DENSO
Crafting the Core

派生開発における影響分析の自動化に向けた取り組み事例 ～解析指向プログラミングアプローチの考案～

柏原一雄

株式会社デンソークリエイト

目次

1. はじめに

- 派生開発において，変更に対する影響範囲の特定漏れを防ぐために，ベースソフトに対するコード解析を自動化する.

- 既存の静的コード解析ツールでは解析が困難な関数ポインタコールやプリプロセッサ命令を，解析可能とする必要がある.

- コード解析の自動化は，高度な静的コード解析ツールにより実現するのではなく，ツールで解析可能なコーディングをすることで実現する.

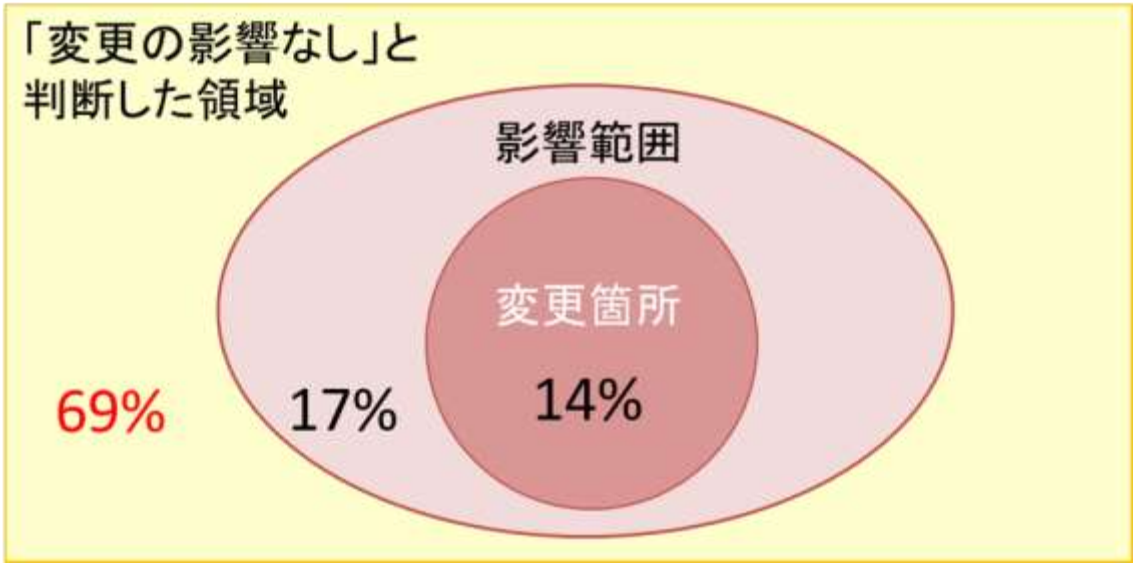
- ツールが解析可能なコーディングをするために，コーディング基準を定義し，コーディング基準を定着・改善するための仕組みを構築した.

2. 現状分析（1）

■ 自組織の欠陥流出の傾向

- 影響範囲の特定漏れに分類される欠陥が多い.
- 影響範囲を特定するために，静的コード解析をベースとした影響分析手法を利用している.

【自組織における欠陥流出領域の分布】

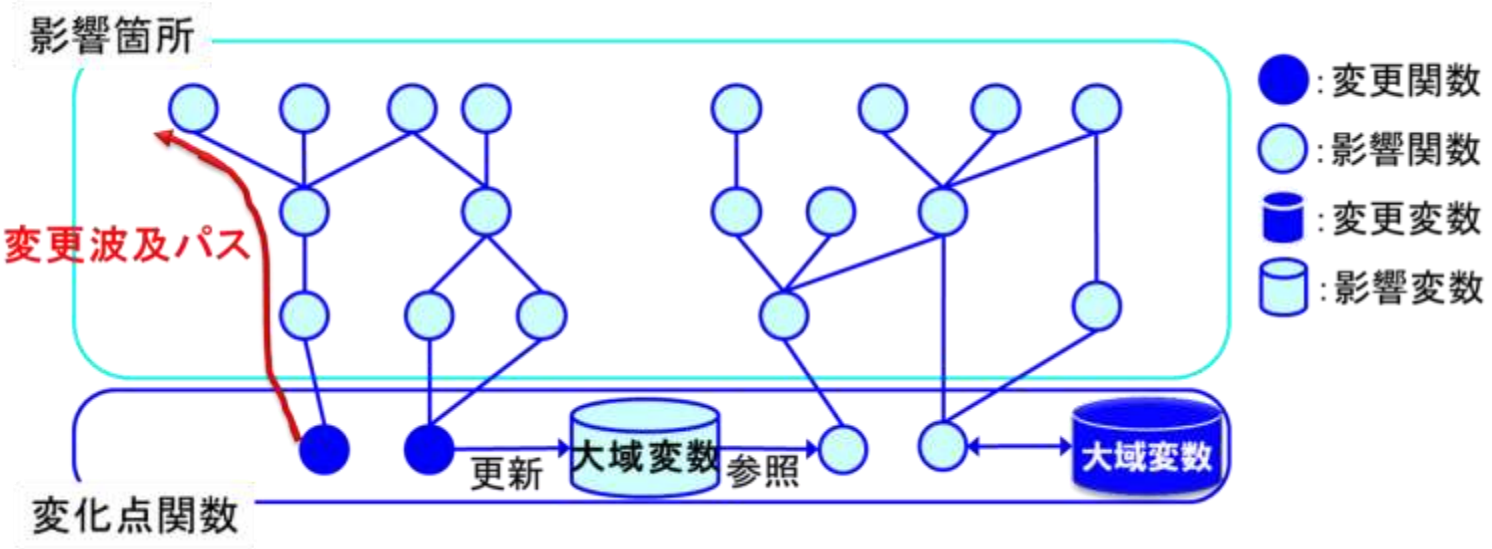


**変更に対する影響範囲を確実に特定するため，
静的コード解析を活用する**

2. 現状分析（2）

■ 利用している影響分析手法：
影響波及パス分析法[1]

