コンパイラ認定と言語サブセットの相互補完による安全・安心なソフトウェア

LDRA

SuperTest

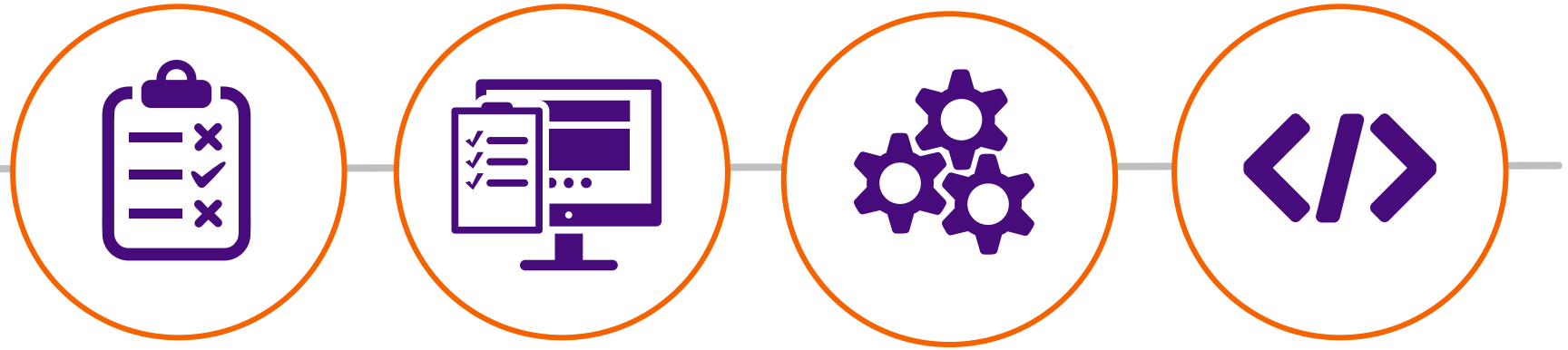
コーディング規約

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、問題のあるサブセットを避けることで、安全で安心なソフトウエアを実装することができる

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

未定義の動作：言語規格で定義されない動作

- 未定義の動作について、コンパイラは「どのように」実装してもよい
- 以下の関数は値を返さない

コンパイラは警告を出すがコードは生成されて実行される！

この関数を呼び出した側で戻り値を参照した場合、どうなるかはわからない

```
int32_t undef(void)
{

int32_t noret(void)
{
    global += undef();
    // no return exp.
}
```

```
$ gcc -Wall unspec.c
unspec.c: In function ‘noret’:
unspec.c:21:1: warning: control reaches end of non-void function [-Wreturn-type]
   21 | }
      | ^
```

未規定の動作：規定されない動作

- 未規定の動作とは、規定されていない値を使用したり、規格で複数の可能性があって、そのどれを選択するかの要求がない場合のこと
- 例：関数の実引数を評価する順序

  引数の noret()関数は、引数globalが評価される前に呼び出されることがある

  ⇒ 同じプログラムでも処理系によって結果が異なる可能性があり、

  他のプラットフォームへ移植する際に不具合を生じる可能性がある

```
int main(void)
{
    printf("%d %d\n", global, noret());
}
```

処理系定義の動作：未規定の動作を処理系が独自に実装

- ある機能を実装する際に複数の選択肢があり、コンパイラベンダーが1つを選ばなければならない場合、処理系定義の動作となる
  処理系定義の動作は定義されなければならない
  （処理系が違えば、同じコードでも同じ動作とは限らない！）

- 符号付き整数を右シフトした場合の最上位ビットの伝播

```
signed int value = 0x80000000;
value = value >> 1;
```

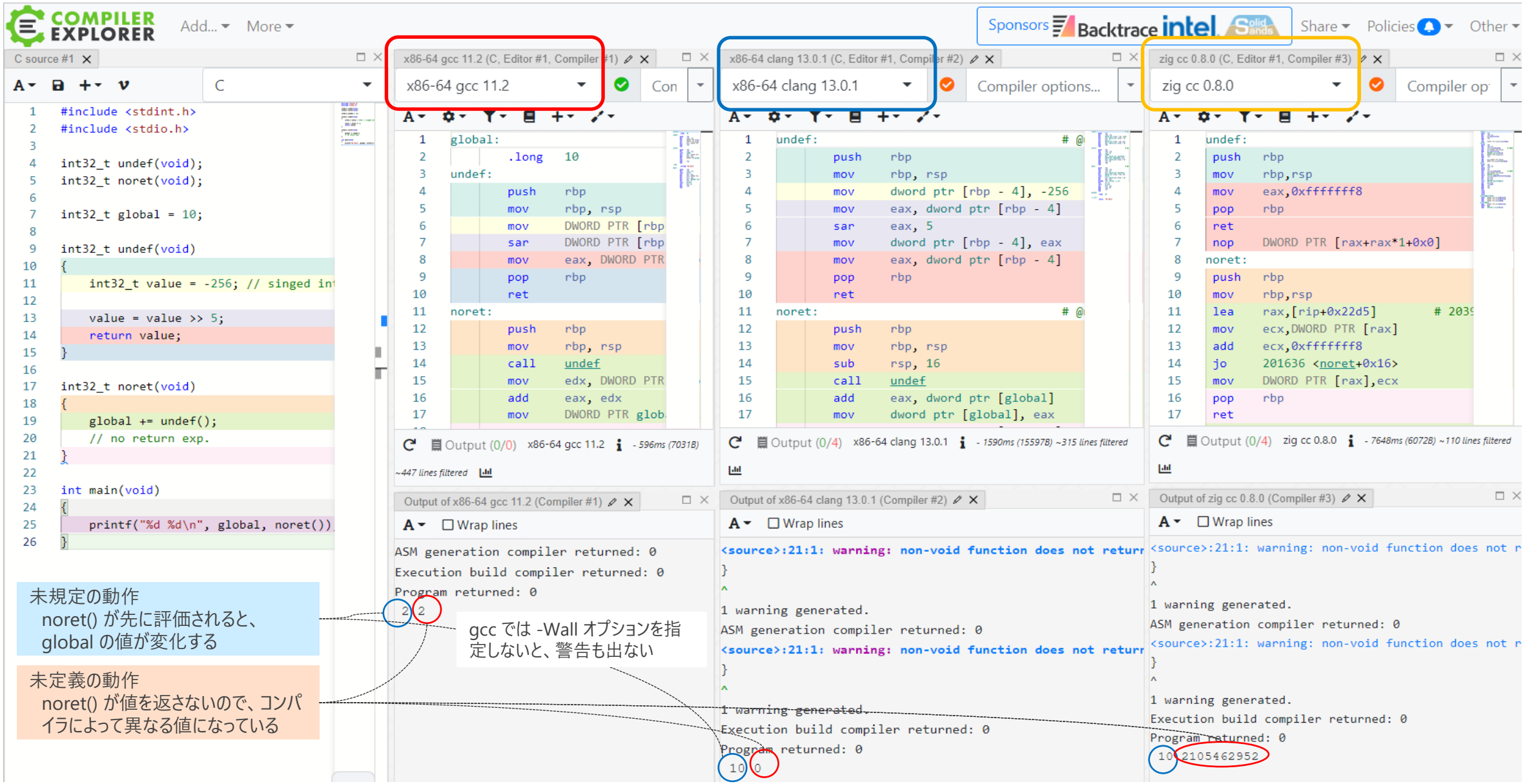
C言語規格では、「未規定の動作のうち、各処理系が選択した動作を文書化するもの」として定められており、未規定の動作の中で、処理系によってどの動作が選択されたか文書化されたものが「処理系定義の動作」

