

Polyspace® によるソフトウェア安全性・信頼性の確保

MathWorks Japan

アプリケーションエンジニアリング部 (制御)

アプリケーションエンジニア

田中 康博

バグを見つけることができますか？

```
ソース
検索結果の統計 WhereAreTheErrors.c
1 int new_position(int sensor_pos1, int sensor_pos2)
2 {
3     int actuator_position;
4     int x, y, tmp, magnitude;
5
6     actuator_position = 2; /* default*/
7     tmp = 0;                /* values */
8     magnitude = sensor_pos1 / 100;
9     y = magnitude + 5;
10
11     while (actuator_position < 10)
12     {
13         actuator_position++;
14         tmp += sensor_pos2 / 100;
15         y += 3;
16     }
17     if ((3*magnitude + 100) > 43)
18     {
19         magnitude++;
20         x = actuator_position;
21         actuator_position = x / (x - y);
22     }
23     return actuator_position*magnitude; /* value */
24 }
```

ゼロ割の可能性

オーバーフローの可能性

未初期化の可能性

actuator_position = x / (x - y);

Polyspace のコード証明

```
1 int new_position(int sensor_pos1, int sensor_pos2)
2 {
3     int actuator_position;
4     int x, y, tmp, magnitude;
5
6     actuator_position = 2; /* default*/
7     tmp = 0; /* values */
8     magnitude = sensor_pos1 / 100;
9     y = magnitude + 5;
10
11     while (actuator_position < 10)
12     {
13         actuator_position++;
14         tmp += sensor_pos2 / 100;
15         y += 3;
16     }
17     if ((3*magnitude + 100) > 43)
18     {
19         magnitude++;
20         x = actuator_position;
21         actuator_position = x / (x - y);
22     }
23     return actuator_position*magnitude;
24 }
```

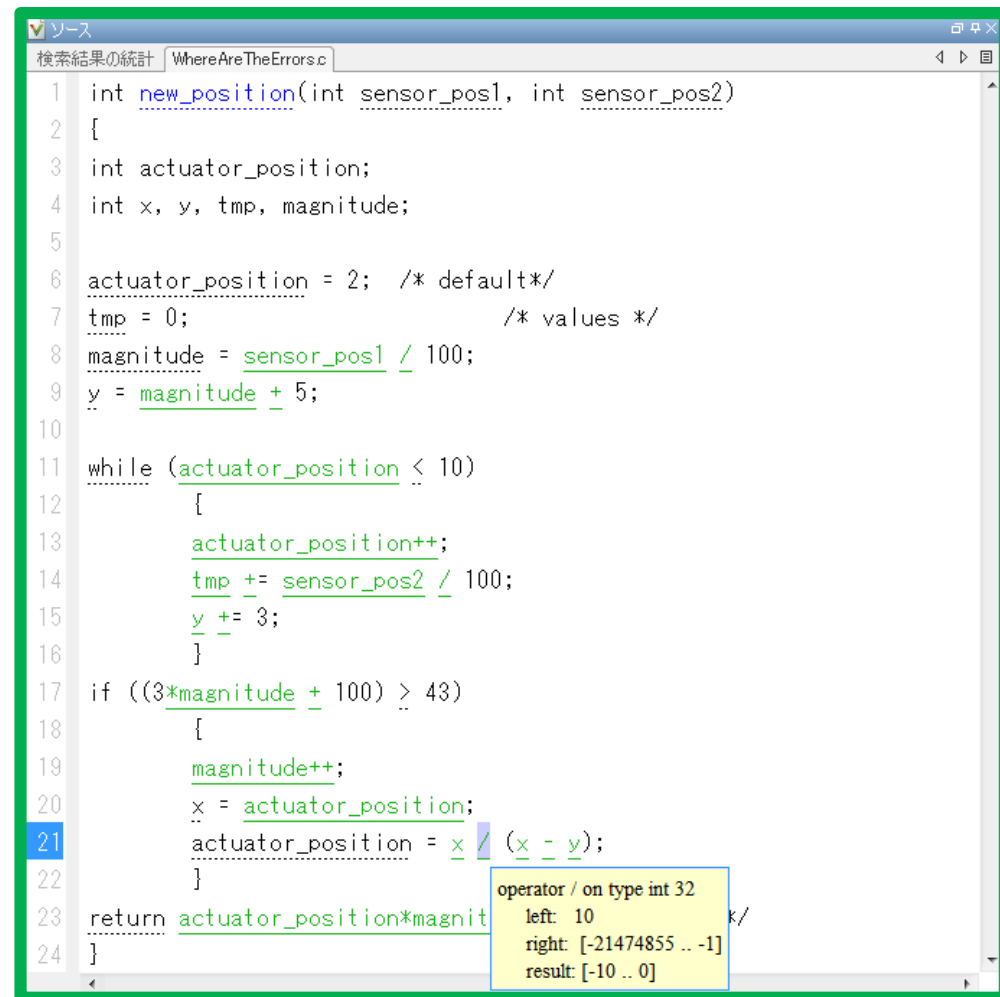
operator / on type int 32
left: 10
right: [-21474855 .. -1]
result: [-10 .. 0]

- ✓ あらゆる条件でコードの安全性を証明
- ✓ テストを行わずに網羅的に解析
 - 実行時エラー
 - 条件による実行時エラー
 - 到達不能なコード

actuator_position = x / (x - y);

アジェンダ

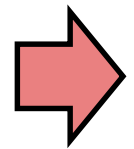
- Polyspaceについて
- ソフトウェア検証に関するチャレンジ
- Polyspace静的解析の解決案
- Polyspaceのメリット
- ユーザー事例 & まとめ



```
1 int new_position(int sensor_pos1, int sensor_pos2)
2 {
3     int actuator_position;
4     int x, y, tmp, magnitude;
5
6     actuator_position = 2; /* default*/
7     tmp = 0; /* values */
8     magnitude = sensor_pos1 / 100;
9     y = magnitude + 5;
10
11     while (actuator_position < 10)
12     {
13         actuator_position++;
14         tmp += sensor_pos2 / 100;
15         y += 3;
16     }
17     if ((3*magnitude + 100) >= 43)
18     {
19         magnitude++;
20         x = actuator_position;
21         actuator_position = x / (x - y);
22     }
23     return actuator_position*magnitude;
24 }
```

operator / on type int 32
left: 10
right: [-21474855 .. -1]
result: [-10 .. 0]