- 影響波及パス分析法は，想定外のデグレード可能域を絞り込み，未変更箇所で発生するデグレードを防止することを目的とした手法である.
- 変更箇所を起点とした関数コールの繋がりを表現した「影響波及パス」を利用する．影響波及パスの表現方法のイメージを以下に示す.
  [影響関数名(公開関数)]→[影響関数名]→・・・→[変化点関数名]



2. 現状分析（3）

■ 利用している影響分析手法：
CRUDマトリクスによる設計影響分析[2]

- 「CRUDマトリクス」とは，機能とデータを2軸にもつマトリクスであり，機能によるデータアクセス仕様を可視化したものである．
- CRUDマトリクスによる設計影響分析の手法は，機能を関数に，データを変数に置き換えることによって，関数による変数アクセス仕様を表形式で可視化し，関数と変数の相互作用を分析する手法である．
- 具体的には，公開関数と外部変数の関係を可視化している．

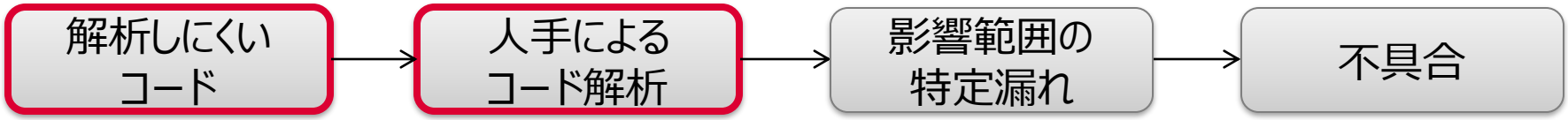
機能名	関数コールツリー	外部変数		
		外部変数1	外部変数2	外部変数3
A機能	FuncA	-	-	-
	FuncB	-	-	WR
	FuncC	W	-	-
	FuncD	-	-	-
	FuncE	-	W	-
B機能	FuncF	-	-	-
	FuncG	-	-	-
	FuncH	-	-	-
	FuncI	R	R	-

W：書き込み（Write）
R ：読み込み（Read）

2. 現状分析（4）

■ 静的コード解析を利用した影響分析の問題

- ベースソフトは，既存のツールでは解析が困難な関数ポインタコールやプリプロセッサ命令を使用しており，人手による解析が必要となる.
- 人手でソースコードを解析していることが原因で，影響範囲の特定漏れが起きやすくなる.
  - データを参照する関数は複数あり，その関数を呼び出す関数も複数ある. 人手による解析では，理解する対象が増えることにより，見逃しも増える.
  - 短期間での開発が求められる場合には，影響分析する範囲を絞り込むことも多い. 人手による解析では，絞り込み時に判断を誤ることがある.



ベースソフトの解析容易性を高め，
静的コード解析を自動化する必要がある

2. 現状分析（5）

■ 解析しにくいコードの例

- 多バリエーションのソフト開発のために，関数ポインタやプリプロセッサ命令で，機能・処理の切り替えを可能としている

【関数ポインタ】

pFunc()で呼び出す関数を切り替える

```
void funcA(){
  ・・・
  pFunc();
  ・・・
}
```

【プリプロセッサ命令】

funcB(),funcC()のどちらを呼び出すかを切り替える
（一方は無効コードとなる）

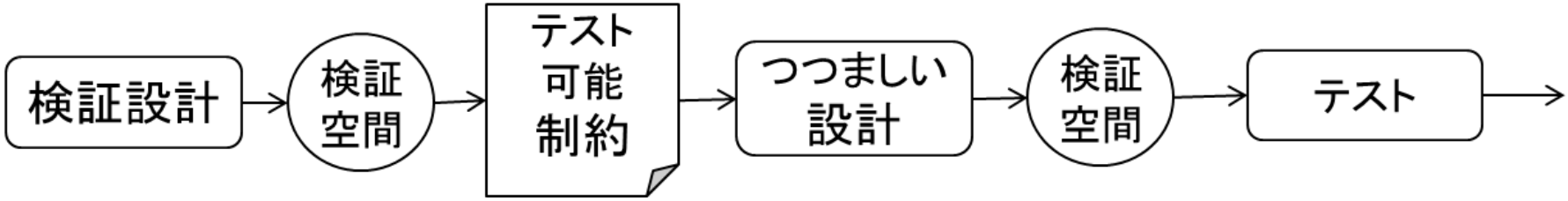
```
void funcA(void){
#if ( AAA == NOT_SUPPORT )
   funcB();
#else
   funcC();
#endif
}
```

3. 課題提起（1）

■ コード解析を自動化するために参考にした理論

- **検証指向設計**
  [3]松尾谷,「テストから観た実体のモデリングと実装構造の評価
  ～検証指向設計の実現に向けて～」,JaSST'15東海,2015