論理シフトとして実装の場合：最上位ビットが右に移動され、新たな最上位には 0 が入って 0x40000000

算術シフトとして実装の場合：最上位ビットは符号であり、符号ビットの値（例では1）が引き続いて入って 0xC0000000

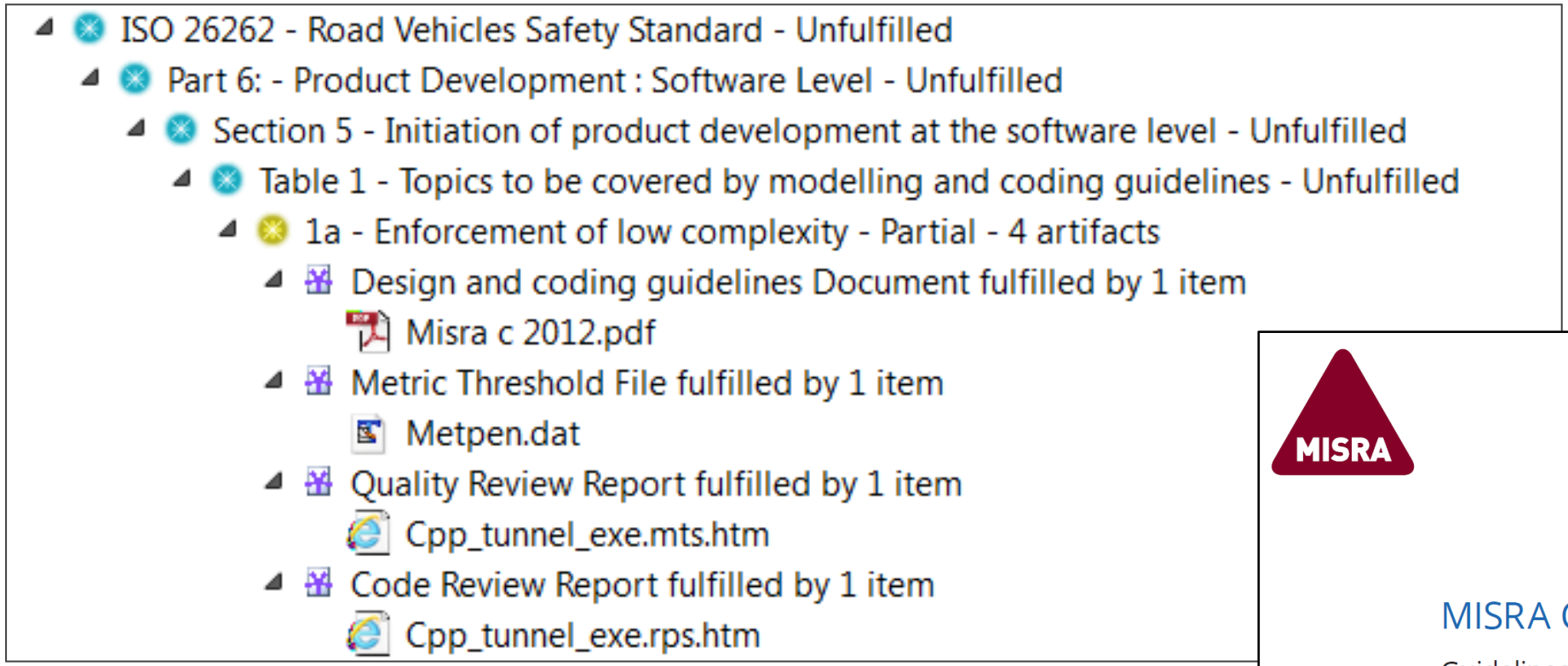
- 処理系定義の動作は未定義の動作とは違い正常なプログラム動作。処理系が同じなら動作に再現性がある。しかし処理系が異なった場合に動作が異なる可能性があり、プログラムの移植時に注意が必要！

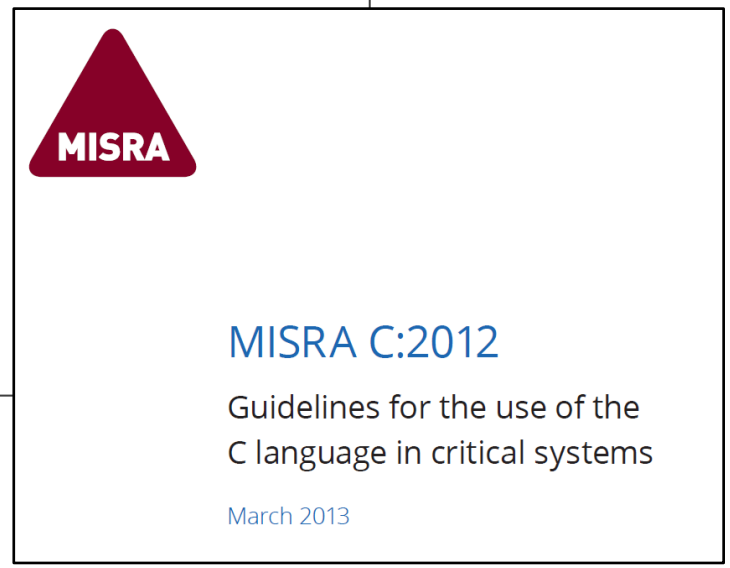
未定義、未規定動作の例


COMPILER EXPLORER
Add... More
Sponsors Backtrace intel Share Policies Other
C source #1
C
x86-64 gcc 11.2 (C, Editor #1, Compiler #1)
x86-64 gcc 11.2
x86-64 clang 13.0.1 (C, Editor #1, Compiler #2)
x86-64 clang 13.0.1
Compiler options...
zig cc 0.8.0 (C, Editor #1, Compiler #3)
zig cc 0.8.0
Compiler op
1 #include <stdint.h>
2 #include <stdio.h>
3
4 int32_t undef(void);
5 int32_t noret(void);
6
7 int32_t global = 10;
8
9 int32_t undef(void)
10 {
11 int32_t value = -256; // singed int
12
13 value = value >> 5;
14 return value;
15 }
16
17 int32_t noret(void)
18 {
19 global += undef();
20 // no return exp.
21 }
22
23 int main(void)
24 {
25 printf("%d %d\n", global, noret());
26 }
Output (0/0) x86-64 gcc 11.2 - 596ms (7031B)
~447 lines filtered
Output (0/4) x86-64 clang 13.0.1 - 1590ms (15597B) ~315 lines filtered
Output (0/4) zig cc 0.8.0 - 7648ms (60728B) ~110 lines filtered
Output of x86-64 gcc 11.2 (Compiler #1)
Output of x86-64 clang 13.0.1 (Compiler #2)
Output of zig cc 0.8.0 (Compiler #3)
Wrap lines
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 0
2 2
1 warning generated.
10 0
10 2105462952
未規定の動作
noret() が先に評価されると、
global の値が変化する
未定義の動作
noret() が値を返さないので、コンパイラによって異なる値になっている
gcc では -Wall オプションを指定しないと、警告も出ない

