DENSO
Crafting the Core

# 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

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，変更箇所を漏れなく特定することは重要な課題である

- 変更箇所を確実に特定するために，XDDPをはじめとする手法を適用しているが，変更漏れは防止しきれていない

- ベースソフト調査方法に問題があることを明らかに，解決するための手法を現場で開発した

**ソースコードの変更箇所を漏れなく特定するための**
**ベースソフト調査手法を提案する**

# 2. 現状分析（1）

■ ベースソフトの変更箇所のタイプ

| タイプ | 説明 | 変更箇所特定手法 |
|---|---|---|
| A) **変更仕様**から直接特定可能な箇所 | 変更要求・変更仕様を実現するために，変更が必要となる関数・変数等である. | XDDP |
| B) **ソースコードの変化点**から影響を受ける箇所 | ソースコードの変化点が，制御とデータの流れを介して影響を与える箇所である. | 影響波及パス分析法 |
| C) **設計制約の変化**から影響を受ける箇所 | 変更要求に対応することで，ベースソフトの設計の制約が変化することがある. この制約の変化に対応するために，変更が必要となる箇所である. | 本研究の提案手法 |

※設計制約の例：機能の並行実行不可,機能のキャンセル不可,
バッファへの格納可能データ数,データの参照可能条件 等

**「設計制約の変化から影響を受ける箇所」を特定する手法がない**

# 2. 現状分析（2）

## ■ 変更箇所の特定漏れによる欠陥

- XDDP[2]と影響波及パス分析法[3]を適用していても、防ぎきれない欠陥（変更漏れ）がある

【欠陥の事例】

| 変更仕様概要 | ベースソフトの設計制約 | ベースソフトの設計 | 欠陥内容 |
|---|---|---|---|
| 機能実行中に，他機能の実行が要求されても，実行中機能の処理をキャンセルせず継続 | （キャンセル不可となった機能は）他機能と並列実行不可 | 機能間で共有しているデータあり | 共有データの排他処理漏れ（共有データの個別データ化漏れ） |
| バッファのサイズを縮小 | バッファオーバーフローの可能性がないようにバッファのサイズを確保すること | 同じデータが格納される（格納可能データ数が同一の）複数のバッファあり | バッファオーバーフローのガード処理漏れ（一部箇所のみ） |

**ベースソフトを把握するための，ベースソフトの調査方法に問題あり**

# 2. 現状分析（3）

## ■ ベースソフト調査方法の問題点

- **機能とデータ関係の把握漏れ**
  - 大規模なソースコードに対して，機能とデータの関係を人の能力のみで完全に把握するのは困難であり，誤りも起きる.
  - 短納期での開発に対応するために，調査範囲の絞込みを行う場合，更に把握漏れが発生しやすくなる.

- **設計制約の変化による影響箇所の抽出漏れ**
  - 設計制約の変化による影響箇所は，担当者の知識をもとに抽出している.知識が不足していれば，影響箇所の抽出漏れが発生する.
  - レビューで，抽出された影響箇所に対して問題を検出することはできても，影響箇所が抽出できていないことを問題として検出することは困難である.

**ベースソフトの調査が，人の能力・知識に依存してため問題が起きる**

# 3. 課題提起

## ■ 課題

- 「設計制約の変化から影響を受ける箇所」を特定するための
  ベースソフト調査手法を開発する

## ■ 課題の解決方針：うまくいく手順を再現する

- 変化点から発生し得る欠陥を予測したうえで，ソースコードから，
  欠陥が発生する可能性のある影響箇所を特定する

## ■ 開発する調査手法の要件

- ベースソフト調査の問題を解決できる
  - 機能とデータ関係の把握漏れ
  - 設計制約の変化による影響箇所の抽出漏れ
- 人の能力・知識に依存しない調査が可能である
- 実開発で現実的に実行可能である

**提案手法により，変更箇所特定の成功の再現性を高める**

# 4. 先行研究（1）

## ■ CRUDマトリクスを用いたソフトウェア設計影響分析手法[7]

- ソースコードから関数と変数の依存関係に関連する情報を自動的に抽出して，CRUDマトリクスとして可視化する手法
- グローバル変数を介した処理間の連携動作を把握し，ソフトウェアを変更した場合の影響を特定しやすくすることを課題としている
- CRUDマトリクスを利用し，影響が波及しそうな機能を絞り込み，人手で設計書やソースコードを調査していた作業が効率化できる

| | 変数1 | 変数2 | 変数3 | 変数4 |
|---|---|---|---|---|
| 関数A | U | C | | |
| 関数B | | RU | | R |
| 関数C | U | D | R | |
| 関数D | U | | R | C |
| 関数E | U | | | D |