  – テストの実力にあった（実施可能なテスト項目数となる）設計を選ぶ
  – 検証空間（影響範囲）が小さくなる設計を選ぶ
  – 設計を選ぶための設計制約を設ける



テストの能力の範囲内で，設計する

3. 課題提起（2）

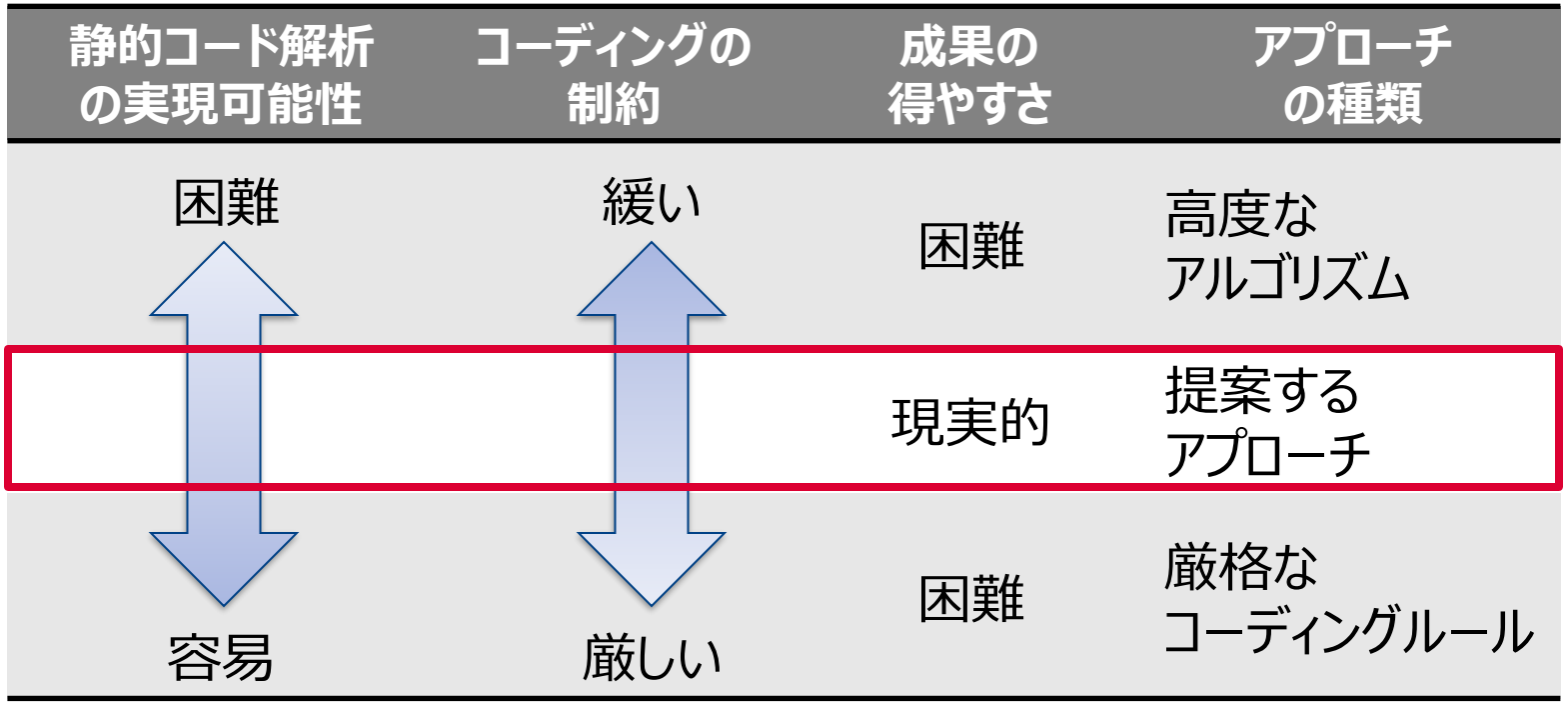
■ コード解析を自動化するためのアプローチ

- **解析指向プログラミングアプローチ**
  - どのようなソースコードでも解析可能な高度な静的コード解析ツールを用意するのではなく，静的コード解析ツールの能力の範囲内でコーディングをする.
  - 静的コード解析ツールで解析可能とするために，コーディング基準を定め，コーディング基準に従ったソースコードにする.
  - コーディング基準に従ったコーディングがされることで，ツールによるコード解析の自動化が可能となる.