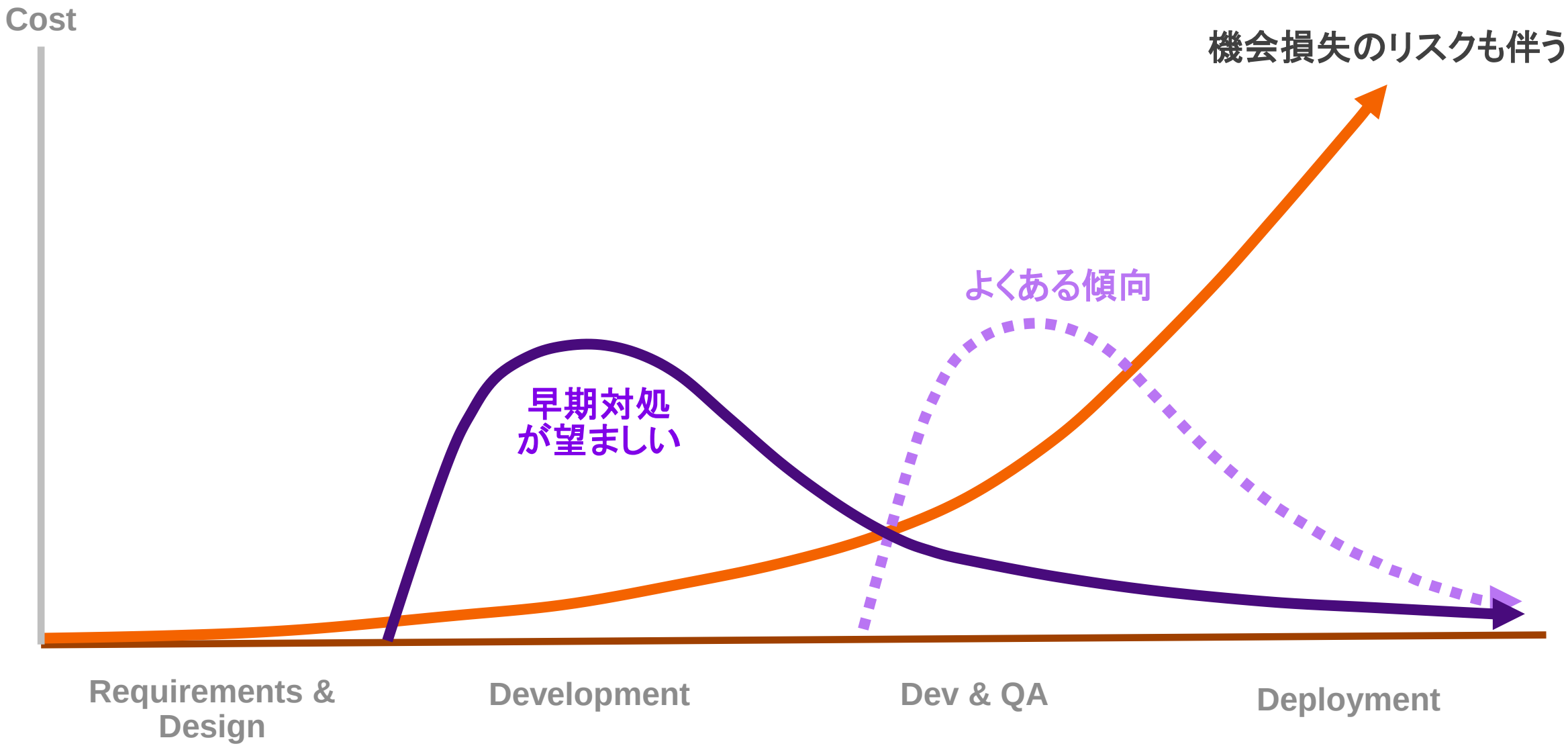
未定義、処理系定義、未規定の動作 を避ける

- アプリケーションの信頼性や移植性が損なわれるので

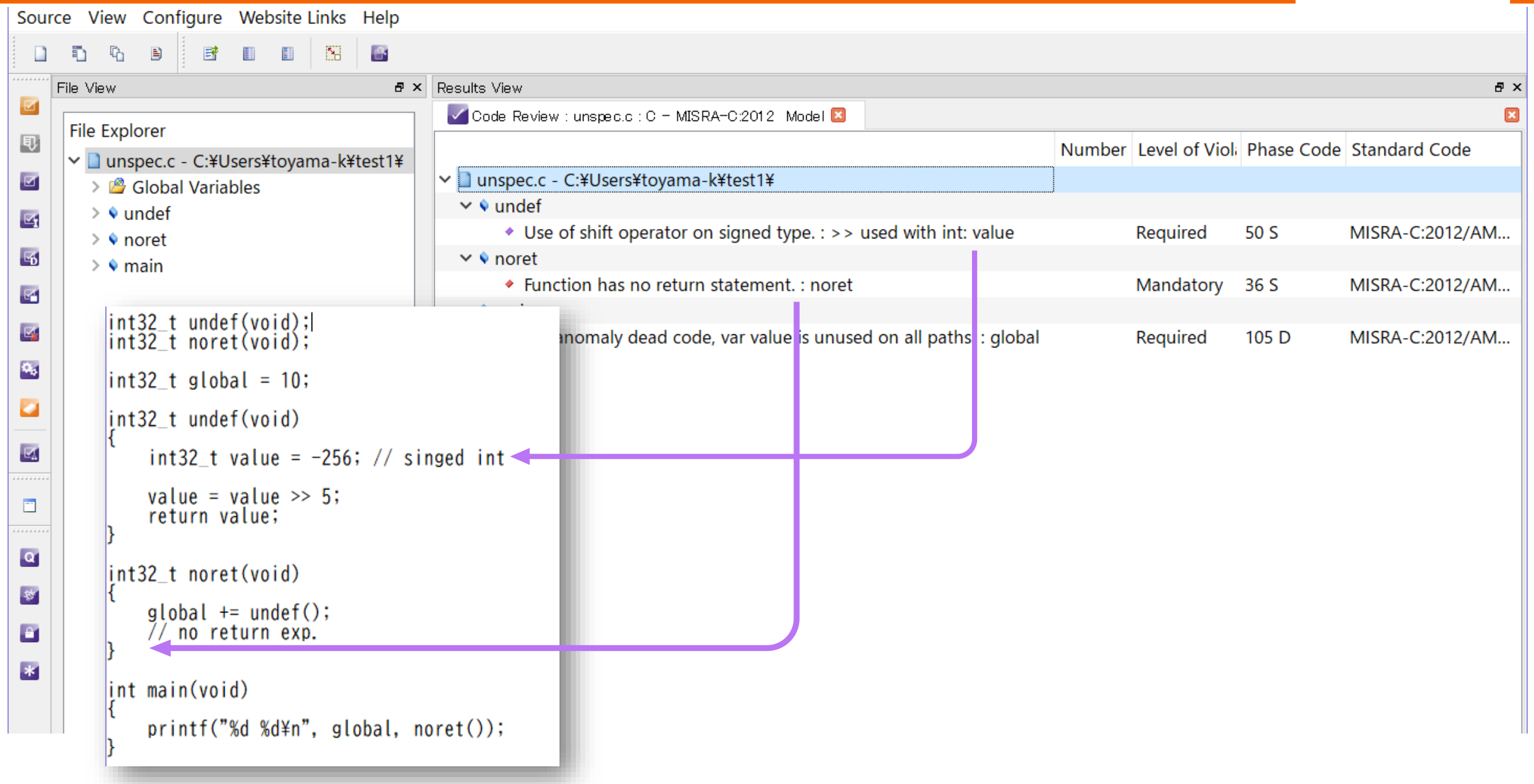




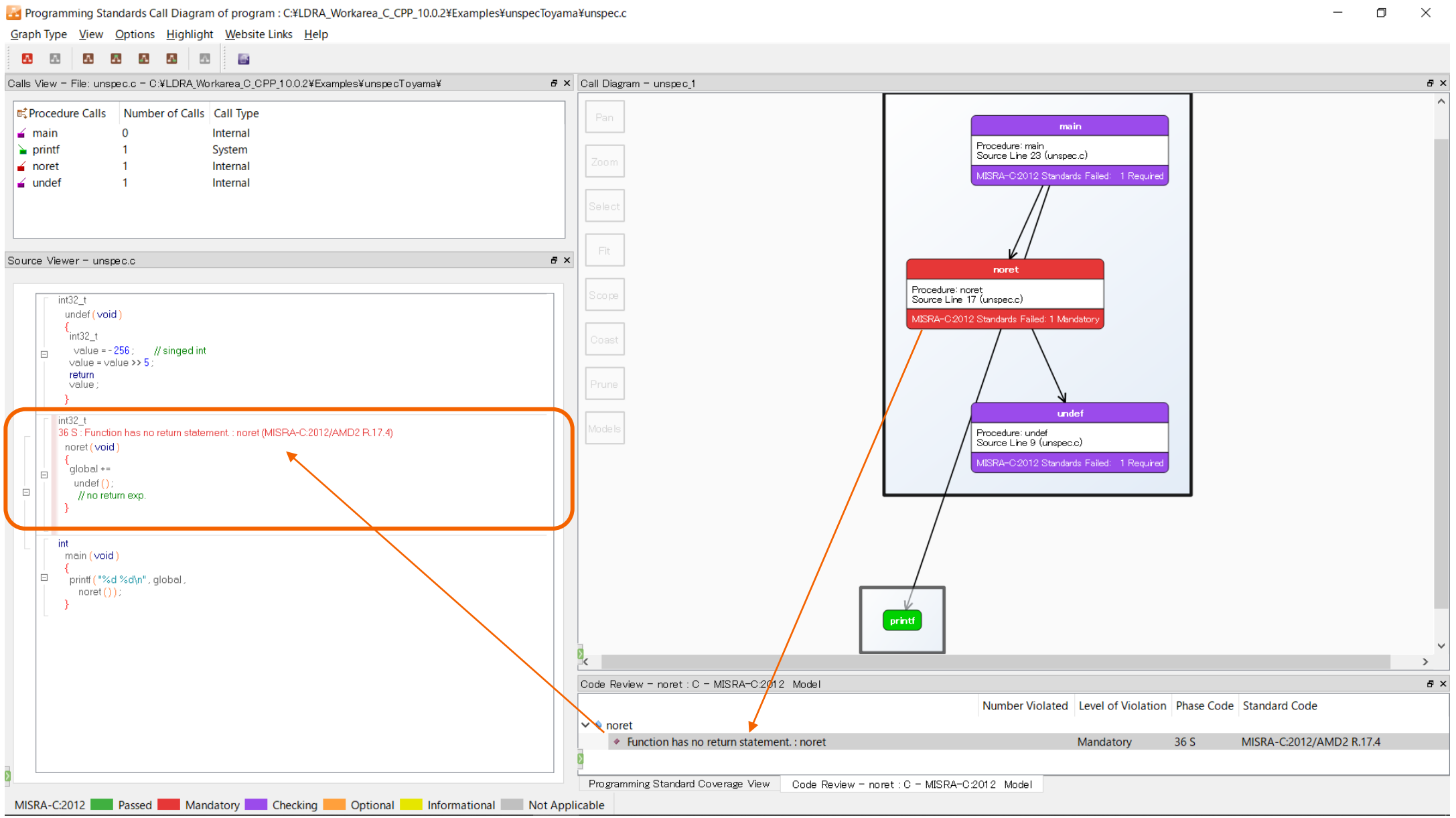
バグの改修コストは後工程になるほど高くつく



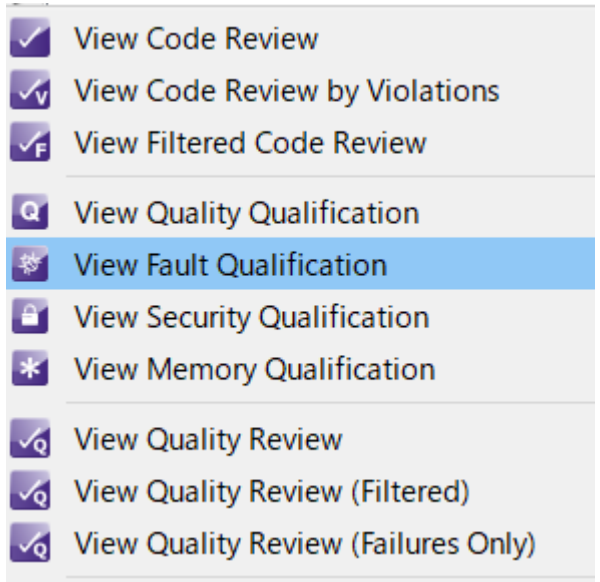
コーディング規約チェック

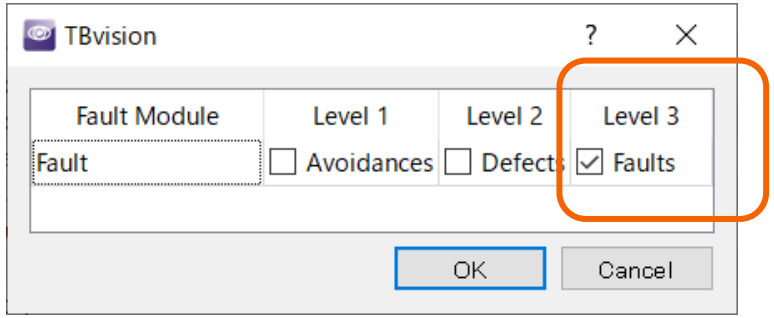


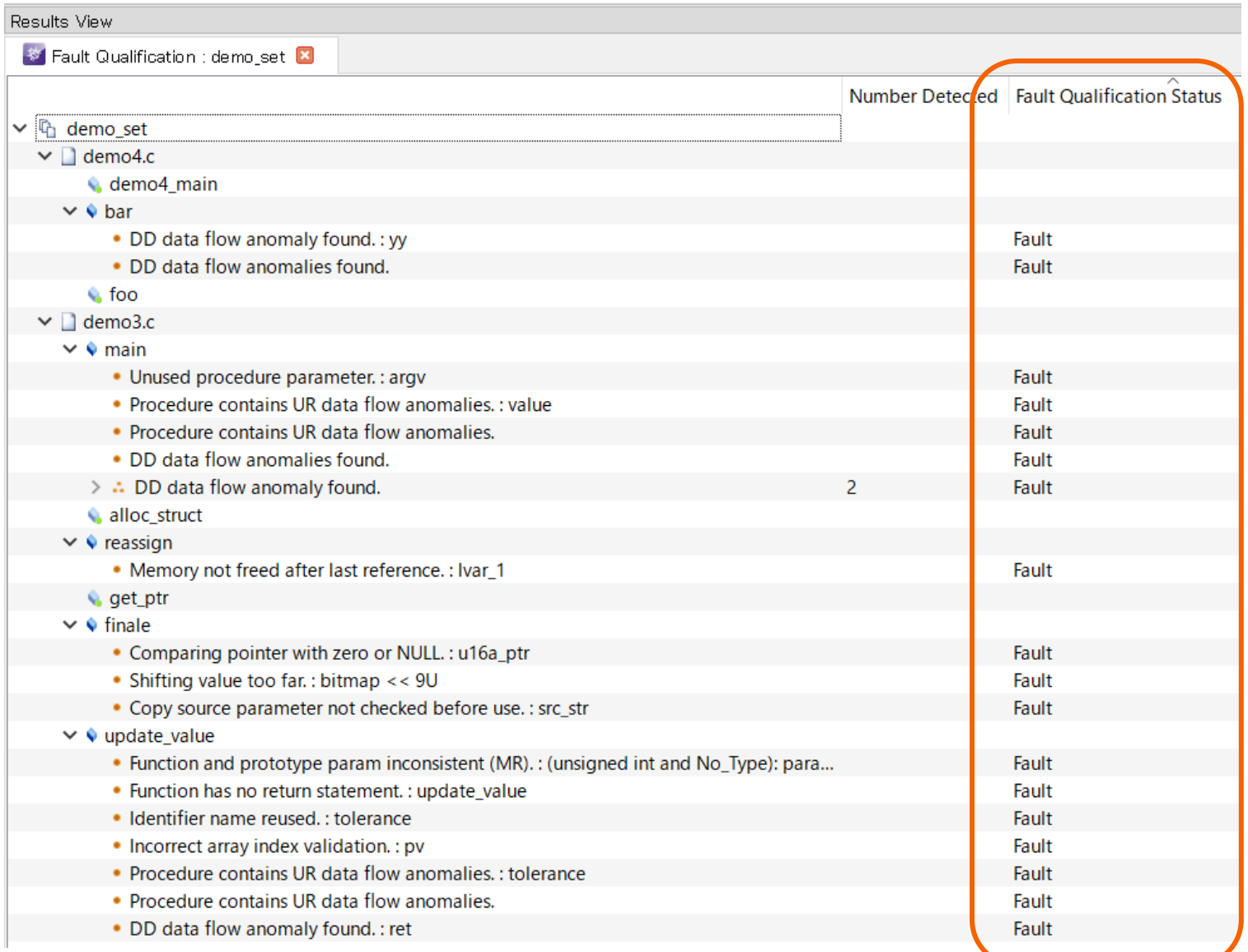
関数コールグラフに視覚化



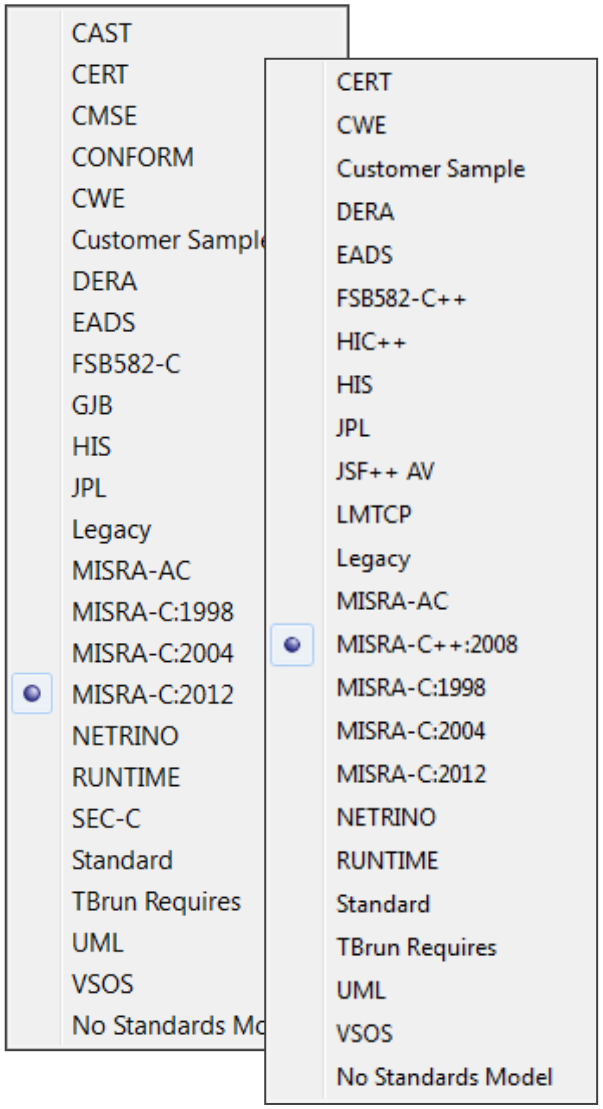
リスクベースのアプローチ

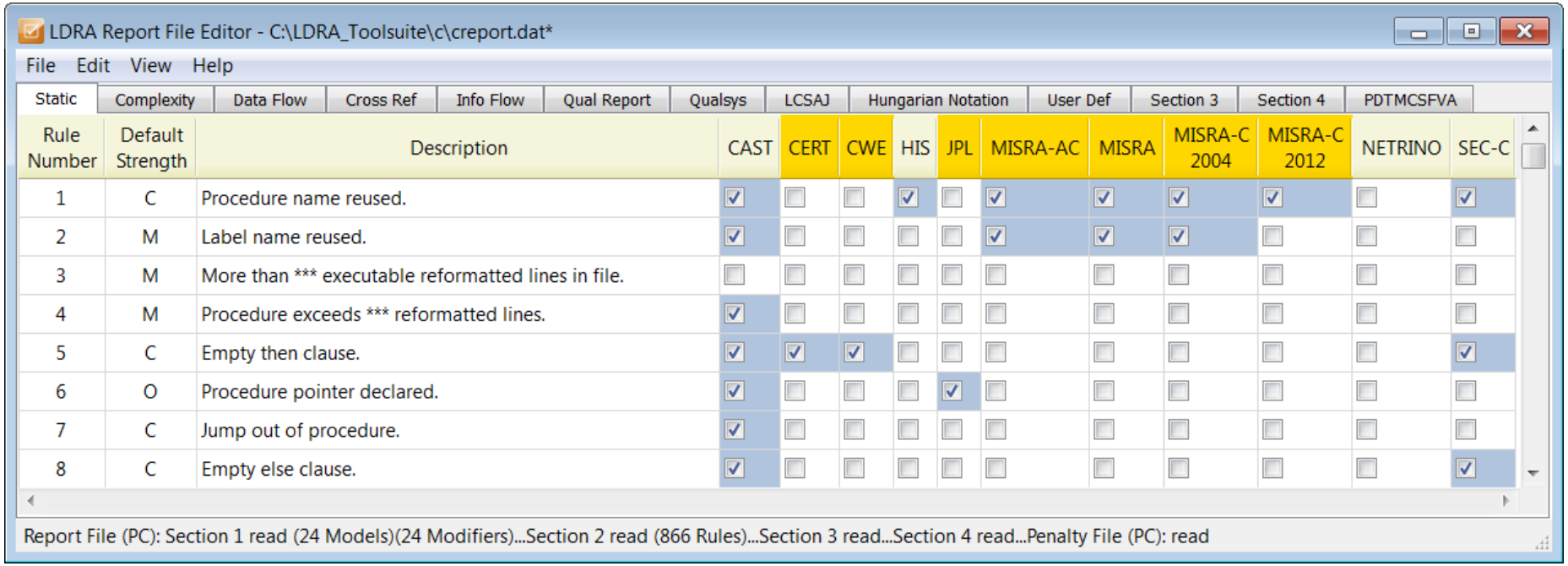






CERTなど各種コーディング規約をサポート