アジェンダ



Polyspaceについて

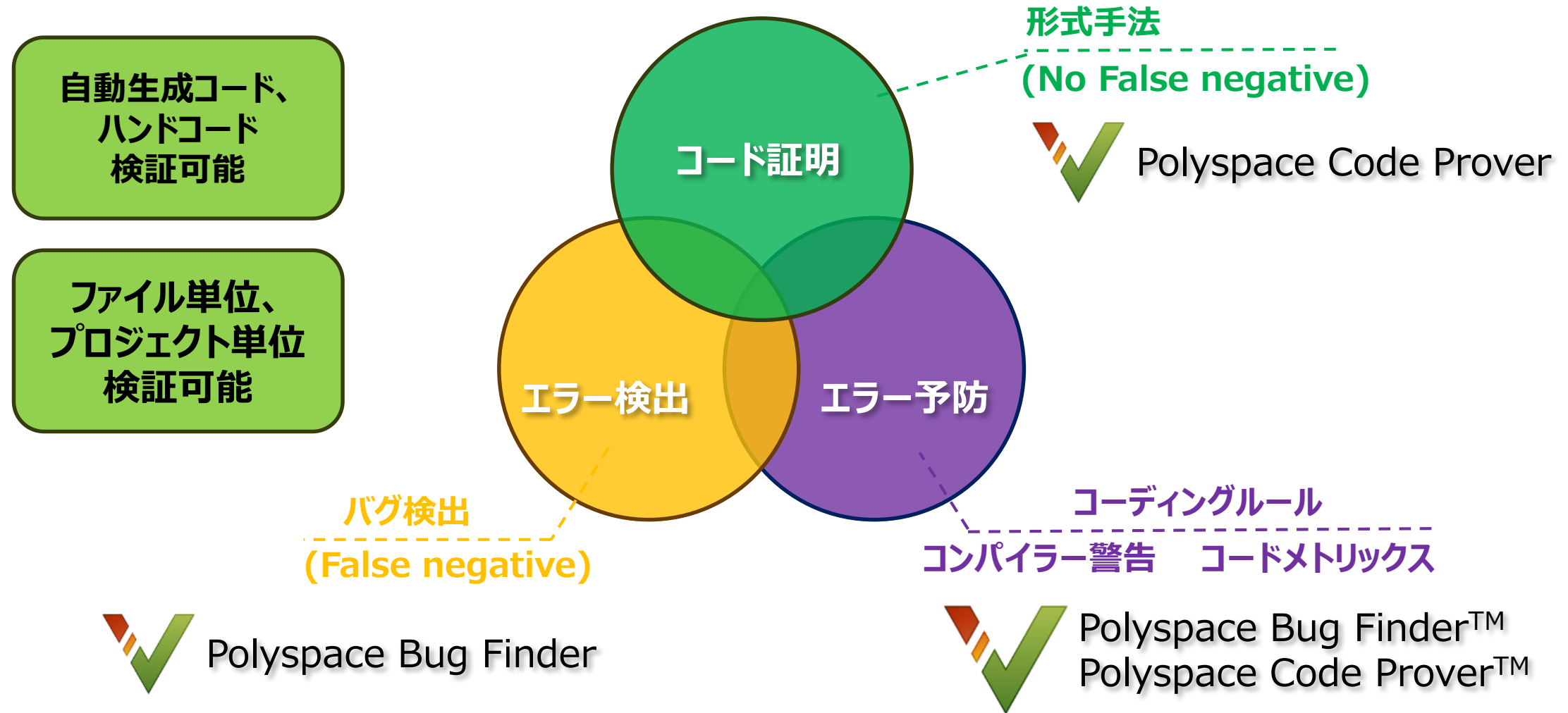
- ソフトウェア検証に関するチャレンジ
- Polyspace静的解析の解決案
- Polyspaceのメリット
- ユーザー事例 & まとめ

```
1 int new_position(int sensor_pos1, int sensor_pos2)
2 {
3     int actuator_position;
4     int x, y, tmp, magnitude;
5
6     actuator_position = 2; /* default*/
7     tmp = 0; /* values */
8     magnitude = sensor_pos1 / 100;
9     y = magnitude + 5;
10
11     while (actuator_position < 10)
12     {
13         actuator_position++;
14         tmp += sensor_pos2 / 100;
15         y += 3;
16     }
17     if ((3*magnitude + 100) >= 43)
18     {
19         magnitude++;
20         x = actuator_position;
21         actuator_position = x / (x - y);
22     }
23     return actuator_position*magnitude;
24 }
```

operator / on type int 32
left: 10
right: [-21474855 .. -1]
result: [-10 .. 0]

Polyspaceの静的解析ソリューション

コードの信頼性を確保するための機能を提供



Polyspace 沿革

- 1999年に PolySpace が製品化され、2007年に MathWorks 製品に加わる
- 1996年から実用化の段階に
 - Ariane 5 のソフトウェアでの実証
- 300以上のユーザーにおいて2000以上のライセンスが使用される



Polyspace ユーザーリスト

- 航空宇宙・防衛
- 自動車
- 産業装置
- 交通・運輸
- 家電製品
- 医療機器

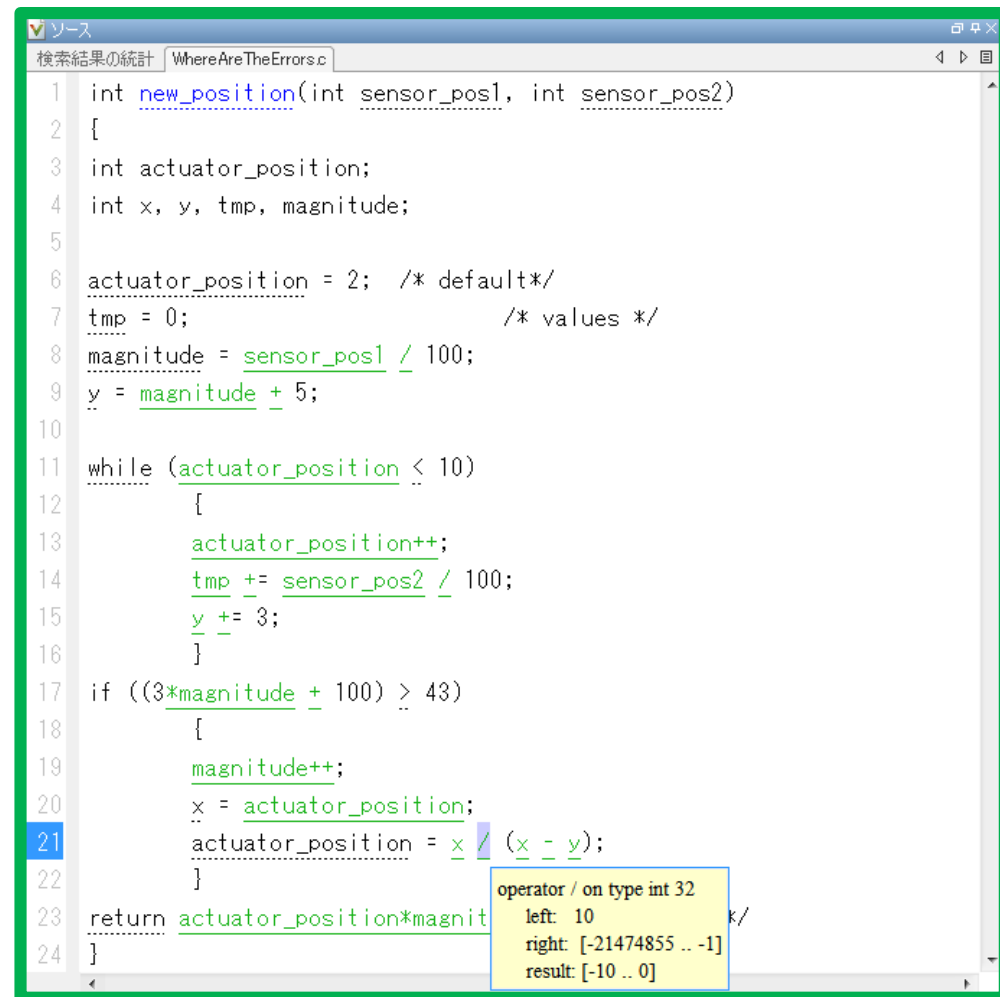


アジェンダ

- Polyspaceについて

➡ ソフトウェア検証に関するチャレンジ

- Polyspace静的解析の解決案
- Polyspaceのメリット
- ユーザー事例 & まとめ



```
1 int new_position(int sensor_pos1, int sensor_pos2)
2 {
3     int actuator_position;
4     int x, y, tmp, magnitude;
5
6     actuator_position = 2; /* default*/
7     tmp = 0; /* values */
8     magnitude = sensor_pos1 / 100;
9     y = magnitude + 5;
10
11     while (actuator_position < 10)
12     {
13         actuator_position++;
14         tmp += sensor_pos2 / 100;
15         y += 3;
16     }
17     if ((3*magnitude + 100) >= 43)
18     {
19         magnitude++;
20         x = actuator_position;
21         actuator_position = x / (x - y);
22     }
23     return actuator_position*magnitude;
24 }
```

operator / on type int 32
left: 10
right: [-21474855 .. -1]
result: [-10 .. 0]