検証指向設計を参考に発想転換
静的コード解析ツールの能力の範囲内で，コーディングする

3. 課題提起（3）

■ 解析指向プログラミングアプローチの位置づけ



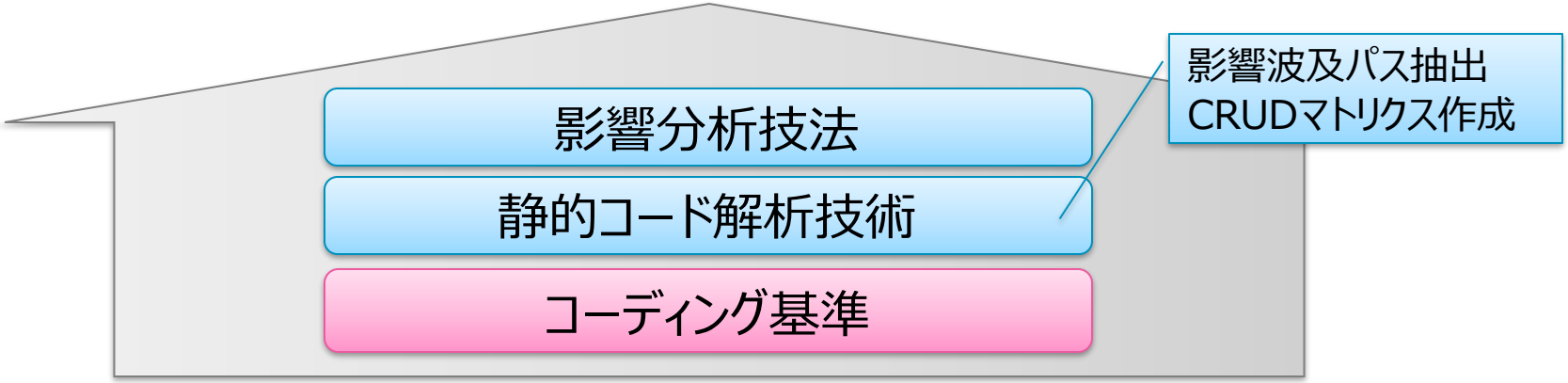
※厳格なコーディングルールの例：NASAのコーディング標準「The Power of 10」
- Rule8：プリプロセッサの利用はヘッダのincludeと簡単なマクロ定義に限定する
- Rule9：関数ポインタの使用は許可されない

確実に成果を得るための現実的なアプローチ

3. 課題提起（4）

■ 課題

- 解析指向プログラミングアプローチの考え方のもと，必要となる仕掛けを考案し，静的コード解析（影響波及パス抽出，CRUDマトリクス作成）の自動化を可能とする.



■ 課題解決のポイント

- 既存の静的コード解析ツールでは解析が困難な箇所へ対応する.
  - 関数ポインタコール
  - プリプロセッサ命令による処理切り替え
- コーディング基準は，成果を得るために，定着及び改善させる.

4. 解決策の提案（1）

■ 課題の解決方針（1）

- 関数ポインタコール・プリプロセッサ命令の解析のための仕掛け

A) 関数ポインタコール
関数ポインタコール箇所には，プリプロセッサ命令（#ifndef）を使って，直接コールの形式で，呼び出される可能性のある関数を併記する.

```
#ifndef CALL_ANALYSIS
  pFunc();
#else /* CALL_ANALYSIS */
   FuncX();
   FuncY();
#endif /* CALL_ANALYSIS */
```

B) プリプロセッサ命令による処理切り替え
プリプロセッサ命令の書き方を統一する.
・関数の外にプリプロセッサ命令を記述
・#ifのみ使用可能
等

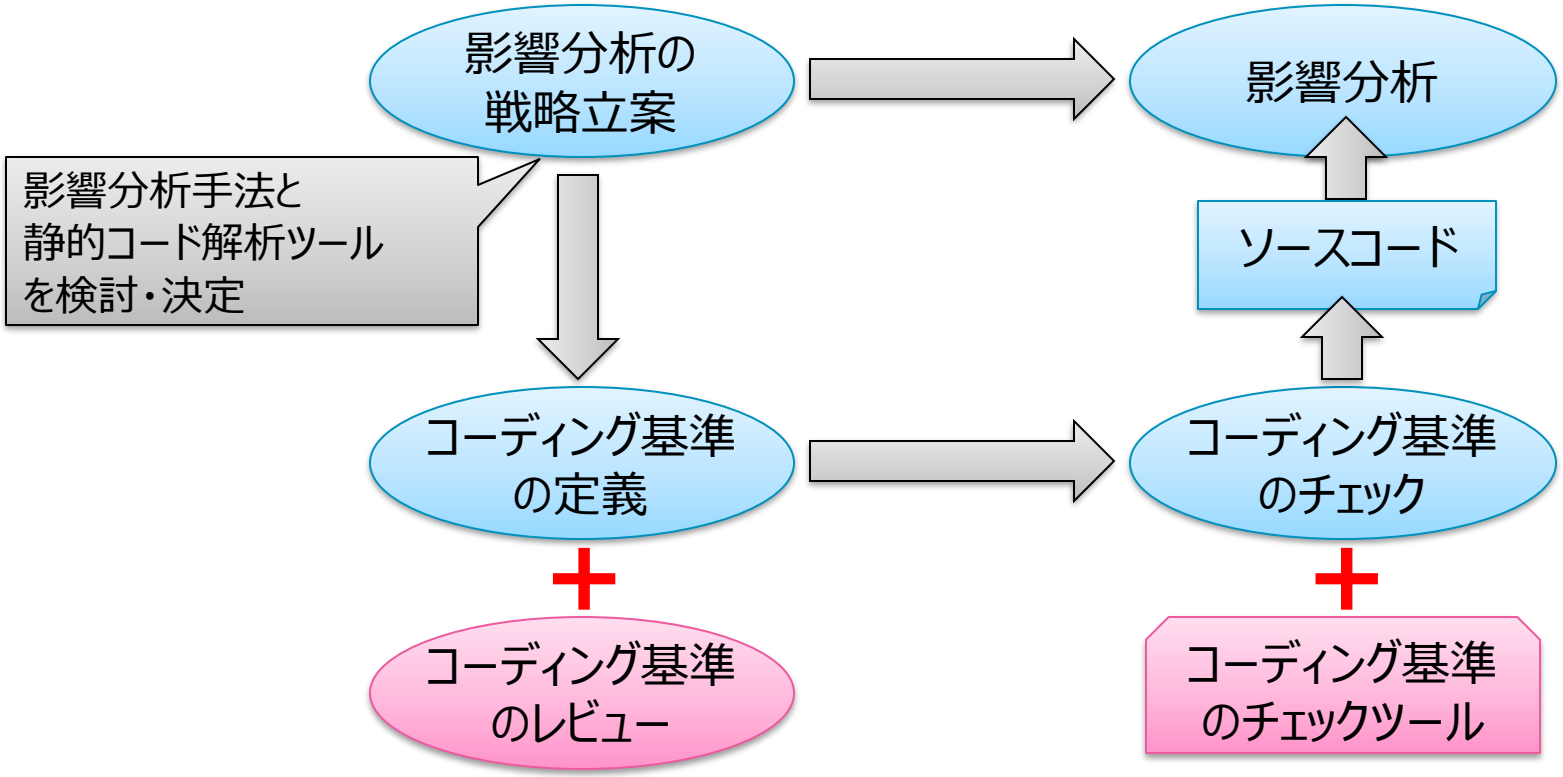
```
#if ( AAA == NOT_SUPPORT )
Void funcB(void){
}
#endif

#if ( AAA == SUPPORT )
Void funcB(void){
}
#endif
```