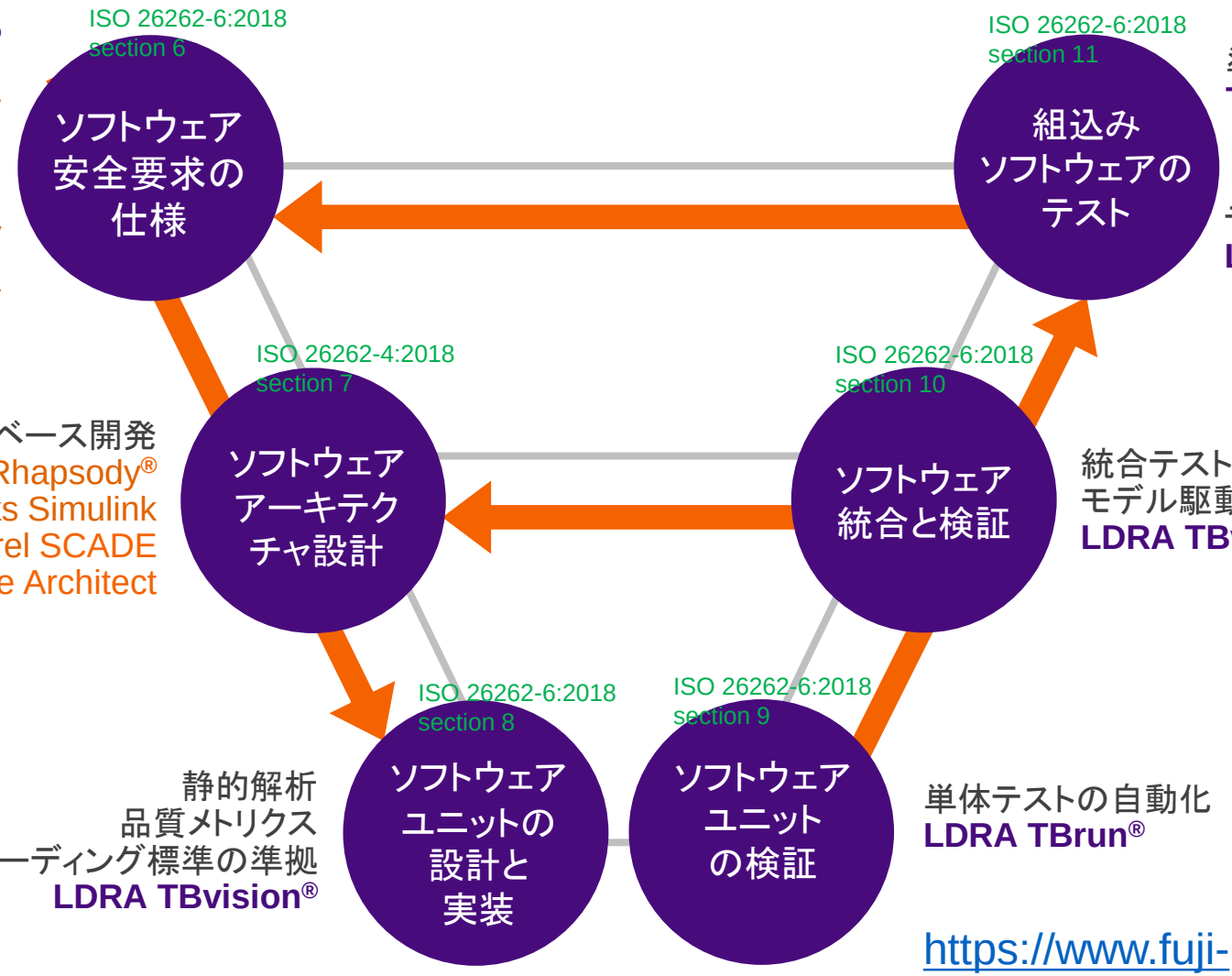




Rule Number	Default Strength	Description	CAST	CERT	CWE	HIS	JPL	MISRA-AC	MISRA	MISRA-C 2004	MISRA-C 2012	NETRINO	SEC-C
1	C	Procedure name reused.	☑	☐	☐	☑	☐	☑	☑	☑	☑	☐	☑
2	M	Label name reused.	☑	☐	☐	☐	☐	☑	☑	☑	☐	☐	☐
3	M	More than *** executable reformatted lines in file.	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
4	M	Procedure exceeds *** reformatted lines.	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
5	C	Empty then clause.	☑	☑	☑	☐	☐	☐	☐	☐	☐	☐	☑
