# バグ混入コミットによる過去制約違反の実証研究

○近藤 偉成[1]，近藤 将成[2]，ヌリ オリビエ[1]，亀井 靖高[2]，肥後 芳樹[1]

1: 大阪大学

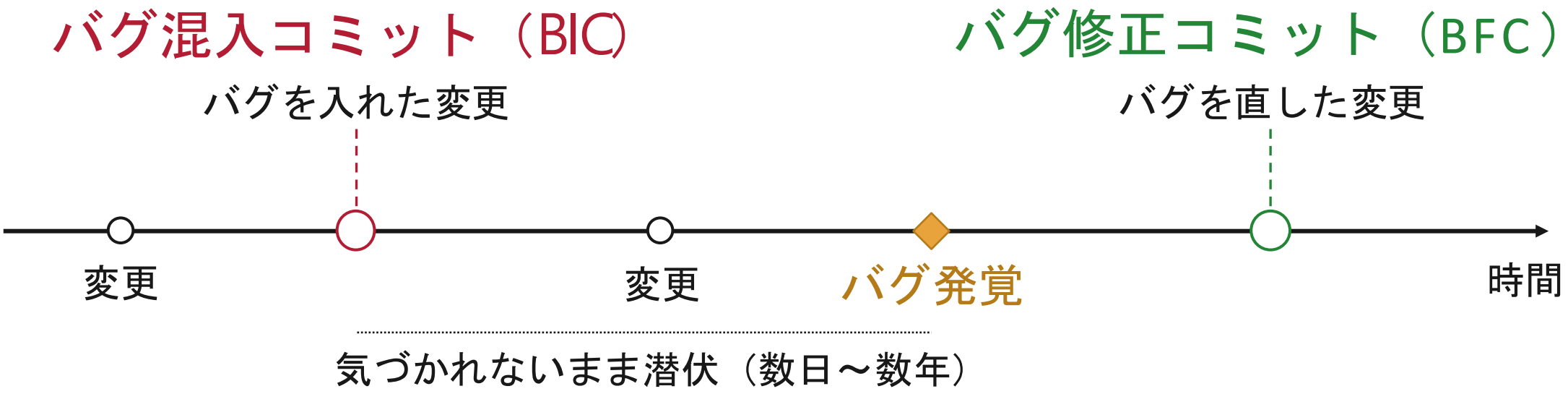
2: 九州大学

# 背景：バグの予防・修正には発生要因の分析が重要

バグの発生要因を理解するために**バグ混入コミットが分析**される

課題：バグの修正には多くの時間がかかる

解決策：バグの発生要因の理解により予防・検出・修正を改善



# 課題：バグ混入コミットの変更内容だけでは原因分析困難

バグ混入コミットの変更差分やメタ情報など**変更内容**を分析[1]

- 何行変更したか
- どのようなコードを追加・削除したか
- どのファイルを変更したか
- 誰が変更したか
- いつ変更したか

→ 変更内容だけではバグの原因が説明できない場合がある

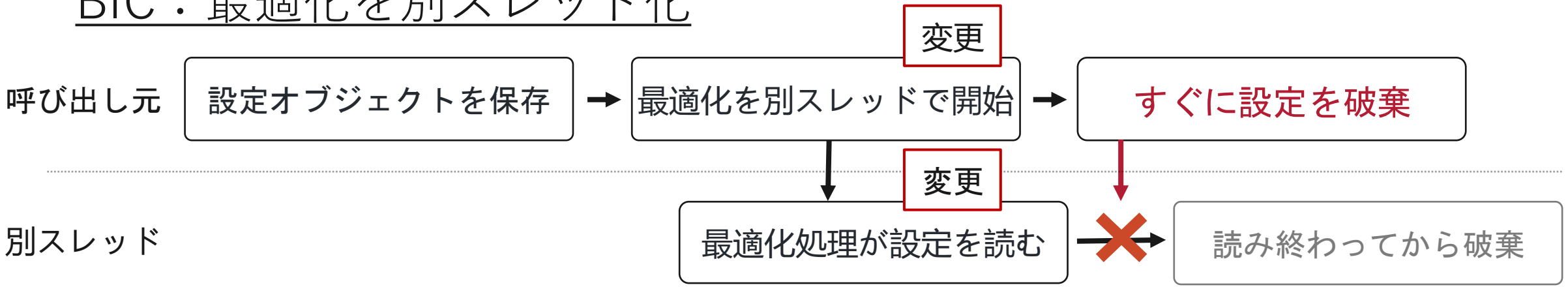
[1] Śliwerski et al., “When Do Changes Induce Fixes?”, MSR, 2005