ランタイムエラーとは？

Polyspaceはランタイムエラーの有無を証明する！

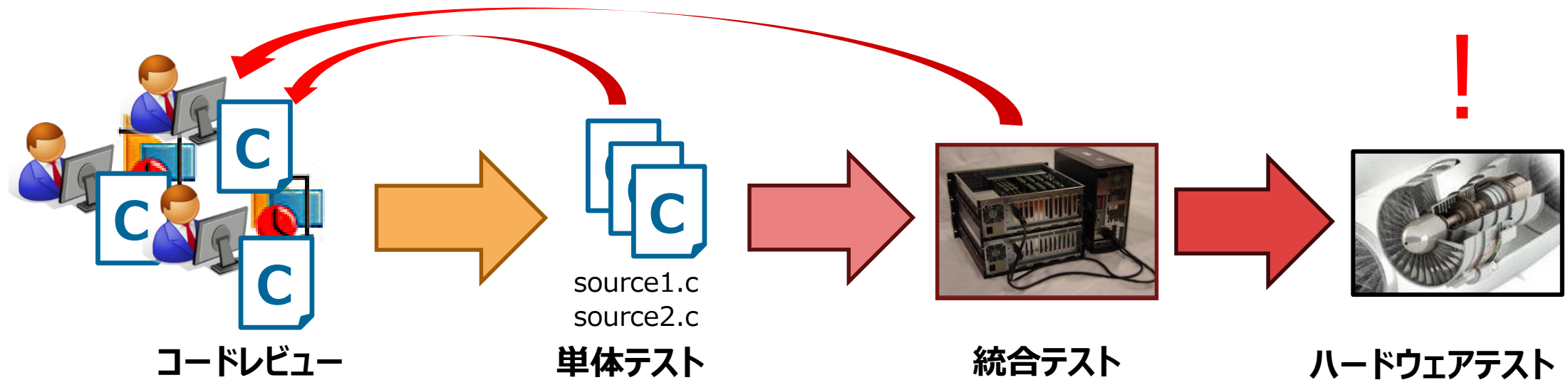
- プログラム実行中に発生するソフトウェアの不具合
- プログラムのクラッシュや暴走等をもたらす

- 未初期化変数の参照
- 配列外アクセス
- ゼロ割
- 不正なポインタ参照
(NULL参照、領域不足)
- オーバーフロー・アンダーフロー

- 不正な型変換
(小さな型への変換による
オーバーフロー)
- 不正な数学関数コール
(負数の平方根等)
- デッドコード
- 設計標準規約※

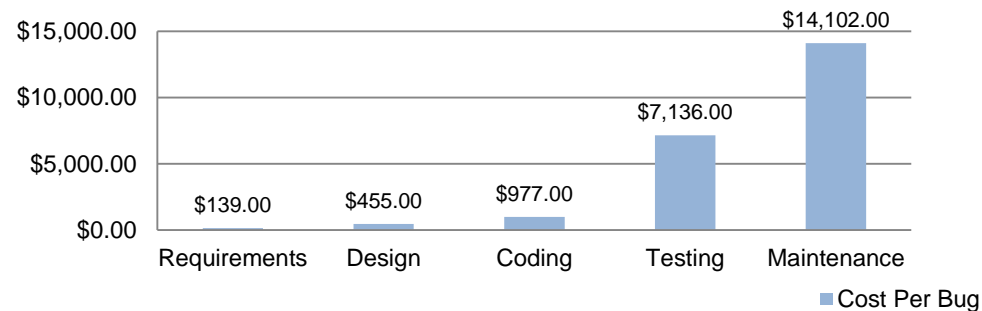
開発後期でのバグ検出の危険性

動的テスト完了後・製品リリース後にエラーが判明した！



Software Development Lifecycle Phase

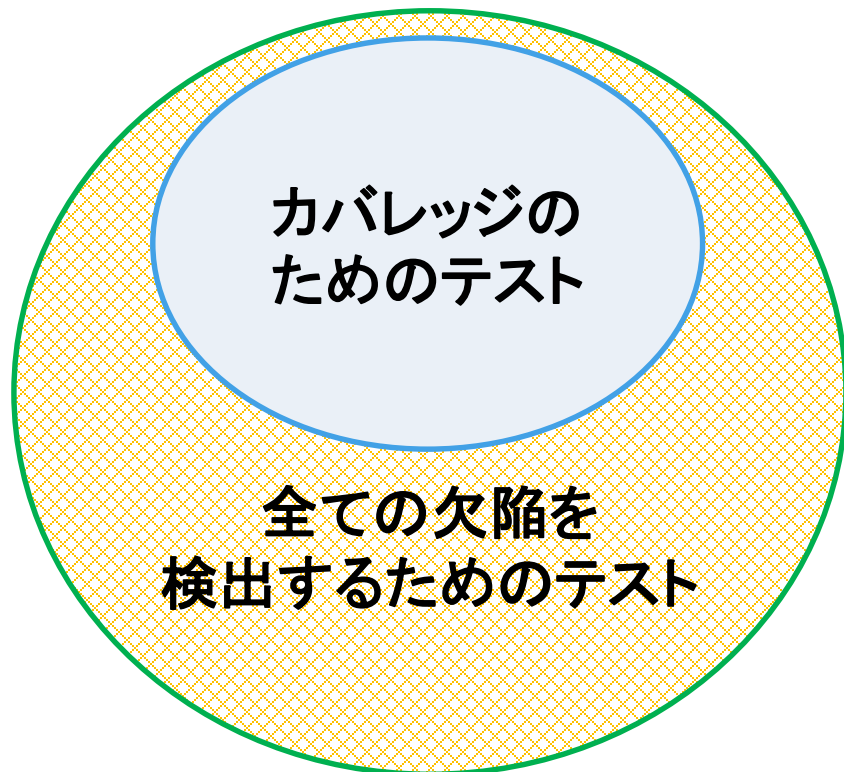
Source: B.Boehm and V. Basili :Software Defect Reduction Top 10 List", IEEE Computer