6	O	Procedure pointer declared.	☑	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐
7	C	Jump out of procedure.	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
8	C	Empty else clause.	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐	☑

https://www.fuji-setsu.co.jp/files/Implementing_ISO_26262_second_edition_white_paper_v2.0JP.pdf#page=15

V Model – ソフトウェア開発プロセスと自動化ツールチェーン



要件トレーサビリティ
TBmanager®
IBM® Rational®DOORS®/NG
Intland Software codeBeamer
Jama Software
Jira Software
Polarion ALM
PTC Integrity
SilkRoad ALM
Systemite SystemWeaver
ReqIF, MS Word & Excel

準拠管理
TBmanager®

テスト検証/カバレッジ解析
LDRA TBvision®

モデルベース開発
IBM® Rational® Rhapsody®
MathWorks Simulink
Esterel SCADE
Enterprise Architect

統合テストと
モデル駆動テスト
LDRA TBvision® & TBrun®

静的解析
品質メトリクス
コーディング標準の準拠
LDRA TBvision®

単体テストの自動化
LDRA TBrun®

プログラミング標準のチェック
と品質メトリクス
LDRA TBvision®

https://www.fuji-setsu.co.jp/files/Implementing_ISO_26262_second_edition_white_paper_v2.0JP.pdf#page=9