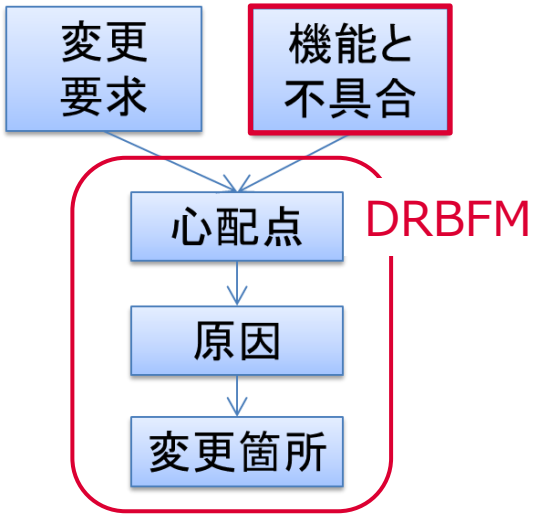
C：作成（Create）
R：参照（Read）
U：更新（Update）
D：削除（Delete）

**CRUDマトリクスを参考に，**
**データと機能の関係を把握するための仕組みを開発**

# 4. 先行研究（2）

## ■ XDDPへのDRBFMの導入[5]

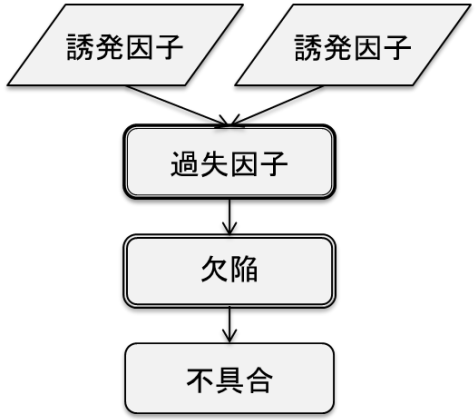
- 変更要求から心配点（＝発生し得る欠陥）を漏れなく抽出することを課題としている
- 機能と過去の不具合情報の関係を入力としてDRBFMを実施し，心配点と原因が抽出できる




## ■ ソフトウェア欠陥予測アルゴリズム[6]

- 「失敗の知識を活用するには失敗のメカニズムを示す要因と結果の要素が必要」という考え方をもとにした手法
- 欠陥混入メカニズムの表現・蓄積・利用することで，同一条件下で発生する欠陥を予測できる

【欠陥混入メカニズムの表現】



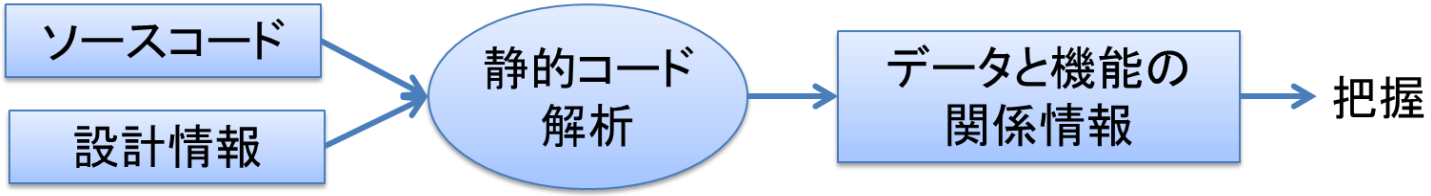

**変更要求から，発生し得る欠陥→原因→変更箇所を特定**
**そのために，欠陥情報を蓄積・利用する仕組みを構築**

# 5. 解決策の提案（1）

■ 問題の解決方針：
機能とデータ関係の把握漏れ

- 「機能とデータの関連」は，CRUDマトリクスを拡張・変更し表現する
  - 関数と変数の関係のみではなく，関数コールツリーも表現し，公開IFと変数の関係を把握可能とする
  - データアクセスの種類は，変更箇所特定の目的で必要となるW：書き込み（Write）とR：読み込み（Read）の2種類のみとする