ソフトウェア品質確保の効率化に向けて

提案：静的解析によるソフトウェア品質の確保

- 従来のテスト手法 → テストケースの作成、実行、確認が必要



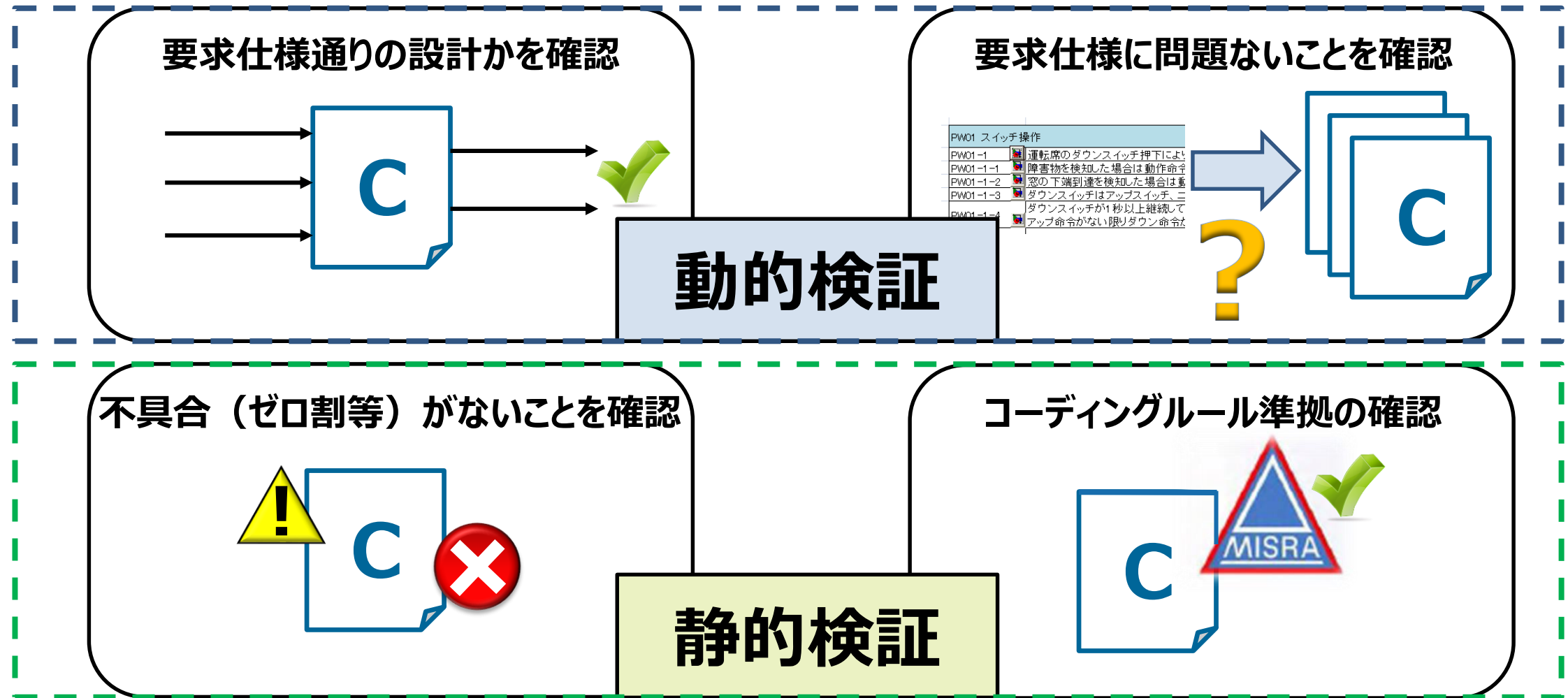
$$x / (x - y)$$

- 未初期化変数 `int32` の場合、
- オーバーフロー 4.61×10^{18} の
- ゼロ除算 テストケースが可能

全て可能なテストケースの
作成・実行・レビューは不可能！

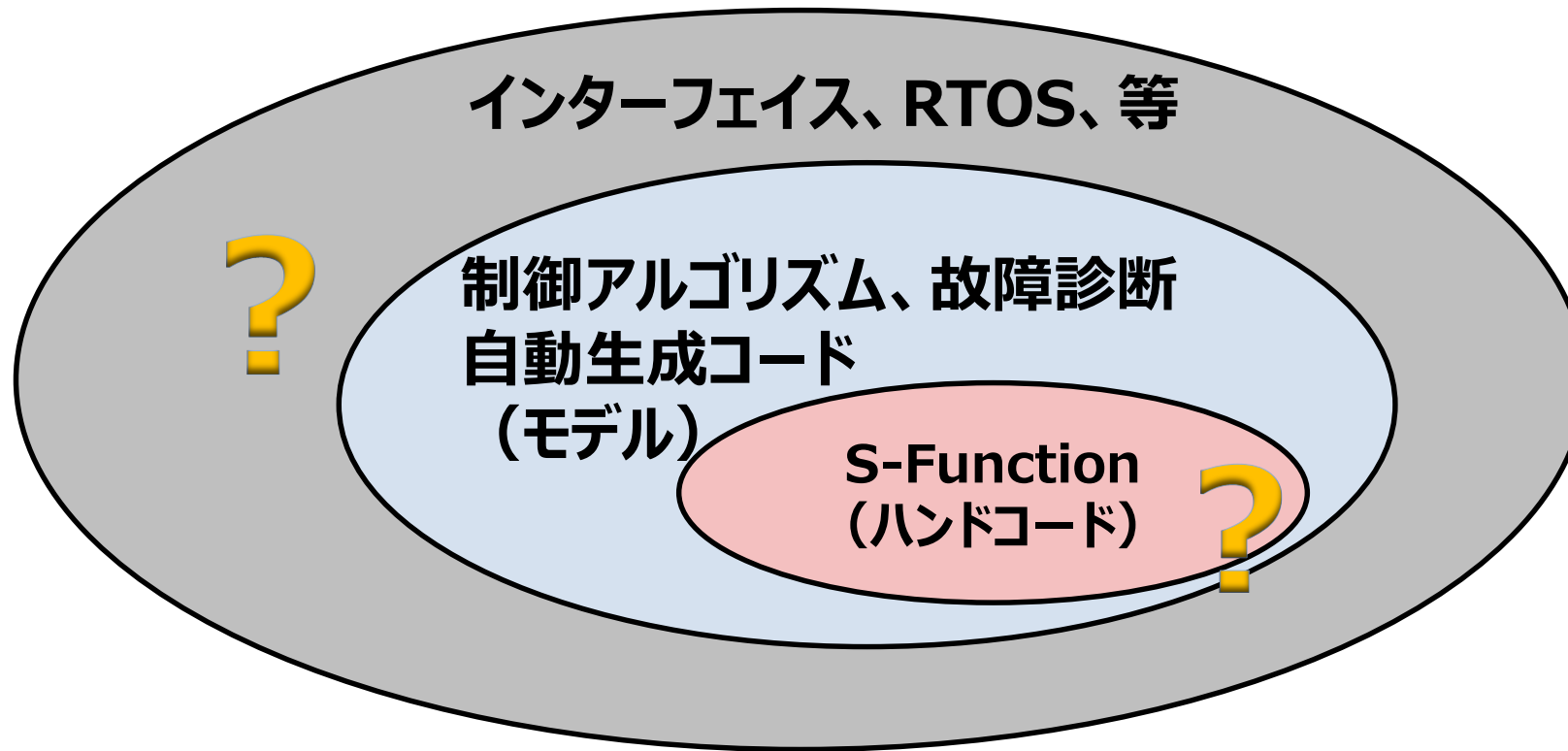
コード検証の目的

適切な検証手法を使用してプロセスを効率化



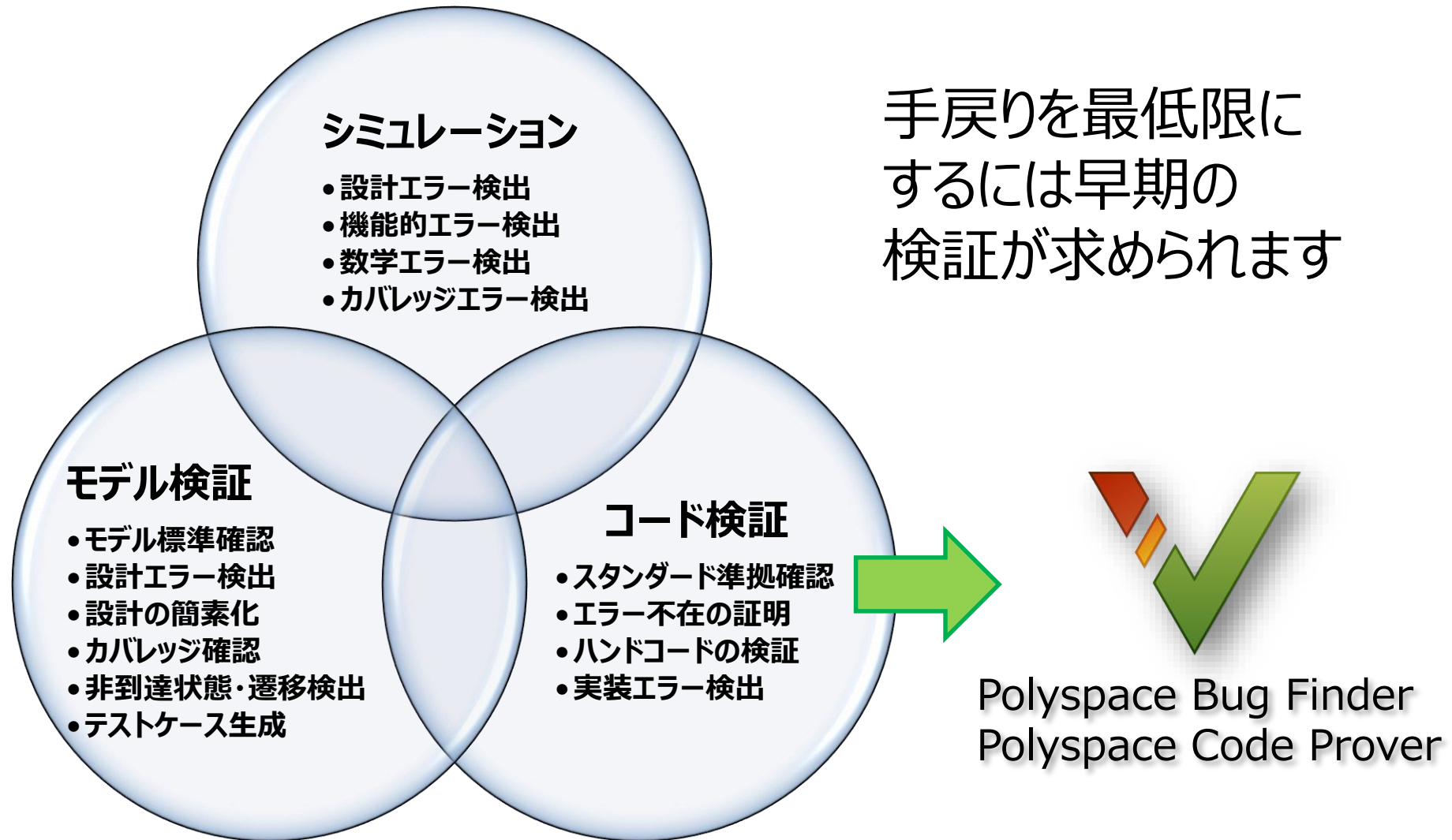
モデルベースデザインでも

自動生成コードとハンドコードと統合していませんか？



コード検証が必要になります

モデル検証とコード検証の役割分担



アジェンダ

- Polyspaceについて
- ソフトウェア検証に関するチャレンジ

➡ Polyspace静的解析の解決案

- Polyspaceのメリット
- ユーザー事例 & まとめ

```
1 int new_position(int sensor_pos1, int sensor_pos2)
2 {
3     int actuator_position;
4     int x, y, tmp, magnitude;
5
6     actuator_position = 2; /* default*/
7     tmp = 0; /* values */
8     magnitude = sensor_pos1 / 100;
9     y = magnitude + 5;
10
11     while (actuator_position < 10)
12     {
13         actuator_position++;
14         tmp += sensor_pos2 / 100;
15         y += 3;
16     }
17     if ((3*magnitude + 100) >= 43)
18     {
19         magnitude++;