コンパイラの品質と機能安全

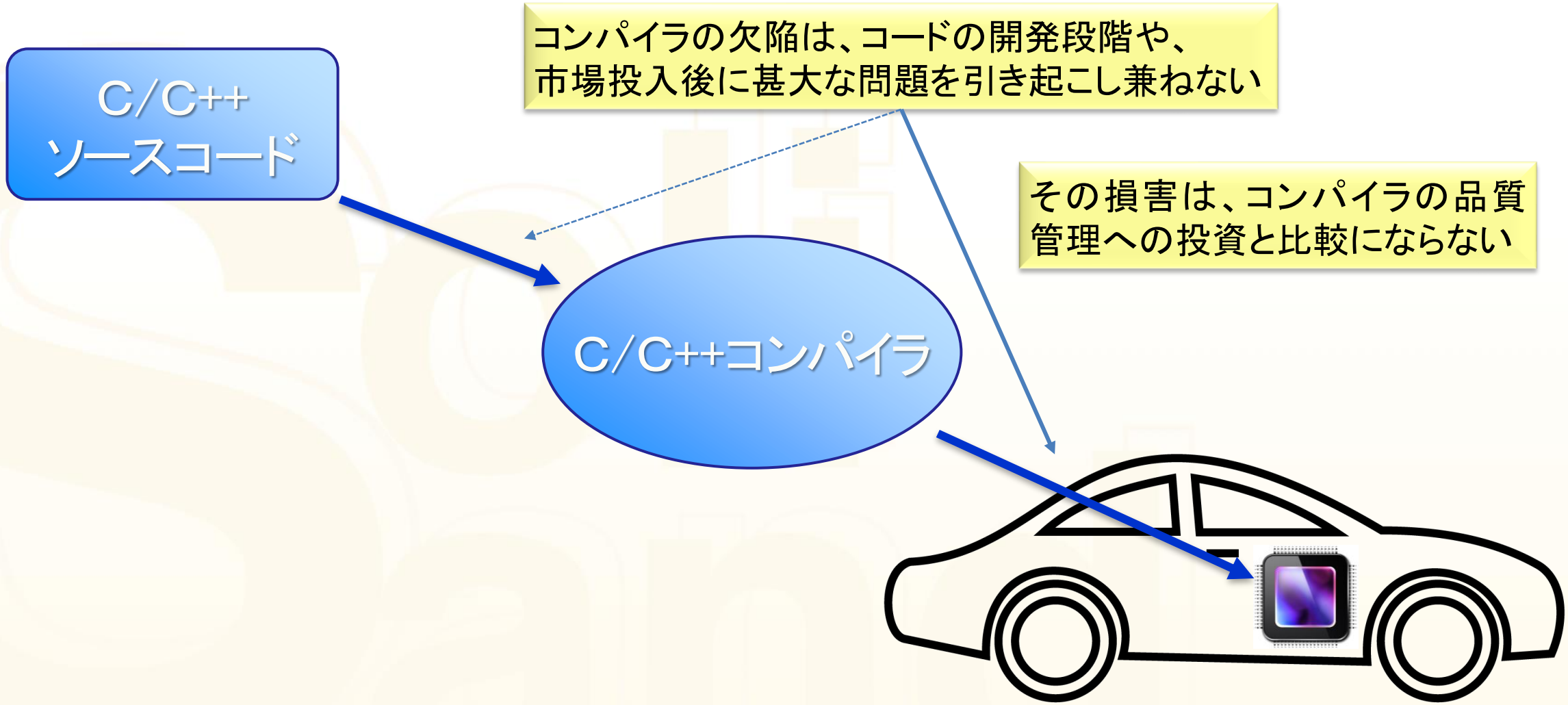
～ ユーザーがコンパイラをテストすべき理由 ～

FUJI SETSUBI

www.fuji-setsu.co.jp

Solid Sands

コンパイラの品質:バグのないコンパイラはない



コンパイラ: テストも認証も難しい

- 巨大で非常に複雑　GCCを例に取ると、、
  - C, C++コンパイラのソースコードサイズ：500万行
  - 開発期間：40年近く、何千人ものエンジニアが関与
  - MISRA等への準拠やテストのカバレッジは実施されない
- MPUのバリアントやオプション
  - 膨大な組合せ全てをメーカーはテストできない
- 開発はいつも進行中で修正、改善、機能追加など何らかの変更がある
  - 複雑なコンパイル過程での変更が、以降の処理の流れを全く違ったものに
  - バージョン3.4が信頼できるなら3.5も信頼できるということもない

コンパイラを認証することは現実的ではない

コンパイラは言語規格で評価できる

C++14 に対する G++7.3.0 のテスト結果サマリー

診断機能を評価

コード生成機能を評価

	LOG
Compile errors	fail/chck/pass
Language	81/ 93/10250
Library	22/ 0/2029
subtotal	103/93/12279
Run errors	fail/chck/pass
Language	4/ 0/8026
Library	0/ 0/1934
subtotal	4/ 0/9960
Total	107/ 93/12275

SuperTest™ : 言語規格に対するテストスイート

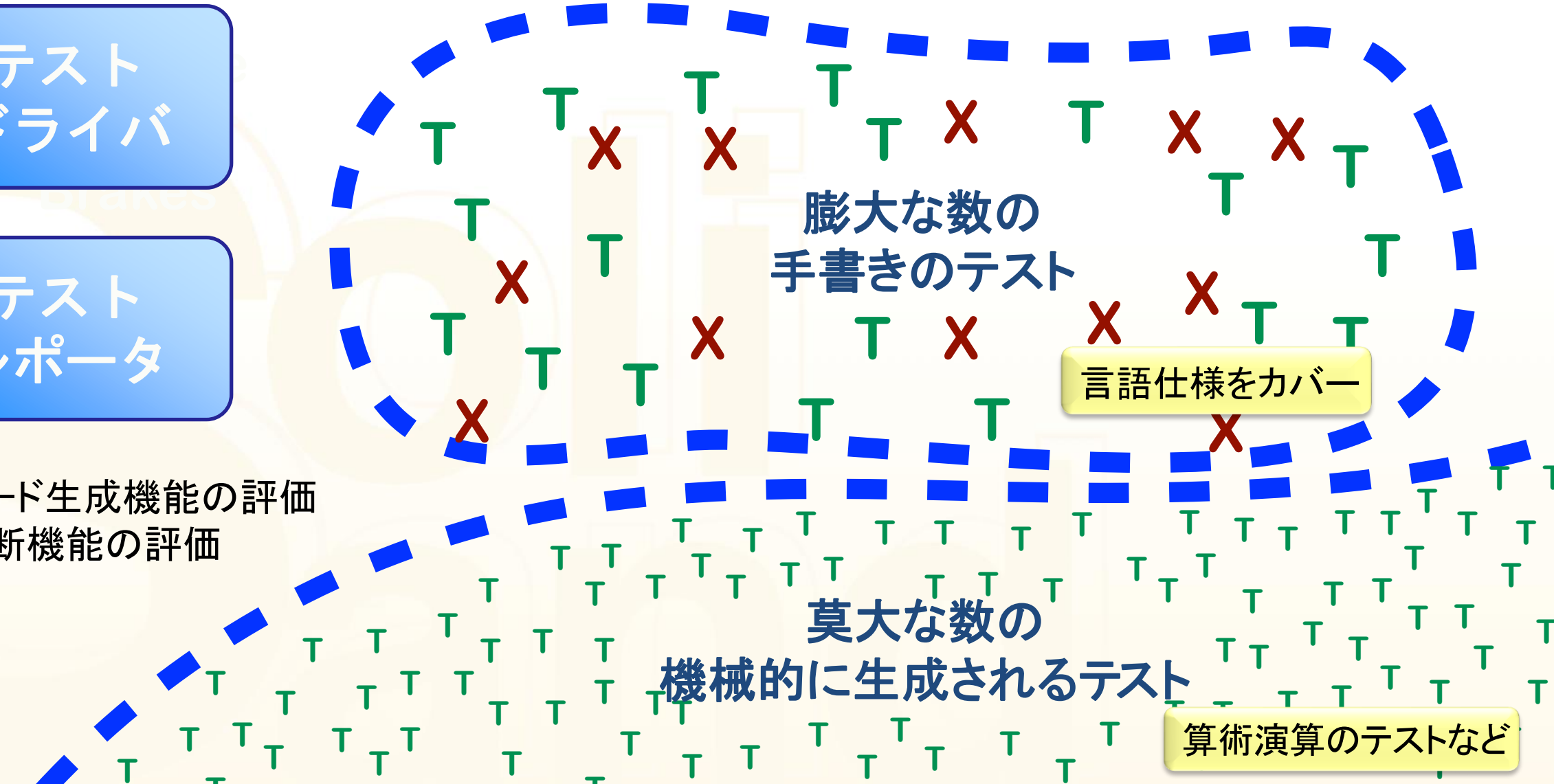
- C, C++言語規格への適合性、正確性、堅牢性をテストする
  - x 300万件以上のソースコード集
  - 30年以上ISO 言語標準規格に対応してきた経験と実績
- 開発に使用する固有のユースケースでコンパイラを評価
  - コード生成機能をオブジェクトコードの振る舞いで評価するテスト
  - 診断機能を評価するテスト
    プログラムの構文や意味規則を診断
  - C,C++ 標準ライブラリのテスト

SuperTestの概要

テスト
ドライバ

テスト
レポータ

T：コード生成機能の評価
X：診断機能の評価



SuperTest ユーザ

主要なコンパイラメーカーやデバイスメーカーがコンパイラのテストや
ツール認定を目的に活用するなか、コンパイラユーザによる採用も増えています

コンパイラユーザがSuperTestで得た成果

- 同じコンパイラを1000本以上使用
  - バージョンアップごとにチェックし、使ってはいけない最適化などの社内ルールを施行。過去バージョンに戻すような問題が起こっていない
- ライブラリのテスト
  - C99にC89のライブラリが混在し数学関数で問題になるところを事前に回避
- Depthスイートテスト機能
  - アーキテクチャ固有のデータ長（例えば int が24ビット）に合わせて演算精度のテストができる
- 呼出し規約テスト機能でABI規則への準拠を確認
  - 他のサプライヤーの製品と繋がることによる問題の切り分け

コンパイラの品質を鵜呑みにできない

機能安全とコンパイラのテスト、ツール認定

- 機能安全規格でソフトウエアツールチェインの適正なレベルの信頼が要求される
- コンパイラの中身はユーザのコントロール外
- 固有のユースケースでコンパイラをテストして認定できるSuperTest
- コンパイラのテストも開発の早期段階で行うことが賢明！