- 静的コード解析技術を活用することで，担当者の能力を補う

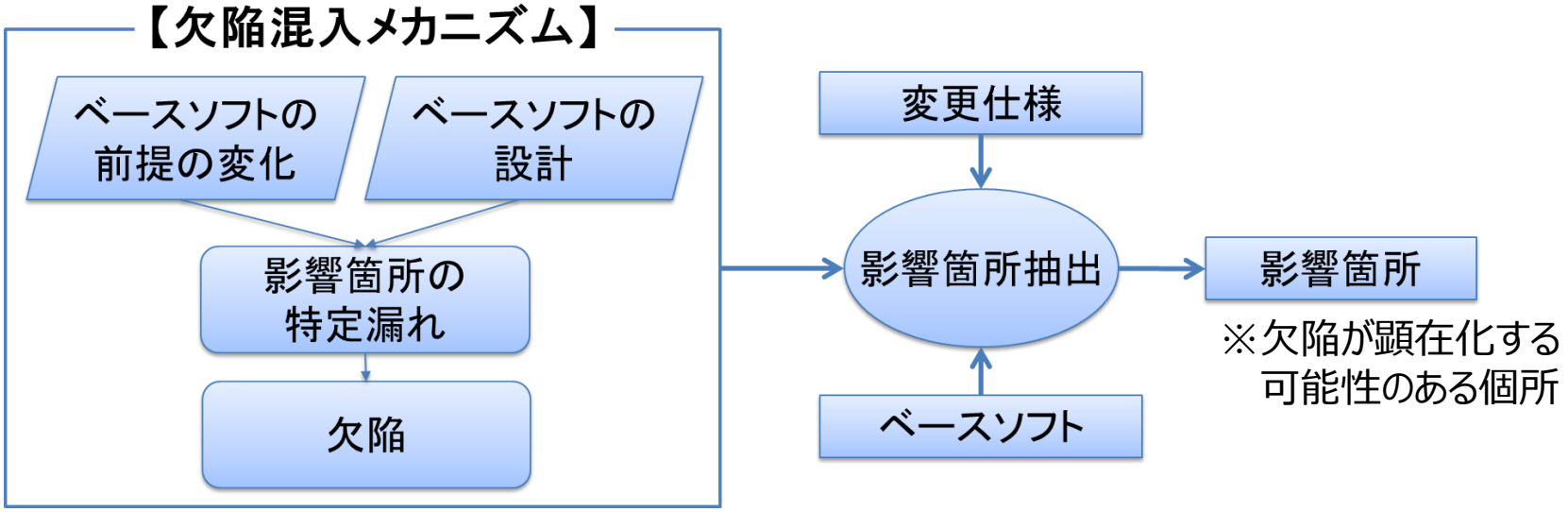



**静的コード解析を活用し，機能とデータ関係の把握**

# 5. 解決策の提案（2）

■ 問題の解決方針：
設計制約の変化による影響箇所の抽出漏れ

- 「設計制約の変化」と「ベースソフトの設計」を要因として発生する「欠陥」の関係を示した情報（＝**欠陥混入メカニズム**）を蓄積する
- 欠陥混入メカニズムの知識を利用し，担当者の知識を補い，変更箇所が問題なく特定できているときの手順を再現可能とする

