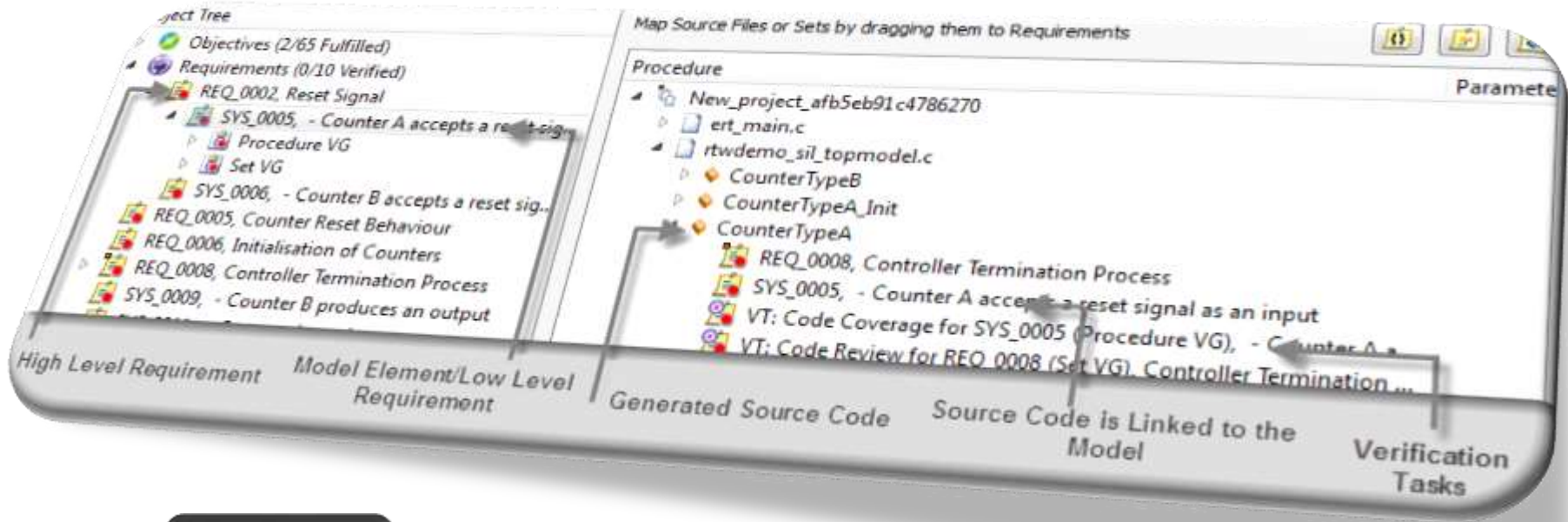
次のスライドで

セキュリティ要求を定義する

- 開発ライフサイクルの早期段階でセキュリティ要件を特定し文書化して、後続の開発成果物がそれら要件に従っていることを評価する。セキュリティ要件が定義されない場合、そのシステムのセキュリティを効果的に評価することはできない。