更新のしやすさも考慮し，
必要となる設計情報を解析可能な形でソースコード内に残す

4. 解決策の提案（2）

■ 課題の解決方針（2）

- コーディング基準の定着・改善の仕掛け

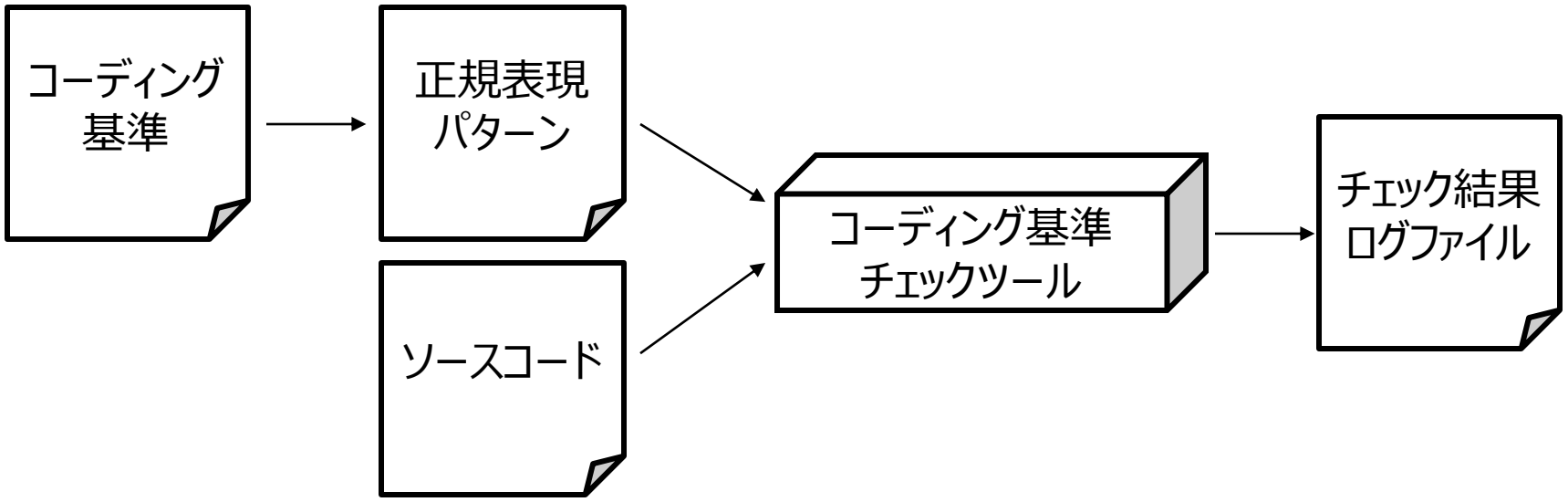


コーディング基準を定着させるためのチェックツールを用意
コーディング基準を改善するためのレビューを実施

4. 解決策の提案（3）

■ コーディング基準のチェックツール

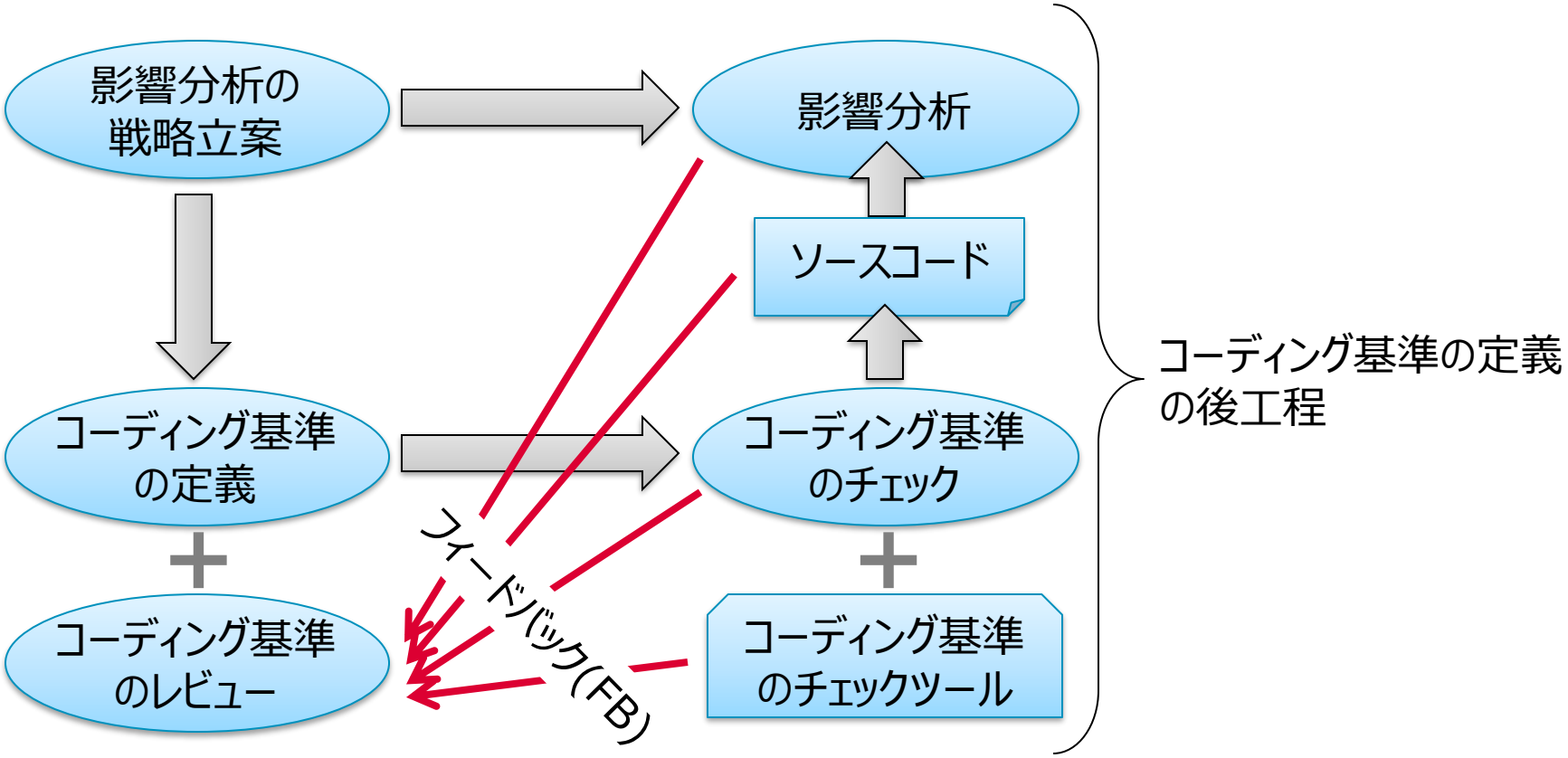
- 以下の要件を満たすツールを開発
  - 余分な警告が出力されない
  - CI（継続的インテグレーション）環境から実行可能
  - チェック結果ログファイルから警告箇所へのジャンプ可能
  - 保守しやすいように正規表現によるパターンマッチングを利用



開発者に負担をかけずチェック可能とする

4. 解決策の提案（4）

■ コーディング基準のレビュー（1）



後工程で成果を得るため，後工程から基準へのFBを得る

4. 解決策の提案（5）

■ コーディング基準のレビュー（2）

【コーディング基準のレビュー観点】

FB元	観点	確認事項
コーディング基準のチェックツール	チェックツールの実現容易性	コーディング基準チェックツールが開発しやすい（＝正規表現が利用しやすい）ルールであるかを確認する.
コーディング基準のチェック	チェックツールの使用性	コーディング基準チェックツールで，警告が余分な箇所で出力される場合がある. このような場合，コーディング基準の判定方法を見直す.
ソースコード	効率性 保守性	効率性・保守性を高めるために，基準を逸脱せざるを得ない（基準が邪魔になる）場合がある. このような場合，コーディング基準に例外条件を追加し，静的コード解析の制約とすることを検討する.