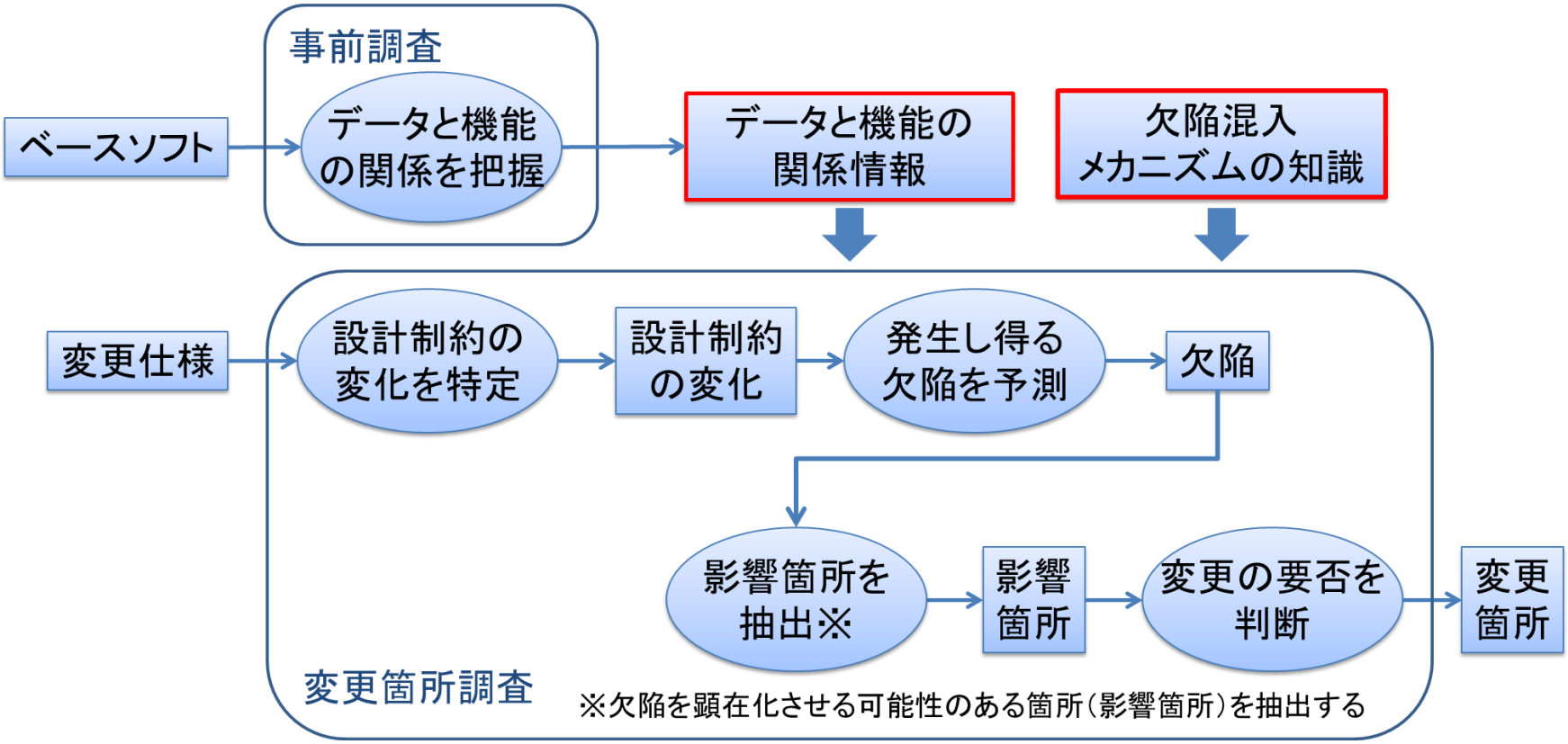



**欠陥混入メカニズムの知識を蓄積し，影響箇所抽出に利用**

# 5. 解決策の提案（3）

■ 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

- 「データと機能の関係情報」と「欠陥混入メカニズムの知識」を利用するベースソフト調査手法



# 5. 解決策の提案（4）

## ■ データと機能の関係情報の表現方法

- 「データと機能の関係」を表現した成果物を，**CTDマトリクス（Call tree with data マトリクス）**と呼ぶ
- CTDマトリクスは，関数コールツリーと外部変数を2軸にもつ

| 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）

変数にアクセスする関数（機能）を特定可能

各機能からアクセスする変数を特定可能

複数機能で共有している変数を特定可能

# 5. 解決策の提案（5）

## ■ 欠陥混入メカニズムの知識の表現方法

- 欠陥混入メカニズムの知識を蓄積した成果物を，**DIM辞書（Defect injection mechanism 辞書）**と呼ぶ
- 調査対象ソフトが変わっても利用できるように，抽象化して表現する

【欠陥混入メカニズムの知識に含める要素】

| 要素 | 説明 |
|---|---|
| 設計制約の変化 | ベースソフトの設計制約の変化を示す．欠陥混入メカニズムの知識を検索するためのキーとなる情報である． |
| 欠陥 | 設計制約の変化により，引き起こされる可能性のある欠陥を示す． |
| ベースソフトの設計 | 欠陥の要因となるベースソフトの設計を示す．予想した欠陥が発生し得る箇所（影響箇所）を絞り込むために利用する知識である． |
| 設計の表現技法 | ベースソフトの設計を把握するために利用する表現技法を示す． |
| ベースソフトの調査手順 | 予想した欠陥が発生し得る箇所を絞り込み，ソースコードの変更要否を判断する手順を示す． |

# 5. 解決策の提案（6）

## ■ DIM辞書

| No | 設計制約の変化 | 欠陥 | ベースソフトの設計 |
|---|---|---|---|
| 1 | Before)<br>機能が並列実行不可能<br>After)<br>機能が並列実行可能 | 共有データの排他処理漏れ | ・機能間で共有しているデータ(外部変数)あり |
| 2 | Before)<br>バッファサイズの制約あり<br>After)<br>バッファサイズの制約なし<br>(バッファオーバランが発生する可能性のあるサイズも許容) | バッファオーバーランのガード処理漏れ | ・同じ種類のデータが格納されるバッファ(配列)が複数あり<br>・バッファ(配列)が保持されているユニットとは別ユニットから書き込みがされる箇所あり |
| 3 | Before)<br>処理のキャンセル不可能<br>After)<br>処理のキャンセル可能 | デッドロックの対策処理漏れ | ・処理実行中の状態を複数のレイヤ(ユニット)でそれぞれ管理 |
| 4 | Before)<br>バッファのデータの歯抜けなし<br>After)<br>バッファのデータの歯抜けあり | データ取得時の無効値判定処理漏れ／有効値範囲判定処理誤り | ・バッファのデータの並べ替え処理なし<br>・バッファ(配列)のデータ読み込み箇所が複数 |