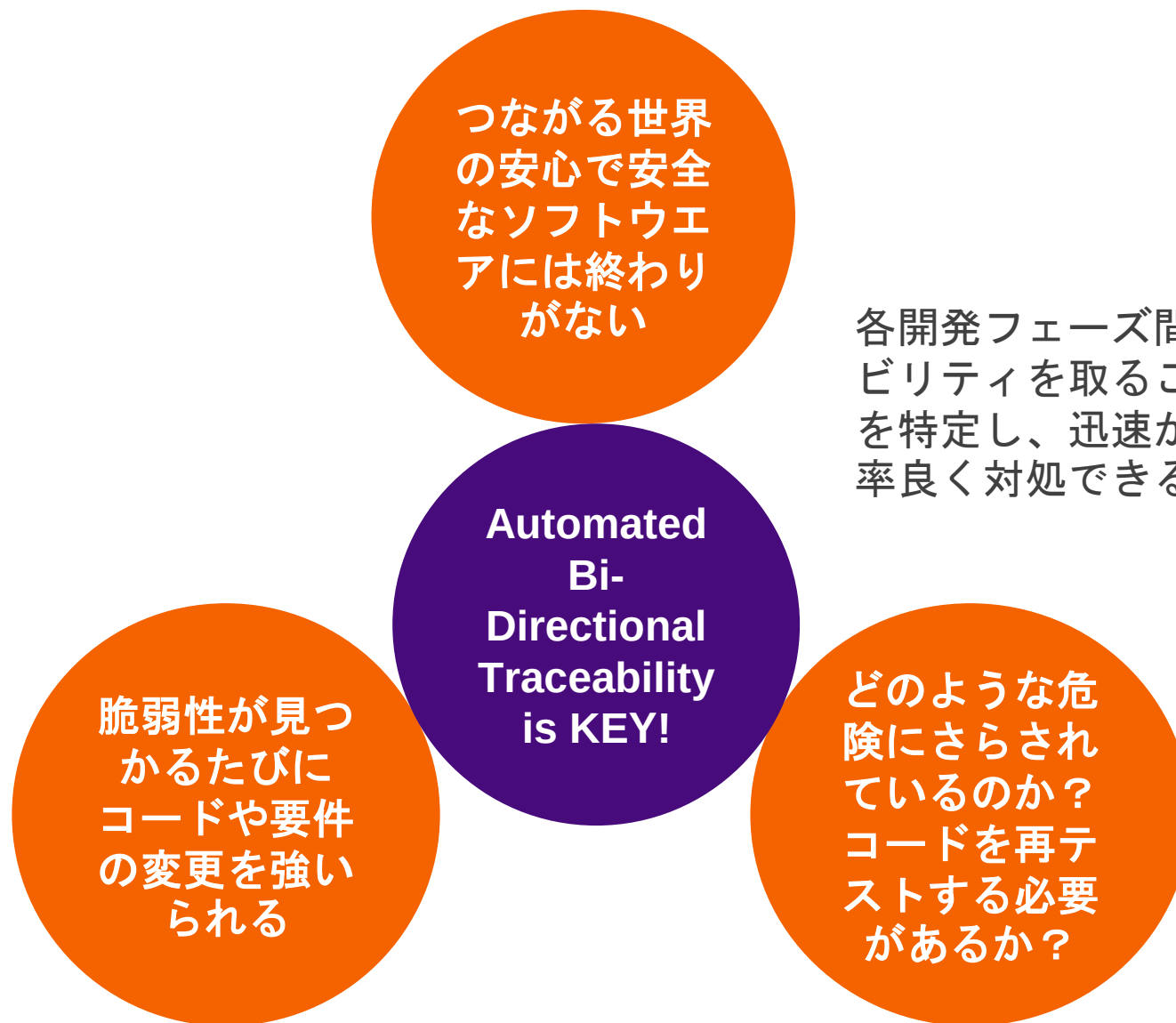
脅威をモデル化する

- 脅威をモデリングすることで、ソフトウェアが受ける脅威を予測する。脅威のモデリングでは、主要な資産の特定、アプリケーションの分類、各資産またはコンポーネントへの脅威の特定と分類、脅威をリスクレベルで格付け、そして脅威を緩和するための戦略を、設計・コード・テストケースで実装する。



Maintenance





各開発フェーズ間のトレーサ
ビリティを取ることで、問題
を特定し、迅速かつコスト効
率良く対処できるようになる



System Requirements

Software High-Level Requirements

Software Low-Level Requirements

Source Code

Relationships

(0) Three-Level Requirements to Mappings

(13) Requirements 1

(34) Requirements 2

(58) Requirements 3

(47) Mappings

SYS_0010, Display , (2 Notes)

SYS_0020, Initialisation and configuration

SYS_0030, Output Calculation , (1 Note)

SYS_0040, Photometer, (1 Note)

SYS_0050, Cleanliness factor

SYS_0060, Lighting control unit , (1 Note)

SYS_0070, Luminaries

SYS_0080, Sirens and Signs, (1 Note)

SYS_0090, Failed Power Supply

SYS_0100, Lighting Adjustment

SYS_0110, Lamp output units

SYS_0120, Lamp sizes , (1 Note)

SYS_0130, Required luminance at ground level

Requirement Body

The Tunnel Lighting system shall be configurable via an external file and take into account tunnel dimensions, zones, spacing for signs ,and efficiency factor

HLR_0010, Starting display software, (1 Note)

HLR_0020, Input option photometer nominal ran...

HLR_0030, Input options photometer input out ...

HLR_0040, Input options exit

HLR_0050, Input options door closing no...

LLR_0010, Instantiate Cell

LLR_0020, Initialise Cell , (1 Note)

LLR_0030, Set Emergency output level

LLR_0040, Set PoweredOutputLevel , (1 Note)

LLR_0050, Cal

LLR_0060, Get

LLR_0070, Get

LLR_0080, Get

LLR_0090, Get

LLR_0100, Get

LLR_0110, Get

LLR_0120, Init

LLR_0130, Set

LLR_0140, Get

Bool TunnelData::Cell::InitialiseCell(const Sint_32 Lu...