20         x = actuator_position;
21         actuator_position = x / (x - y);
22     }
23     return actuator_position*magnitude;
24 }
```

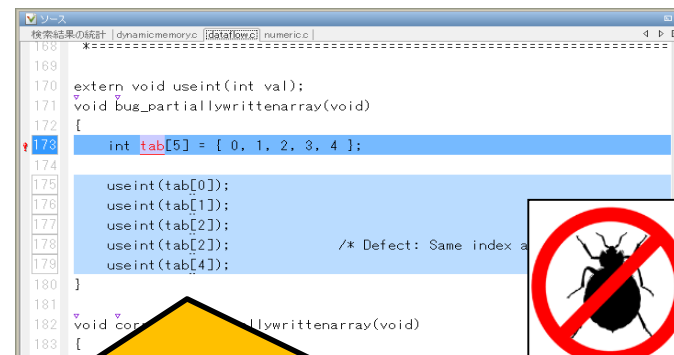
operator / on type int 32
left: 10
right: [-21474855 .. -1]
result: [-10 .. 0]



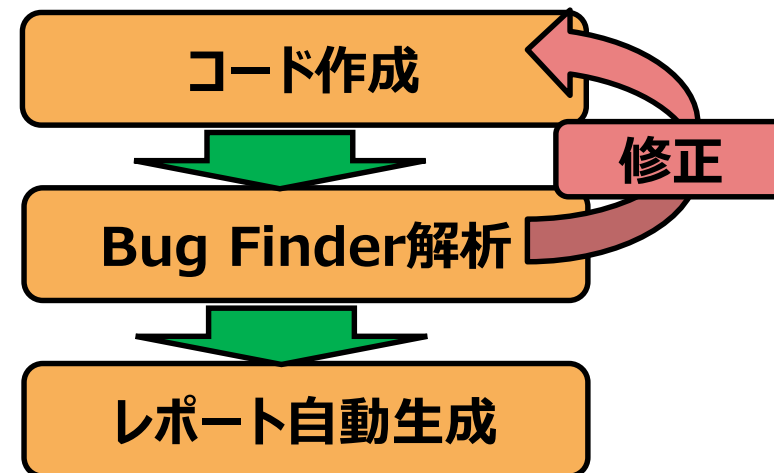
Polyspace Bug Finder のスピーディー解析 素早い欠陥検出・修正を可能とする！

- セキュリティ脆弱性・バグ検出
 - コード作成後、すぐに欠陥を検出・修正
- コーディングルールチェック
 - エラー予防と再利用性向上
 - MISRA準拠を確認
- コードメトリクス解析
 - コードの複雑度を測定

開発プロセスの上流で不具合を発見！
コードを統合前に多くの欠陥を修正！
コード開発の効率に繋がる！



時間を掛けずに大部分のバグを識別



セキュリティ脆弱性・バグ検出項目 (抜粋)

数値

- ゼロ割、オーバーフロー
- 標準ライブラリ数学ルーチンの無効な使用
- ...

動的メモリ

- メモリリーク
- 前に解放したポインターの使用
- ...

データフロー

- 読取りのない書込み
- 未初期化変数
- ...

リソース管理

- 読取り専用リソースに書込み
- 以前に閉じられたリソースを使用
- ...

静的メモリ

- 配列の範囲外アクセス
- NULLポインター
- ...

セキュリティ・暗号化

- パス操作が脆弱
- 疑似乱数発生器が脆弱
- ...

同時実行

- データレース
- デッドロック
- ...

適切な手法

- 未使用のパラメータ
- 値渡しの大きな引数
- ...

プログラミング

- 等号演算子による浮動小数点の比較
- 宣言の不一致
- ...

汚染されたデータ

- 汚染された文字列形式
- 汚染されたサイズでのメモリの割り当て
- ...