「設計制約の変化」は、“当てはまるか？”の判断がしやすいように表現する

「ベースソフトの設計」欄には，影響箇所の特定時に注目すべき点を示す

# 6. 解決策の評価（1）

## ■ 評価方法

- DIM辞書を準備し，CTDマトリクスの生成ツールを開発したうえで，次の2つの観点で，提案手法の評価を実施
  - a.「設計制約の変化による影響箇所」の抽出が可能か？
  - b. 過去に発生した変更箇所の特定漏れを防止することが可能か？

【評価観点と評価方法】

| ID | 評価観点 | 評価方法 |
|---|---|---|
| a | 「設計制約の変化による影響箇所」の抽出が可能か？ | 開発経験の異なる複数の技術者（対象者：5人）に対して，アンケート方式で確認を実施.<br>DIM辞書を入力に，「発生し得る欠陥の予測」と「欠陥を顕在化させる設計の特定」が可能かを質問. |
| b | 過去に発生した変更箇所の特定漏れを防止することが可能か？ | ベースソフト調査の再実施を行い，変更箇所が特定できるかを確認.<br>過去に変更箇所の特定漏れが発生した2件の事例を対象とした. |

# 6. 解決策の評価（2）

## ■ 評価結果

a. 「設計制約の変化による影響箇所」の抽出が可能か？

「発生し得る欠陥の予測」と「欠陥を顕在化させる設計の特定」が可能であると回答した技術者は4名，不可能と回答した技術者は1名．
不可能と回答した技術者は，ソフトウェア開発経験１年以下．

b. 過去に発生した変更箇所の特定漏れを防止することが可能か？

ベースソフト調査を再実施した結果，2件の事例共に，
「設計制約の変化から影響を受ける箇所」を特定できた．

## ■ 結果の考察

- 提案手法により，担当者の能力・知識を補い，設計制約の変化による影響箇所の特定が可能となることが確認できた
- 提案手法による効果を得るには，DIM辞書の内容を理解できていることが条件である
- ベースソフトの設計制約が明文化されていれば，変化も捉えやすい

# 7. まとめ

## ■ 研究の成果

- 変更箇所のタイプを分類し，「設計制約の変化から影響を受ける箇所」の特定漏れを防止する手法が必要であることを明らかにした
- 「データと機能の関係情報（ CTDマトリクス）」と「欠陥混入メカニズムの知識（ DIM辞書）」を利用したベースソフト調査手法を開発した
- 提案手法を適用することで，過去に発生した変更箇所の特定漏れを防止できることを確認した

## ■ 今後の進め方

- 欠陥混入メカニズムの知識を蓄積するプロセスを考案する
- DIM辞書を拡充したうえで，手法の評価・改善を実施する
- 提案手法を他プロジェクトに展開するために，「DIM辞書」を利用したベースソフト調査プロセスを定義し，教育の実施を検討する

# おわりに

## ■ 実践コースでの気づき

- 大きな成果を生み出すために，研究成果を積み上げていく．
  **肯定眼**をもって，過去の自分達の成果を見るべし（捨てるべからず）．
- 同様の課題の解決に取り組んでいる先人は多数いる．
  **肯定眼**をもって，先人の論文・書籍等から学ぶ（盗む）べし．
- “うまくいく場合”と“うまくいかない場合”を比較することで，問題点が見える．
  **肯定眼**をもって，“うまくいく場合”に着目すべし．

## ■ 謝辞


- 研究会活動において，飯泉様・足立様・清水様から，
  本研究に対する有益な助言　と
  技術者としての姿勢に対する厳しく・温かいご指導　をいただきました．
  ここに感謝の意を表します．

DENSO

Crafting the Core

# 参考文献


[1]清水吉男，「派生開発における 母体に由来するバグとその対応」，JaSST'09 Tokyo，2009

[2]清水吉男，「「派生開発」を成功させるプロセス改善の技術と極意」，技術評論社，2007

[3]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

[4] SQiP研究会第６分科会（A グループ），「変更の影響範囲を特定するための「標準調査プロセス」の提案」，ソフトウェア品質管理研究会 第30年度分科会成果報告，2015