影響分析	合目的性	コーディング基準に準拠することで，静的コード解析ツールにより，影響波及パス抽出やCRUDマトリクス作成が可能となったかを確認する.

レビューを通して，後工程の関係者全員が納得する基準にする

4. 解決策の提案（6）

■ コーディング基準のレビュー（3）

【コーディング基準の定義要素】

基準内容	・コーディングのルールを示す． 適合例・不適合例を含む．
判定方法	・基準への適合を機械的に判定する条件・方法を示す． 基準チェックツールの仕様となる．
例外条件	・コーディングのルールの例外条件を示す． ルールを準拠すべきでない場合もあるため，その情報を示す．
目的	・基準を守ることで，得られる効果を示す． 自動化できる活動や削減できる検証活動等．

レビューで確認することを，コーディング基準の定義要素とする

5. 解決策の適用結果（1）

■ 手法の適用対象プロジェクト

- C言語で開発しているミドルウェア
- ソースコードの規模：50～100KLOC
- アーキテクチャを見直し，機能の新規開発を行うフェーズ
- 開発担当者10名以上

■ 手法の適用方法

1. 解析容易性を向上させるために，既存のコーディング基準に対して，新たに基準（8つ）を追加した.
2. 追加したコーディング基準を，チェックツールで自動チェック可能とした.
3. コーディング基準のレビューは，開発期間中に継続的に実施（開発者も参加）し，必要に応じてコーディング基準とチェックツールを修正した.
4. コーディング基準チェックツールを利用し，問題の検出・修正を実施した.
5. コーディング基準が遵守されたソースコードを前提に，影響波及パス抽出とCRUDマトリクス作成が可能なツールを開発した.