サイバーセキュリティ – 業界活動と標準

自動車業界に見られるコーディング標準 & 実践

- **CERT C** : セキュアコーディングスタンダード
- **ISO/IEC TS 17961** : Cセキュアコーディングルール
- **CWE** : 共通脆弱性タイプ
- **MISRA-C:2012** Amendment 1 : セキュリティガイドライン



コーディングルールチェック

- **MISRA C[®] : 2004** チェッカー
- **MISRA C[®] : 2004 AC AGC** チェッカー
 - 自動生成コード用
- **MISRA C[®] : 2012** チェッカー
- **MISRA[®] C++** チェッカー
- **JSF[®]++** チェッカー



MISRA C:2012 Amendment 1

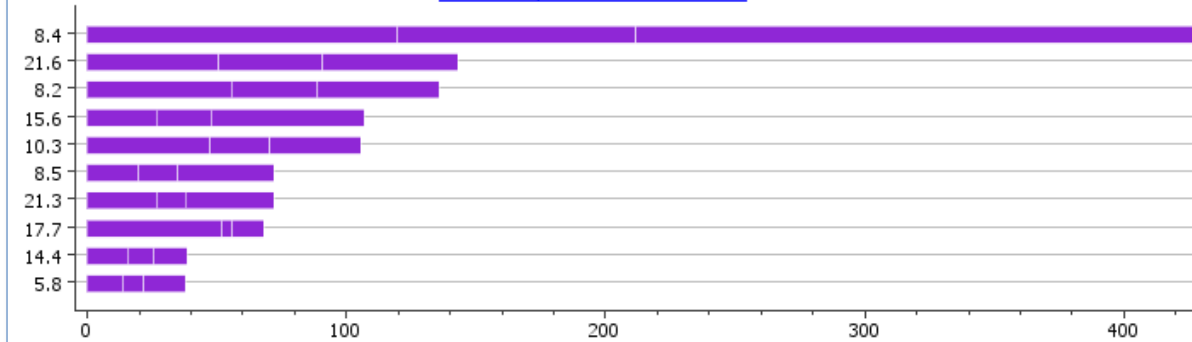
Additional security
guidelines for MISRA C:2012

April 2016

- ✓ **新しいコーディング規約** に対応
(全ての required と mandatory)
- ✓ **新しい指針 (directive) : 1**
- ✓ **新しいルール (rules) : 13**
- ✓ 既存のルール変更 [21.8]

ルール別の MISRA C:2012 違反 (上位 10 件のみ)

違反総数: 1,561 件が見つかりました





Polyspace Code Prover でコードの正しさを証明 全ての実行パスの結果を証明する！

- **Quality (品質)**
 - ランタイムエラーの証明
 - 測定、向上、管理
- **Usage (使用方法)**
 - コンパイル、プログラム実行、テストケースは不要
 - 対応言語：C/C++/Ada
- **Process (プロセス)**
 - ランタイムエラーの早期検出
 - 自動生成コード、ハンドコードの解析可能
 - コードの信頼性を測定

実機実験前にコード信頼性を
確保して手戻りを削減

グリーン：正常

ランタイムエラーが存在しない

レッド：エラー

実行される度にランタイムエラー

グレー：デッドコード

無実行

オレンジ：Unproven

条件によってランタイムエラー

パープル：Violation

MISRA-C/C++, JSF++

変数値範囲

ツールチップ

```
static void pointer_arithmetic (void) {
    int array[100];
    int *p = array;
    int i;

    for (i = 0; i < 100; i++) {
        *p = 0;
        p++;
    }

    if (get_bus_status() > 0) {
        if (get_oil_pressure() > 0) {
            *p = 5;
        } else {
            i++;
        }
    }

    i = get_bu

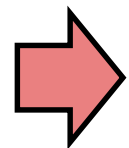
    if (i >= 0) {
        *(p - i) = 10;
    }
}
```

Dereference of local pointer 'p'
(pointer to int 32, size: 32bits):

Points to 4 bytes at offset 400 in
buffer of 400 bytes, so is outside
bounds.

アジェンダ

- Polyspaceについて
- ソフトウェア検証に関するチャレンジ
- Polyspace静的解析の解決案



Polyspaceのメリット

- ユーザー事例 & まとめ

```
1 int new_position(int sensor_pos1, int sensor_pos2)
2 {
3     int actuator_position;
4     int x, y, tmp, magnitude;
5
6     actuator_position = 2; /* default*/
7     tmp = 0; /* values */
8     magnitude = sensor_pos1 / 100;
9     y = magnitude + 5;
10
11     while (actuator_position < 10)
12     {
13         actuator_position++;
14         tmp += sensor_pos2 / 100;
15         y += 3;
16     }
17     if ((3*magnitude + 100) >= 43)
18     {
19         magnitude++;
20         x = actuator_position;
21         actuator_position = x / (x - y);
22     }
23     return actuator_position*magnitude;
24 }
```

operator / on type int 32
left: 10
right: [-21474855 .. -1]
result: [-10 .. 0]

開発プロセスでの検証項目 サンプルワークフロー

要求

設計

システムテスト

- ソフトウェア品質の測定
- 品質レポートの生成

結合テスト

Polyspace Bug Finder™
による**バグ/セキュリティ脆弱性の検出**

- ハグの検出
- MISRAルール違反の検出
- 未使用関数の検出

Polyspace Code Prover™
による**安全性の証明**

- 実行時エラー/未使用コードを見つける
- 実行時エラーがないことを証明する
- MISRA違反を正当化する

MISRAルール違反の安全性の証明

MISRA
ルール違反

安全性の証明
(オーバーフロー無し)

```
13 float risk_of_floatdivisionbyzero(int p)
```

Check Details

Misra_C2004.c / risk_of_floatdivisionby

▼ ID 7: MISRA C:2004 10.1
The value of an expression of integer type shall not be implicitly converted to a different underlying type.
Implicit conversion of the binary - right hand operand of underlying type 'signed int' to 'float' that is not an integer type. (Required)

✓ ID 130: Overflow
Operation [-] on float does not overflow in FLOAT32 range
operator - on type float 32
left: 1.0
right: [-2.1475E+09 .. -0.9999] or [0.0 .. 2.1475E+09]
result: [-2.1475E+09 .. 2.1475E+09]

```
19 tmp = j - p;
```