Bool TunnelData::DataIn::GetData(TunnelData::Tun...

Float_64 TunnelData::Cell::CalculateCellOutput(Floa...

Float_64 TunnelData::Lamp::GetMaximumLumens();

Float_64 TunnelData::Lamp::GetMinimumLumens();

Float_64 TunnelData::LampType::GetMaximumLum...

Float_64 TunnelData::LampType::GetMinimumLum...

Float_64 TunnelData::SystemData::GetEmergencyL...

Float_64 TunnelData::SystemData::GetLampMaxim...

Float_64 TunnelData::SystemData::GetLampMinim...

Float_64 TunnelData::SystemData::GetSoilingFacto...

Sint_32 TunnelData::LampType::GetPowerRequired(...

Sint_32 TunnelData::SystemData::GetDaysBetween...

Sint_32 TunnelData::SystemData::GetExitSignSpaci...

Sint_32 TunnelData::SystemData::GetLampPowerR...

Sint_32 TunnelData::SystemData::GetSirenSpacing();

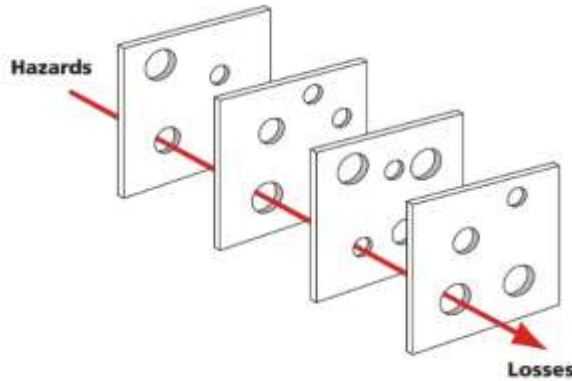
Sint_32 main();

TunnelData::Cell::Cell();