5. 解決策の適用結果（2）

■ 手法の適用結果

- コーディング基準の改善
  - コーディング基準は一度定義して終わりではなく，改善が繰り返された．
    （コーディング基準は，適用を始めてから半年間で合計7回の改訂）
- コーディング基準の定着
  - コーディング基準が，例外条件に適合する場合を除いてすべて遵守された．
- 静的コード解析の自動化
  - ソースコードに加え，一部設計情報（データ-アクセス関数マトリクス）を入力とし，影響波及パス抽出とCRUDマトリクス作成が自動実行可能となった．
  - ツール開発工数1人月に対して，コード解析工数は4人月削減した．

【データ-アクセス関数マトリクス】

データアクセス関数	データ		
	DataA	DataB	DataC
FuncA	WR		R
FuncB		W	
FuncC	R		W
FuncD		R	

W：書き込み（Write）
R ：読み込み（Read）

5. 解決策の適用結果（3）

■ 考察

- ソースコード内に設計情報を残すと，情報が維持されやすい.
- コーディング基準のチェックツール・レビューにより，コーディング基準を定着・改善することができる.
- コーディング基準を遵守した解析しやすいソースコードとなることで，高度なツールがなくても，静的コード解析による影響範囲抽出が可能となる.
- プロジェクトメンバの意識の変化も生まれる.
  メンバへのヒアリングの結果，以下の考え方の変化を確認した.
  - コーディング基準は与えられ守るものではなく，自分達で作り活用するもの
  - 人の目でチェックするのではなく，ツールでチェックできないか考える
  - ツール導入による作業自動化を妨げるような設計書・ソースコードがある場合は，設計書・ソースコードのほうを改善する

6. まとめ（1）

■ 成果

- コード解析を自動化し，影響範囲を確実に特定するために，解析指向プログラミングアプローチを考案した.
- 関数ポインタコール・プリプロセッサ命令の解析のために，必要な設計情報を解析可能な形でソースコード内に残す仕掛けを考案した.
- コーディング基準を定着・改善するために，必要となる仕掛けを考案した.
  - コーディング基準のチェックツール
  - コーディング基準のレビュー
- 解析指向プログラミングアプローチを適用することにより，影響波及パス抽出とCRUDマトリクス作成の自動化が進んだ.

派生開発における影響分析の自動化が進み，
影響範囲の特定漏れを防止する効果に繋がる見込み

6. まとめ（2）

■ 今後の進め方

- 静的コード解析技術の改善
  - データとアクセス関数の関係をソースコードから解析できるようにし，影響波及パス抽出とCRUDマトリクス作成を完全自動化する.
- 影響分析手法の改善
  - 影響箇所の見逃しを防止するために，CRUDマトリクス等の見方のノウハウを蓄積し活用する.
- 手法の効果確認
  - リリース後不具合やレビュー指摘を分析し，影響範囲の特定漏れが防止できる効果に繋がるかを確認する.
- 手法の展開
  - オープンソースの静的コード解析ツールの活用等を検討し，取り組み対象のプロジェクト以外でも影響分析手法を適用しやすい状態にする.

おわりに

■ 謝辞

- 本取り組みに対して有益なご助言を戴いた，
有限会社デバッグ工学研究所 松尾谷氏，堀田氏，
株式会社デンソー 足立氏，株式会社日立ハイテクノロジーズ 飯泉氏，
株式会社システムクリエイツ 清水氏，名古屋市工業研究所 小川氏，
株式会社デンソー技研センター 古畑氏，名古屋大学 森崎氏，
株式会社デンソー 中嶋氏，野田氏，
株式会社デンソークリエイト 辻村氏，山路氏，脇氏，竹下氏
に感謝の意を表します．

- 本取り組みに参加していただいたプロジェクトメンバ・プロジェクト関係者に
感謝の意を表します．

DENSO

Crafting the Core

参考文献

[1] 柏原一雄，「統合テストにおいて影響範囲に対するテスト漏れを防止する「影響波及パス分析法」の提案」，ソフトウェア品質シンポジウム2017，2017

[2] 加藤正恭，小川秀人，「CRUDマトリクスを用いたソフトウェア設計影響分析手法」，情報処理学会全国大会講演論文集，巻：73rd，号：1，ページ：1.249-1.250，2011