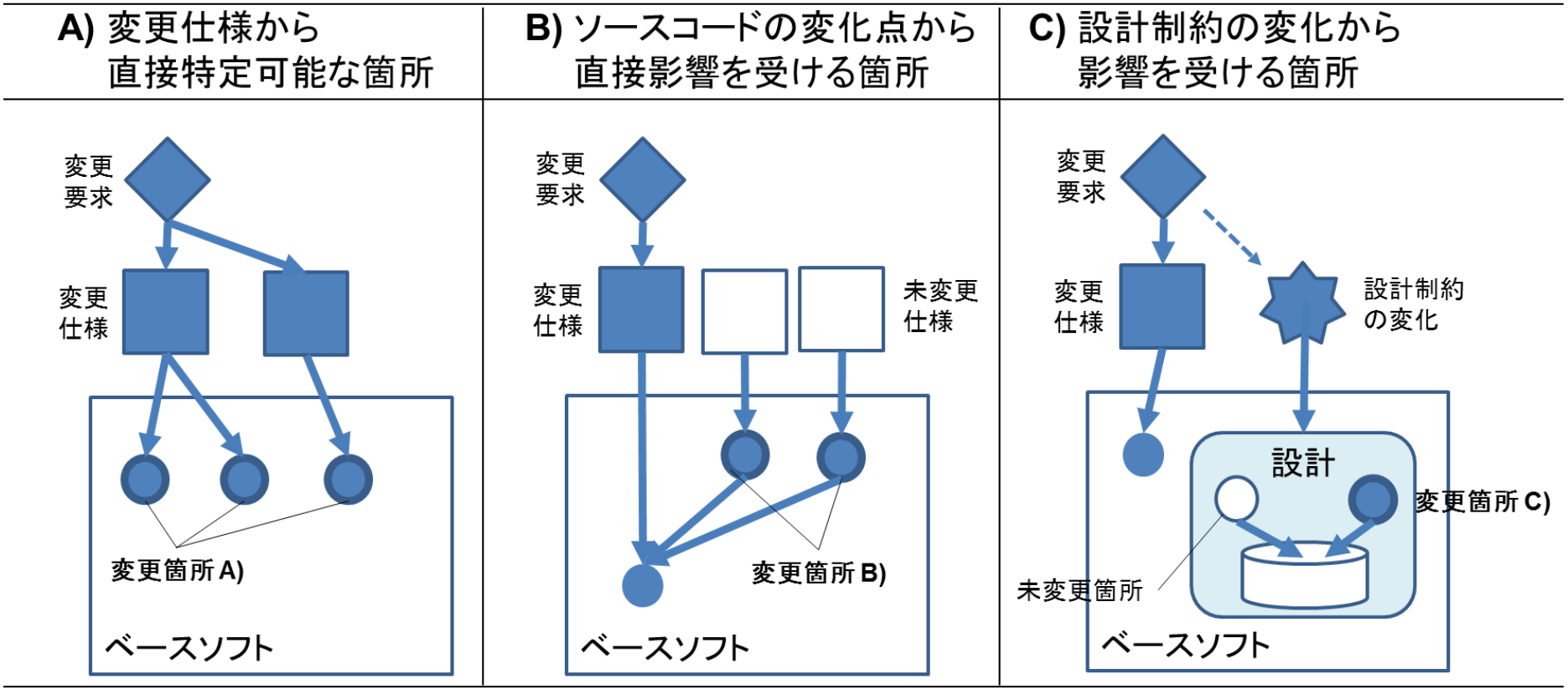
[5]安田隆司，「変更要求仕様書を活用した未然防止方法の提案-XDDPへのDRBFM導入とその効果-」，ソフトウェア品質シンポジウム2011，2011

[6]2014年度 SQiP研究会第7分科会，「ソフトウェア欠陥予測アルゴリズム-欠陥混入メカニズムのモデリング手法を利用した欠陥予測方法の提案-」，ソフトウェア品質シンポジウム2015，2015