要件～コード、テストまで変更のインパクトを成果物間で追跡

まとめ





セキュリティ
対策に単一解
は無い

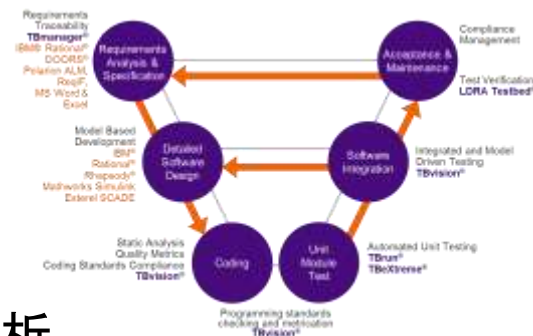
セキュアな
コードはス
イステーズ
モデルの防
御に不可欠

防御技術にか
かわらず、ア
プリケーショ
ンはシステム
の重要な役割
を果たす

つながる世界
の安心で安全
なソフトウェア
には、多重
防御が必要

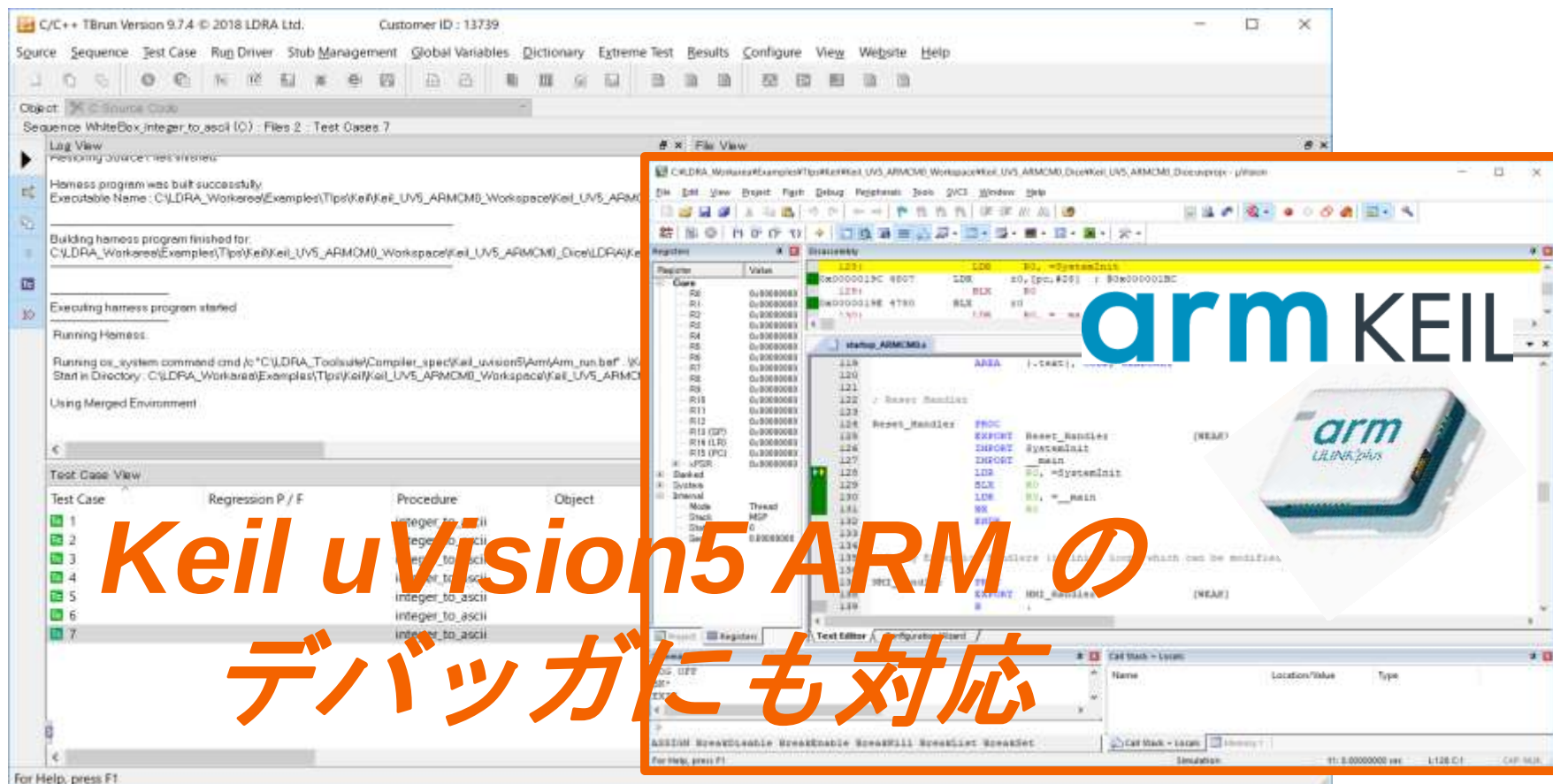


- 静的解析：コーディングスタンダード準拠、コード品質のメトリクス
 - MISRA、CERT C、IPA/SEC ESCR 等各種サポート。ユーザ定義ルール集も編集できる
 - コードの視覚化（コールグラフ、フローグラフ、クロスリファレンス）
- 動的解析：構造化カバレッジ
 - 関数コール、ステートメント、ブランチ、MC/DC、LCSAJ (path coverage).
 - アセンブラレベルカバレッジ (DO-178C Level A)
 - あらゆる組込みコンパイラ、実行環境をサポート
- 単体テスト（Unit/low-level testing）
 - テストハーネス、スタブ、グローバル宣言などの自動生成
 - 堅牢性テストのためのテストケース自動生成
- データフロー・コントロールフローの静的・動的解析
 - DO-178C で認定可能な唯一の汎用ツール
- 要件トレーサビリティに検証実行・オブジェクト管理を統合
- ツール認定キットを提供（DO-178Cに必要）
- TUVの事前認定取得済み



あらゆるターゲット環境に対応

LDRA



Keil uVision5 ARM 環境で静的解析～動的テスト実行とカバレッジ解析
<https://www.fuji-setsu.co.jp/demo/LDRAKeiluVision5ARM.wmv>

Keil uVision5 ARM 環境で要件トレーサビリティとテスト実行
https://www.fuji-setsu.co.jp/demo/TBmanager_KeiluVision5_ARM.wmv

スタンダード認証取得を目的に広く採用

LDRA



航空システム



防衛システム



医療機器



産業機器
電カプラント



鉄道システム



車載システム



**Professor Mike
Hennell**

Member of SC-205 /
WG-71 (DO-178C) formal
methods subgroup

Member of MISRA C
committee and MISRA
C++ committee

Member of the working
group drafting a proposed
secureC annex for the C
language definition
(SC 22 / WG14)