Check Review

▼ MISRA C:2004 10.1

Classification
Not a defect

Status
No action planned

☒ Justified

proven correct

Polyspace Code Prover

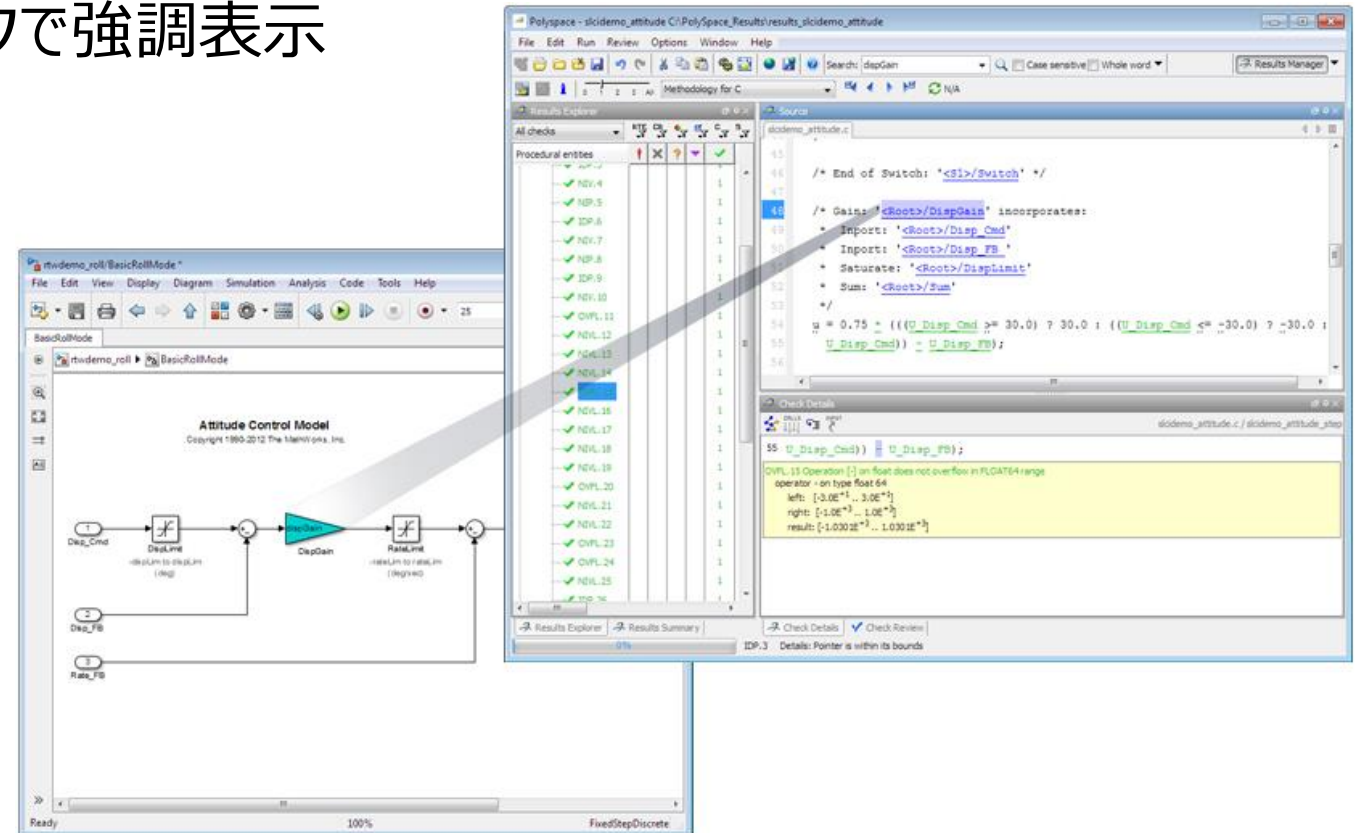
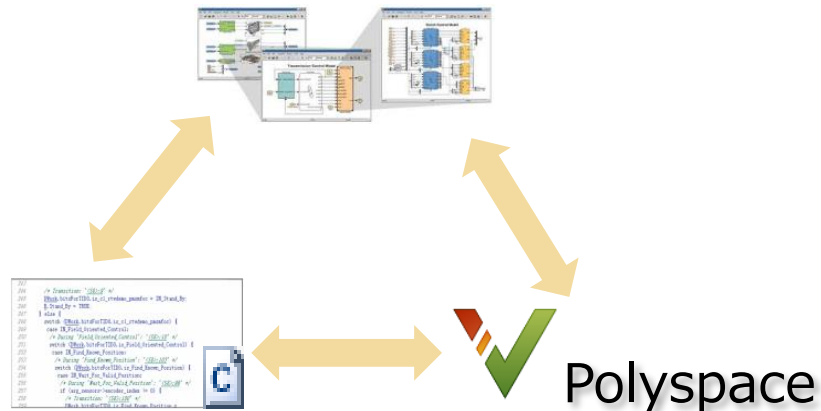
MISRAルール違反の正当化の
根拠としてPolyspaceを活用可能

```
24 } else {  
25     i = 0.0;  
26 }  
27 return i;  
28 }
```

モデルベースデザインとの統合

PolyspaceのSimulink連携機能でモデル修正が容易

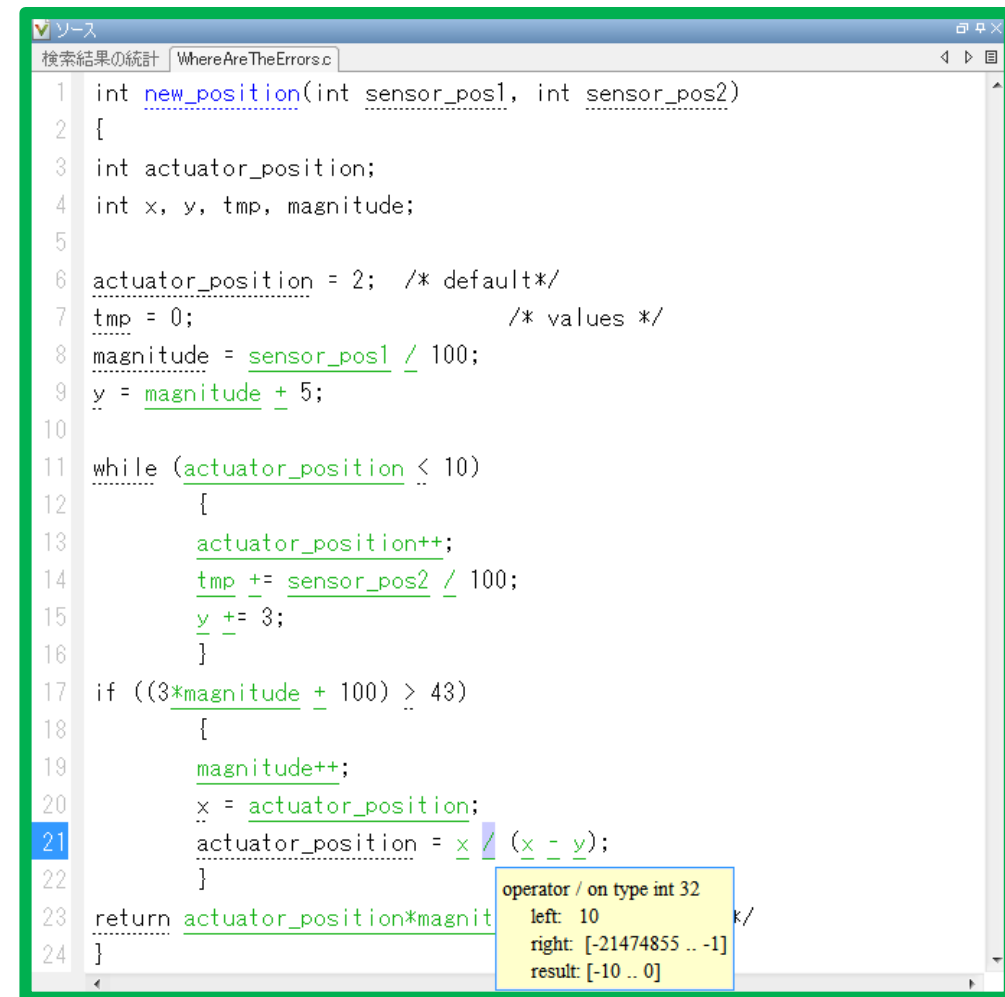
- Simulink®環境からPolyspace解析を実行
- Polyspace解析結果を該当ブロックで強調表示
- ブロックにPolyspaceの注釈を追加
- S-Functionコードの検証
- MISRA C のチェック