[7]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

# 付録

## ■ ベースソフトの変更箇所3タイプのイメージ

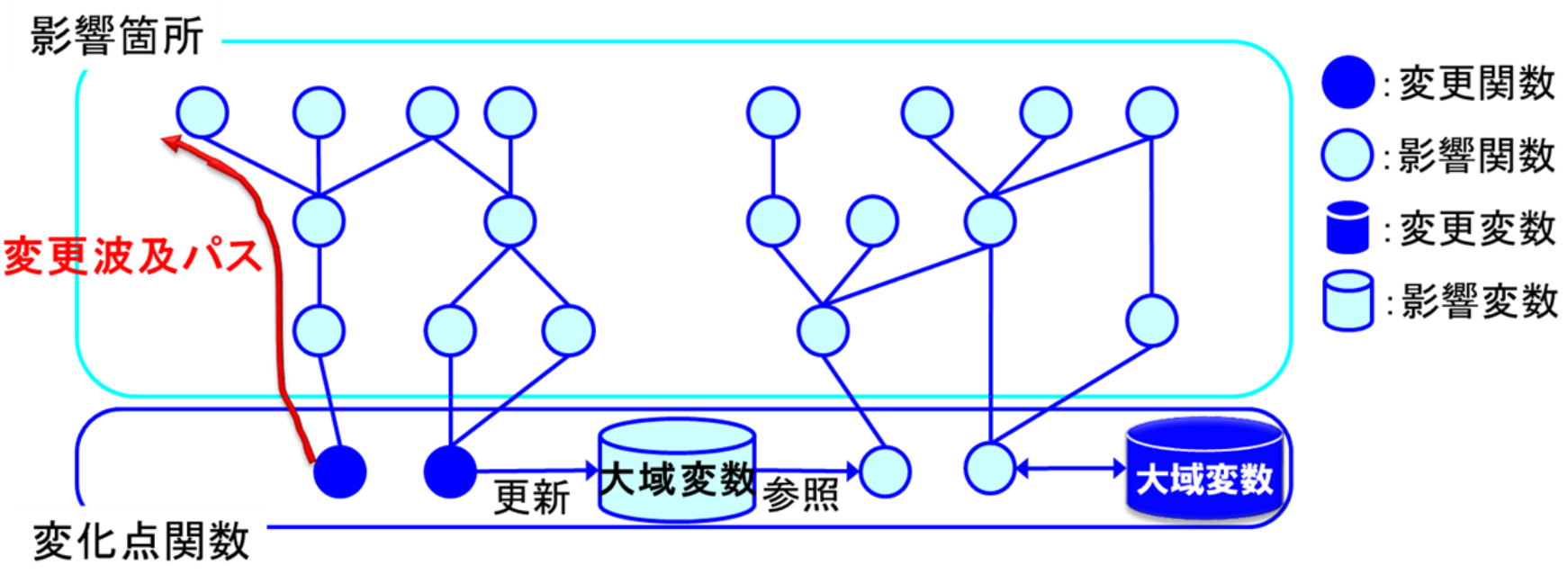


# 付録

## ■ 影響波及パス分析法（1）

【影響波及パスのイメージ】

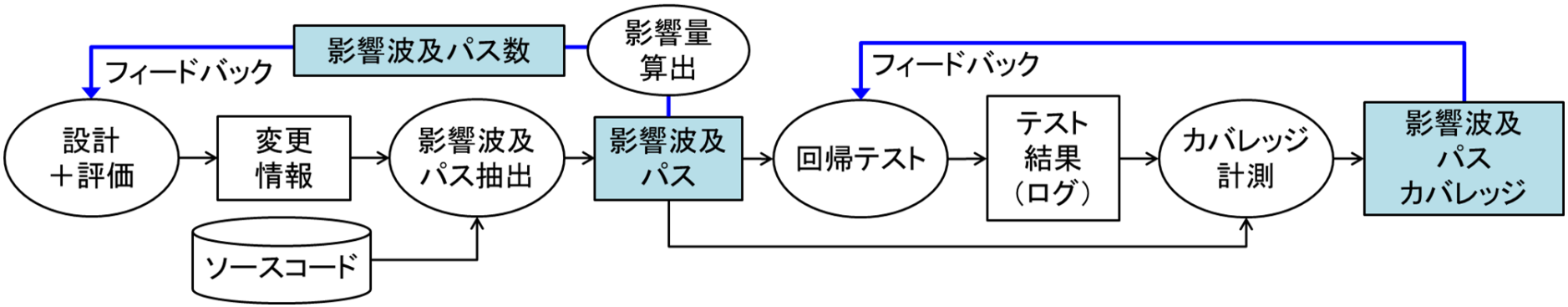
- ✓ 制御とデータの流れによる影響箇所を関数コールパスで表現する
- ✓ 大域変数によって影響が波及する関数も変化点関数と位置付ける



# 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 |

# 付録

## ■ 変更箇所を特定しているときのベースソフト調査手順

| No | 手順 | 入力 | 出力 |
|---|---|---|---|
| 1 | 変更要求をもとに，変化する設計制約を特定する | ・変更要求 | ・設計制約の変化 |
| 2 | 設計制約が変化する場合，その変化に伴い発生の可能性のある欠陥を予測する | ・設計制約の変化 | ・発生し得る欠陥 |
| 3 | 予測した欠陥は，ベースソフトの設計がどのような場合に混入するか特定する | ・発生し得る欠陥 | ・欠陥混入に繋がる設計 |
| 4 | ベースソフトの設計を把握するために利用する設計の表現技法（例：シーケンス，DFD，フローチャート等）を決める | ・欠陥混入に繋がる設計 | ・設計把握のための表現技法 |
| 5 | 4で決めた技法でベースソフトの設計を表現する | ・設計把握のための表現技法 | ・ベースソフトの設計 |
| 6 | 5で作成した成果物を活用し，ベースソフトの設計を把握し，予想した欠陥が発生し得る影響箇所を絞り込む | ・ベースソフトの設計<br>・発生し得る欠陥 | ・影響箇所 |
| 7 | 6で絞り込んだ影響箇所に対して，ソースコードの変更の要否を判断する | ・影響箇所<br>・発生し得る欠陥 | ・変更箇所 |