Bill St Clair

Member of SC-205 /
WG-71 (DO-178C) Object
Oriented
Technology subgroup



Andrew Banks

MISRA Committee Chair
Committee Member for
Second Edition of ISO 26262

Chair of BSI IST/15/-/26
for Software Testing

MISRA representative to BSI
IST/15 for Software and
Systems Engineering

Member of BSI IST/15-/20 for
Bodies of Knowledge and
Professionalization



Dr Clive Pygott

Member of ISO software
vulnerabilities working
group (SC 22 / WG 23)

Member of MISRA
C++ committee

Member of the working
group drafting a proposed
secureC annex for the C
language definition
(SC 22 / WG14)



Liz Whiting

Member of MISRA C
committee language
definition

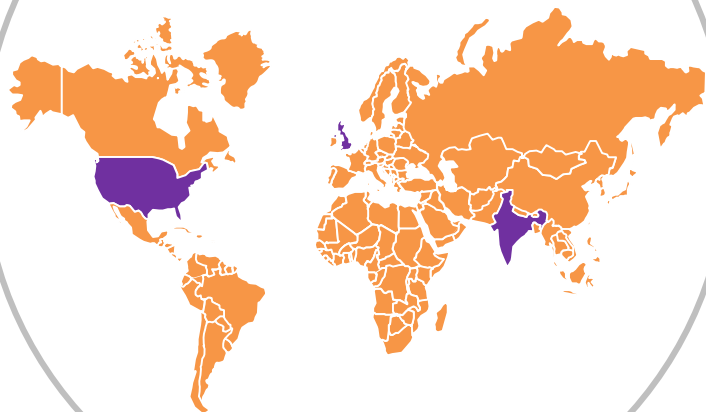


Chris Tapp

Chairman of MISRA
C++ committee

Member of MISRA C
committee language
definition

LDR



ISO 9001認証取得

1975年設立 => 40年以上の実績とブランド

IEC 61508, IEC 62304, EN 50128,

ISO 26262, IEC 60880 のツール認定取得

スタンダード認証プロセスを支援する

ソフトウェアテスト、管理支援ツールを提供

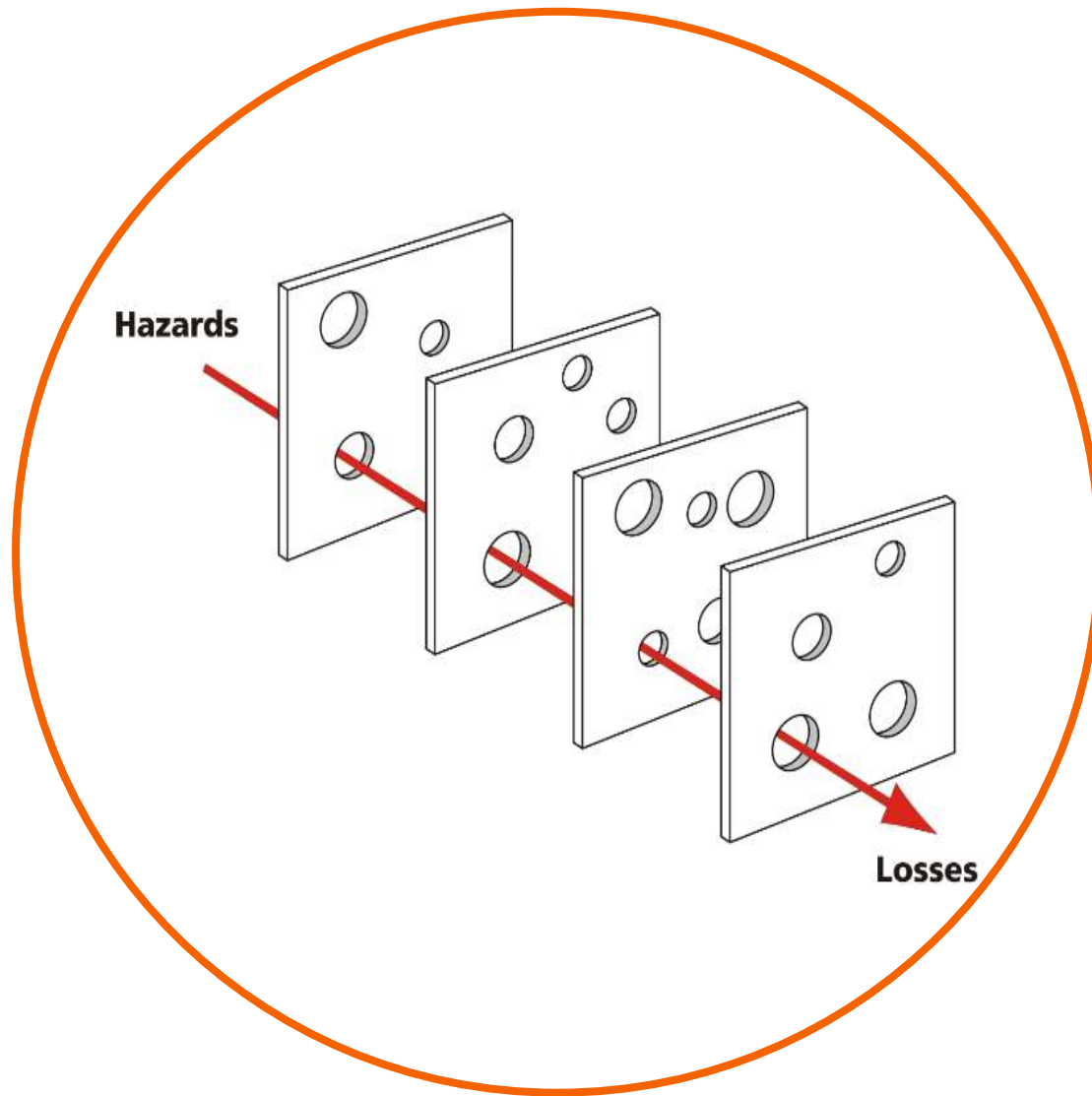
各種スタンダード委員会に貢献

e.g. DO-178C,

MISRA C/C++, CERT