アジェンダ

- Polyspaceについて
- ソフトウェア検証に関するチャレンジ
- Polyspace静的解析の解決案
- Polyspaceのメリット

➡ ユーザー事例 & まとめ



```
1 int new_position(int sensor_pos1, int sensor_pos2)
2 {
3     int actuator_position;
4     int x, y, tmp, magnitude;
5
6     actuator_position = 2; /* default*/
7     tmp = 0; /* values */
8     magnitude = sensor_pos1 / 100;
9     y = magnitude + 5;
10
11     while (actuator_position < 10)
12     {
13         actuator_position++;
14         tmp += sensor_pos2 / 100;
15         y += 3;
16     }
17     if ((3*magnitude + 100) >= 43)
18     {
19         magnitude++;
20         x = actuator_position;
21         actuator_position = x / (x - y);
22     }
23     return actuator_position*magnitude;
24 }
```

operator / on type int 32
left: 10
right: [-21474855 .. -1]
result: [-10 .. 0]

Polyspace ユーザー事例

医療、航空、自動車など様々な分野で利用されています

Miracor Eliminates Run-Time Errors and Reduces Testing Time for Class III Medical Device Software



“From a developer's perspective, the main advantage of Polyspace Code Prover is a higher level of quality and correctness in the code. Polyspace Code Prover helps Miracor demonstrate this quality and correctness to the regulatory community, including the FDA, to prove that our device is safe.”

- Lars Schiemanck, Miracor Medical Systems

https://jp.mathworks.com/company/user_stories/miracor-eliminates-run-time-errors-and-reduces-testing-time-for-class-iii-medical-device-software.html

Solar Impulse Develops Advanced Solar-Powered Airplane



“From a developer's perspective, the main advantage of Polyspace Code Prover is a higher level of quality and correctness in the code. Polyspace Code Prover helps Miracor demonstrate this quality and correctness to the regulatory community, including the FDA, to prove that our device is safe.”

- Lars Schiemanck, Miracor Medical Systems

https://jp.mathworks.com/company/user_stories/solar-impulse-develops-advanced-solar-powered-airplane.html

日産自動車 ソフトウェアの信頼性を向上

「Polyspace 製品によって、高レベルなソフトウェアの信頼性を確保することができます。これは、業界内の他のツールにはできないことです。」

- 菊池光彦氏, 日産自動車



課題

ソフトウェア品質を向上させるために、発見が困難なランタイムエラーを特定する

ソリューション

MathWorks のPolyspace製品を使用して、日産とサプライヤーのコードを包括的に解析する

結果

サプライヤーのバグを検出して評価

ソフトウェアの信頼性が向上

日産のサプライヤーが Polyspace 製品を採用

https://jp.mathworks.com/company/user_stories/nissan-increases-software-reliability.html

規格準拠に向けて Polyspace製品で目標規格達成をアシストします！

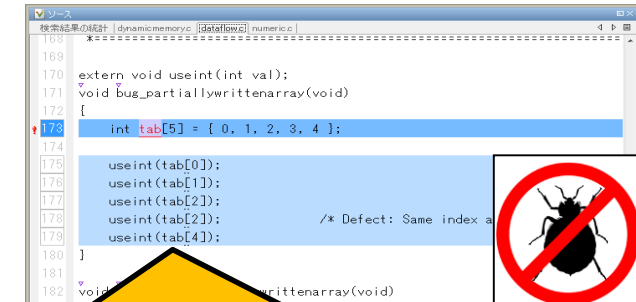
- コーディング規約
 - MISRA-C
 - MISRA-C++
 - JSF++
- ソフトウェアメトリクス
 - HIS Source Code Metrics
 - <http://www.automotive-his.de/>
- 認証規格
 - DO-178B
 - DO Qualification Kit
 - IEC 61508, ISO 26262, EN 50128
 - IEC Certification Kit



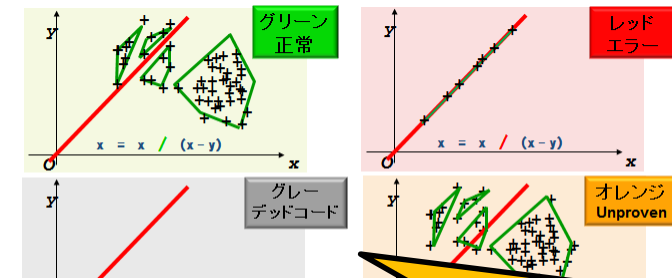
Polyspace ソースコード静的検証 ～まとめ～

- Polyspace Bug Finderで
素早くコードの欠陥を検出！
早期段階で不具合の修正が可能！
- Polyspace Code Proverで
クリティカルシステムにランタイム
エラーの有無を証明！
- 両ツールを使用して
効率的にソフトウェアの品質を
管理・向上！

Simulink環境から実行可能
ファイル・プロジェクト単位で使用可能

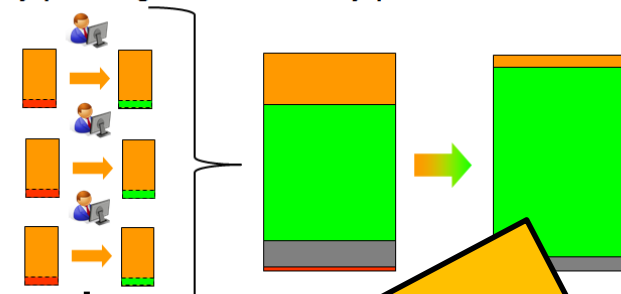


時間を掛けずに大部分のバグを識別



形式手法の解析によるコード信頼性

Polyspace Bug Finder → Polyspace Code Prover



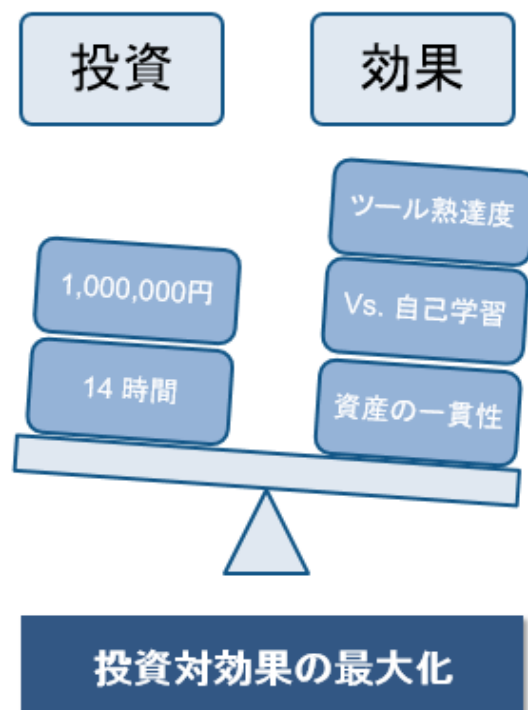
安全なコードを確保して実機実験へ

Polyspace の有効活用に向けて 短期間で習得していただけるような教育カリキュラムをご提供します

MathWorks | Training Services

Polyspace® 検証の紹介

Polyspace® Code Prover™ による C/C++ コードの検証



■ トレーニング サービス

➤ 定期 トレーニング

- ✓ 東京、名古屋、大阪にて定期開催
- ✓ 基礎コース(11)、応用コース(11)、専門コース(4)をご提供

➤ オンサイト トレーニング

- ✓ お客様サイトにて開催
- ✓ ご要望に応じて3つのレベルでカリキュラムのカスタマイズが可能

■ コンサルティング サービス

➤ カスタム “Jumpstart”

- ✓ 顧客モデルをベースにした短期集中型ツール導入サポート

➤ Advisory Service

- ✓ 顧客Projectに合わせた中長期アドバイザリサービス

ご清聴ありがとうございました



Accelerating the pace of engineering and science

© 2018 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.