[3] 松尾谷徹，「テストから観た実体のモデリングと実装構造の評価」，JaSST'15Tokai，2015

[4] @tomboyboy ，「NASAの衛星・宇宙機開発で遵守される10のコーディング標準」，2016

[5] 小寺浩司, 柏原一雄,石津和紀, 北野敏明, 佐藤克, 小室睦, 小川清，「確率論及統計論輪講 精度より成果」，14th WOCS2，2016

[6] 柏原一雄，「欠陥混入メカニズムをもとにした ベースソフト調査手法の提案」，第33年度ソフトウェア品質管理研究会，2018

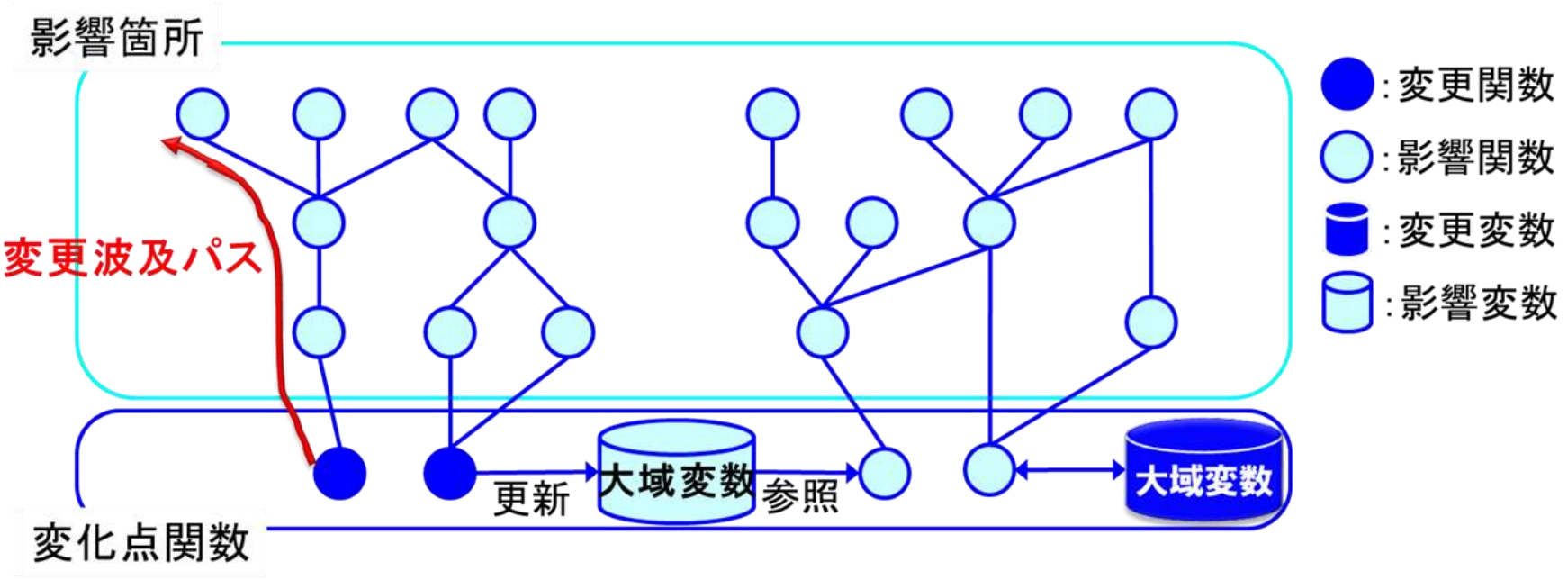
[7] 森崎修司，「ソースコード変更波及解析の 8 カテゴリ - どのくらいご存知ですか？」，http://blogs.itmedia.co.jp/morisaki/2013/03/8---7883.html，2013

付録

■ 影響波及パス分析法（1）

【影響波及パスのイメージ】

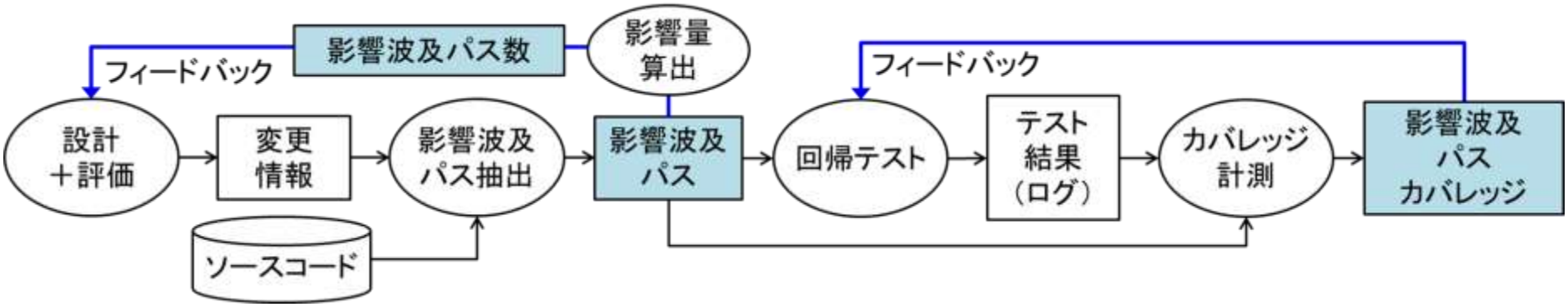
- ✓ 制御とデータの流れによる影響箇所を関数コールパスで表現する
- ✓ 大域変数によって影響が波及する関数も変化点関数と位置付ける



付録

■ 影響波及パス分析法（2）

【影響波及パス分析活用プロセス】



活動	説明
影響波及パス抽出	ソースコードと変更関数・変数の情報を入力として影響波及パスを抽出する
設計＋評価	影響波及パスをもとに算出した影響波及パス数を入力として設計案を選択する
影響関数	影響波及パスとテスト結果を入力としてカバレッジ計測を実施し，影響波及パスカバレッジを算出する
回帰テスト	計測した影響波及パスカバレッジを入力とし，不足しているテスト項目を特定し，回帰テストを再実施する