コンパイラのテスト



C/C++コンパイラをテストして 妥当性を検証する

～ ユーザーまでもがコンパイラを
テストすべき理由とは？ ～



コンパイラテストの重要性



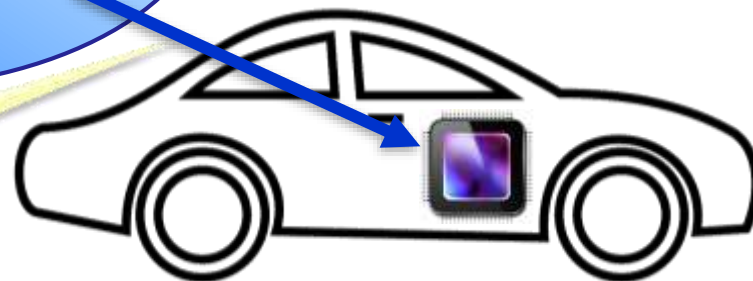
コンパイラは巨大で非常に複雑なソフトウェア

- 品質の検証には膨大な条件の組合わせによるテストが必要
- コンパイラメーカーは全オプションの組合せをテストできない

ブレーキに
対する
Cソースコード

C/C++コンパイラ
500万行規模

間違ったコードが生成されることによる被害が甚大



- C, C++言語標準規格への適合性、正確性、堅牢性をチェックする300万件以上のテスト
- 30年以上の経験の積み重ねで開発
- 主にポジティブとネガティブテスト

■ ■ ■

コード生成機能进行评估するテスト

診断機能进行评估するテスト

コンパイラユーザも採用



顧客事例 <https://www.fuji-setsu.co.jp/products/SuperTest/#customers>

コンパイラユーザ成功例



- 同じコンパイラを1000本以上使用している
 - バージョンアップごとにSuperTestでチェックして、使ってはいけない最適化などの社内ルールを施行することで、過去のバージョンに戻さなければならないような問題が起こっていない。数千人に関わることなので非常に高い成果である。
 - ABI-Tester 機能でABIの規則に準拠
 - 他のサプライヤーの製品と繋がることで問題が発生しても、お相手の問題として切り分けすることができる。
- 各社がSuperTestを使ってくれれば、もっと助かるのだが、

コンパイラユーザ成功例



- ライブラリのテスト
 - C99 なのにC89のライブラリが混同されていて、sin, cosなどの数学関数で問題になるところを、事前に回避できた。
 - 自動運転などで画像認識等はC++のライブラリに依存することになるので、SuperTestのライブラリテスト機能にも注目している。またロードマップ内のFloat16のテストや、AUTOSAR C++14 対応にも期待している。
- Depth Suite 機能
 - アーキテクチャ固有のデータ長（例えば char も int も24ビット）に合わせて演算精度のテストができる。