# 付録

## ■ CTDマトリクス作成に必要な設計情報

- 既存の静的コード解析ツールでは，関数ポインタによる関数呼び出しやポインタを利用してデータにアクセスしている箇所の特定が困難である．これに対して，ツールで解析が困難なことは手動で補う方針で，現実的に実行可能な仕組みを構築した．
- CTDマトリクスは，ソースコードに加えて，以下に示した設計情報を入力として静的コード解析により作成する． 通常の関数コール関係はソースコードから自動解析する．

【関数呼び出し関係リスト】

| 呼出し元関数 | 呼出し先関数 |
|---|---|
| FuncA | FuncB |
| FuncA | FuncC |
| FuncB | FuncC |
| FuncD | FuncE |

【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・・・データの更新処理あり
※R・・・データの参照処理あり

# 付録

## ■ 欠陥混入メカニズム辞書

| No | 設計制約の変化 | 欠陥 | ベースソフトの設計 | 設計の表現技法 | ベースソフトの調査手順 |
|---|---|---|---|---|---|
| 1 | Before）<br>機能が並列実行不可能<br>After）<br>機能が並列実行可能 | 共有データの排他処理漏れ | ・機能間で共有しているデータ（外部変数）あり | – | 1. 並行実行可能となった機能で、共有している外部変数を特定する<br>2. 1で特定した外部変数に対して、排他処理の追加要否を判断する |
| 2 | Before）<br>バッファサイズの制約あり<br>After）<br>バッファサイズの制約なし<br>（バッファオーバランが発生する可能性のあるサイズも許容） | バッファオーバーランのガード処理漏れ | ・同じ種類のデータが格納されるバッファ（配列）が複数あり<br>・バッファ（配列）が保持されているユニットとは別ユニットから書き込みがされる箇所あり | ・DFD | 1. バッファオーバーランの可能性が発生したバッファを特定する<br>2. 1で特定したバッファを起点としてDFDを作成する<br>3. 2で作成したDFDから関連するバッファを特定する<br>4. 3で特定したバッファに対して、オーバーランのガード処理の追加要否を判断する |
| 3 | Before）<br>処理のキャンセル不可能<br>After）<br>処理のキャンセル可能 | デッドロックの対策処理漏れ | ・処理実行中の状態を複数のレイヤ（ユニット）でそれぞれ管理 | ・シーケンス<br>・状態遷移表 | 1. 処理中を示す状態を特定<br>2. 1で特定した状態について状態遷移表を作成する<br>3. 正常処理についてシーケンス図を作成する<br>4. 3で作成したシーケンス図からキャンセルが発生する可能性のあるタイミングを特定する<br>5. 2で作成した状態遷移表に対して、キャンセルトリガが増えることで、デッドロックが発生するケースがないか判 |
| 4 | Before）<br>処理が異常終了することなし<br>After）<br>処理が異常終了することあり | デッドロックの対策処理漏れ | ・処理実行中の状態を複数のレイヤ（ユニット）でそれぞれ管理 | ・シーケンス<br>・状態遷移表 | ↑ |
| 5 | Before）<br>バッファのデータの歯抜けなし<br>After）<br>バッファのデータの歯抜けあり | データ取得時の無効値判定処理漏れ／有効値範囲判定処理誤り | ・バッファのデータの並べ替え処理なし<br>・バッファ（配列）のデータ読み込み箇所が複数 | – | 1. データが歯抜けとなりうるバッファを特定する<br>2. 1で特定したバッファのデータ取得箇所を特定する<br>3. 2で特定した箇所に対して、無効値判定処理の追加要否を判断する |
| 6 | Before）<br>バッファのデータの並び順に関する制約あり<br>After）<br>バッファのデータの並び順に関する制約なし | データ検索処理誤り | ・バッファのデータの並べ替え処理なし<br>・バッファ（配列）のデータ読み込み箇所が複数 | – | 1. 並び順に関する制約がなくなるバッファを特定する<br>2. 1で特定したバッファのデータ取得箇所を特定する<br>3. 2で特定した箇所に対して、データ検索処理の変更要否を判断する |
| 7 | データの有効区間が変化 | データ取得時の無効値判定処理漏れ | ・データの参照箇所が複数 | タイミングチャート | 1. 有効区間が変化したデータを特定する<br>2. 必要に応じて1.で特定したデータ更新タイミングを示すタイミングチャートを作成する<br>3. 1で特定したデータの参照箇所を特定する<br>4. 3で特定した箇所に対して、タイミングチャート等を活用し、無効値判定処理の追加要否を判断する |