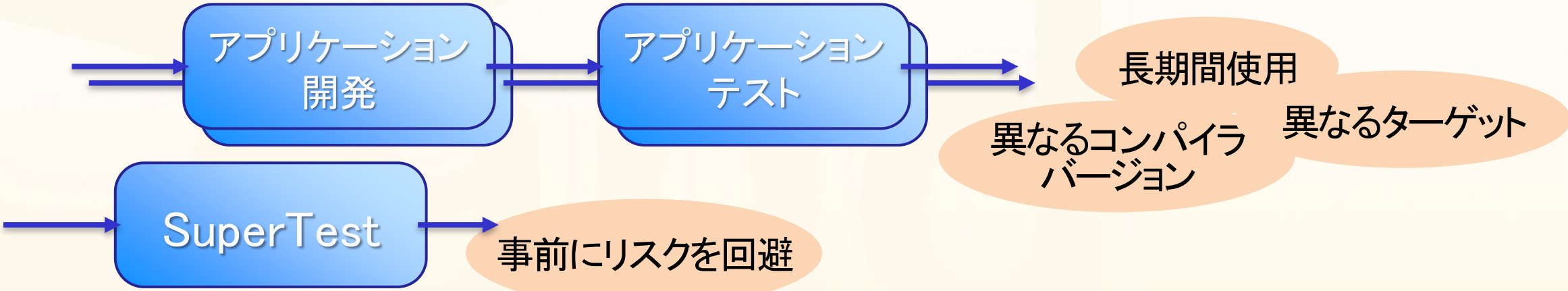
C、C++コンパイラのツール認定の効用

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

市場投入までの時間を加速

コンパイラテストなし: コンパイラの欠陥が後工程や出荷後に見つかり大きな手戻り発生

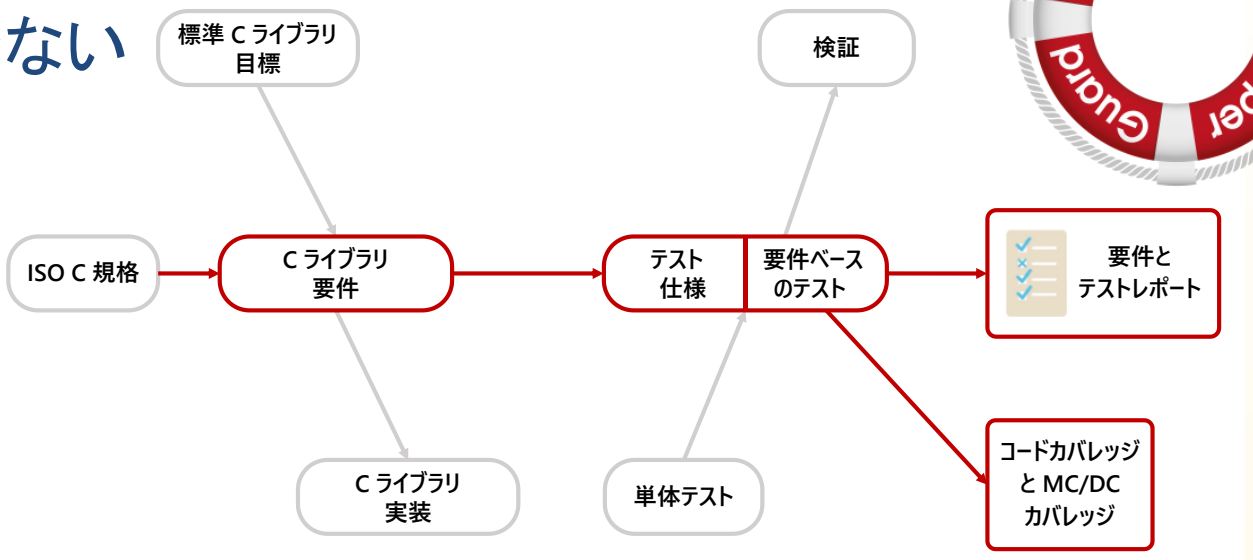
コンパイラテストあり: アプリ開発のプロセスに影響を与えるようなサプライズを回避できる



C,C++ 標準ライブラリ:コンパイラ認定とは別扱い

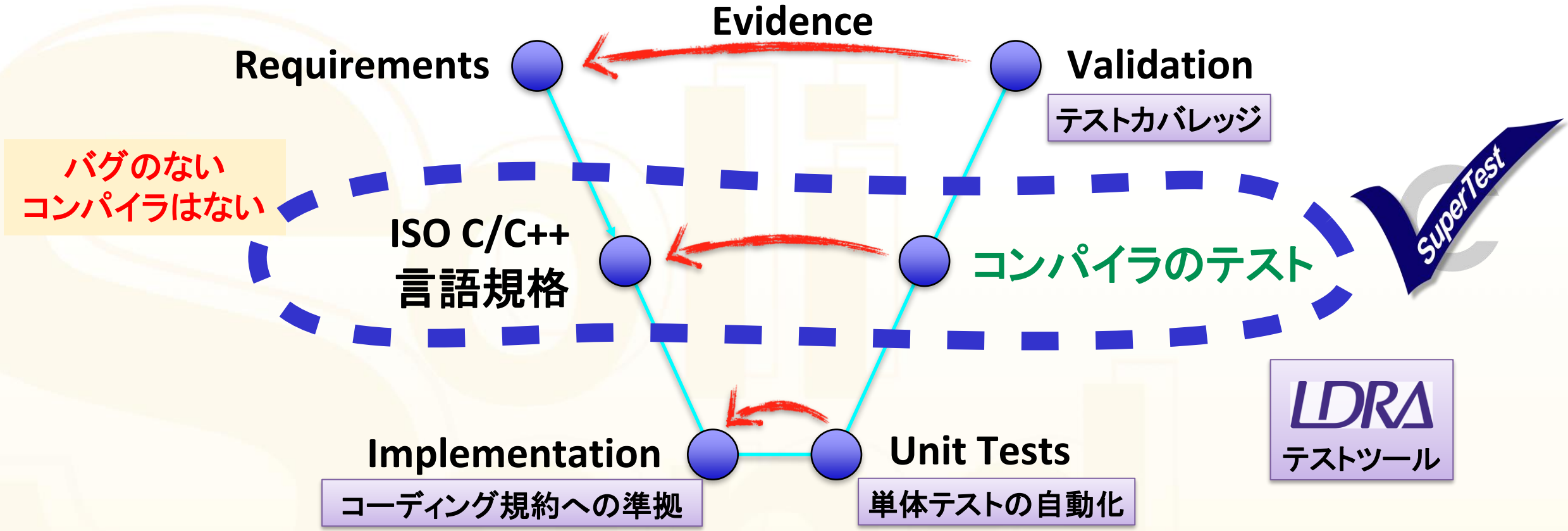
- ライブラリはアプリとリンクしてシステムの一部として実行される
- マクロやヘッダファイルなどソースコードに大きく依存するものもある
  - 言語仕様から導出される要件ベーステスト
  - 個別のコンパイラユースケース下で検証すべき
  - 事前認定されていても安心できない

・要件の分析
・同値クラスの使用
・境界値の定義
・「エラー推測」= 経験



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

検証作業とコンパイラ品質



The V-Model - Requirements Traceability

FUJI SETSUBI

LDRA　スタンダード認証支援テストツール
https://www.fuji-setsu.co.jp/products/LDRA/

SuperTest　C/C++ コンパイラテスト
https://www.fuji-setsu.co.jp/products/SuperTest/

富士設備工業(株)電子機器事業部
https://www.fuji-setsu.co.jp