# 変更内容だけではバグの原因を分析できない例

変更内容の別スレッド化は正しいが**別コードの制約を違反**する

## BIC 以前：順番に実行


## BIC：最適化を別スレッド化



# 本研究の着眼：過去制約違反

バグ混入コミットの変更内容ではなく**違反された過去制約**に着目

## 過去制約違反

**制約**：ソフトウェアが満たすべき振る舞い・関係・前提

2条件を満たす**制約**を破ったBIC

**条件 1**　制約が BIC より前に確立されている（時系列）

**条件 2**　違反は diff 単体には現れない（位置）

# 研究設問

違反された過去制約の**特徴**，**位置**，**明示度**を明らかにする

**RQ1** 過去に確立された制約を違反する BIC はどの程度存在し，
どのような制約が違反されているか？

**RQ2** 違反された制約は，どこで・いつ・誰によって確立されたか？

**RQ3** 違反された制約は，バグ混入時点でどのように明示されていたか？

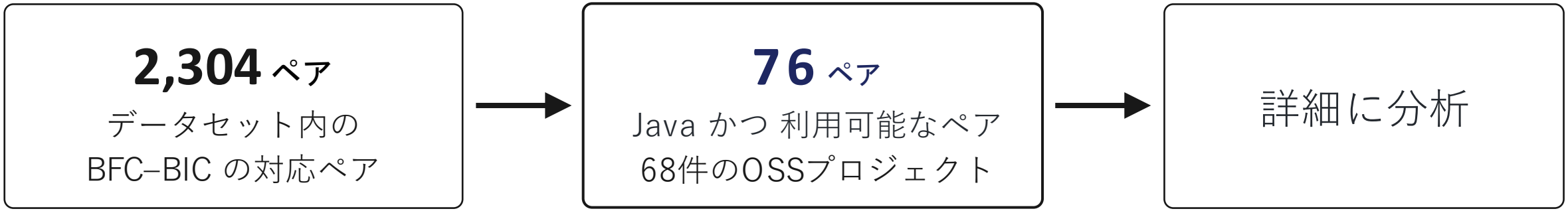
# データセット

開発者がラベル付けした信頼性の高いデータセットを分析する

開発者情報付きデータセット[2]

　開発者がバグ修正時のメッセージでバグ混入コミットを明示

→ BFC と BIC の対応関係の信頼性が高い

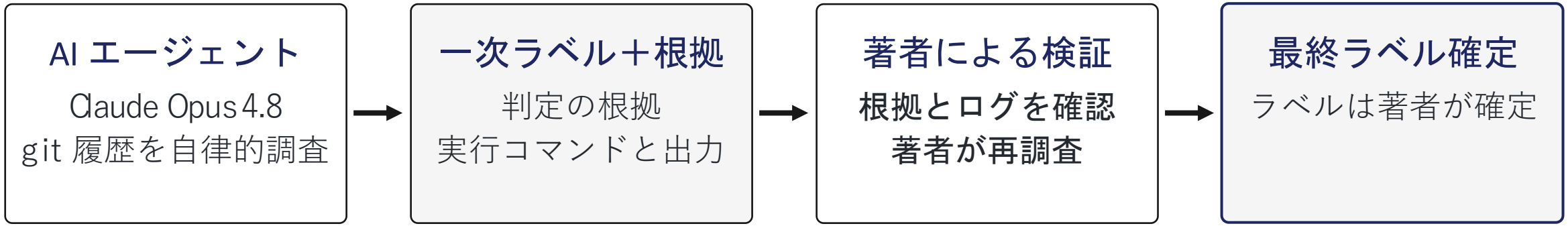



[2] Rosa et al., “A Comprehensive Evaluation of SZZ Variants through a Developer-Informed Oracle,” JSS, 2023.

# 判定方法

AI エージェントと著者により過去制約違反を判定する

- **コーディングルールを与えて AI エージェントを実行**
- **AI エージェントの実行コマンドと出力を記録**
- 著者がログを確認して妥当性を調査
- 著者が再調査を実施して最終ラベルを確定



# RQ1：過去制約違反の頻度

76件のうち**59.2%が過去制約違反**であった

**判定ラベルの内訳（76件）**


- 過去制約違反　45件（59.2%）　BIC以前からの制約を破ったBIC
- 局所的バグ　22件（28.9%）　BICの差分内で説明できるBIC
- BIC誤り　8件（10.5%）　BICの以前 or 以後でバグ混入
- 判定困難　1件（1.3%）　制約が事前に存在したかを示す証拠が不足

# RQ1：違反された制約の種類

データや実行順序に関する**意味的な制約が86.7%**が破られていた

## データ制約

**22** 件　48.9%

データが満たすべき性質

- 値の範囲
- Null
- 状態など

## 実行文脈上の制約

**17** 件　37.8%

実行のされ方に関する前提

- 呼び出し順序
- 並行実行
- ライフサイクルなど

## コード規約

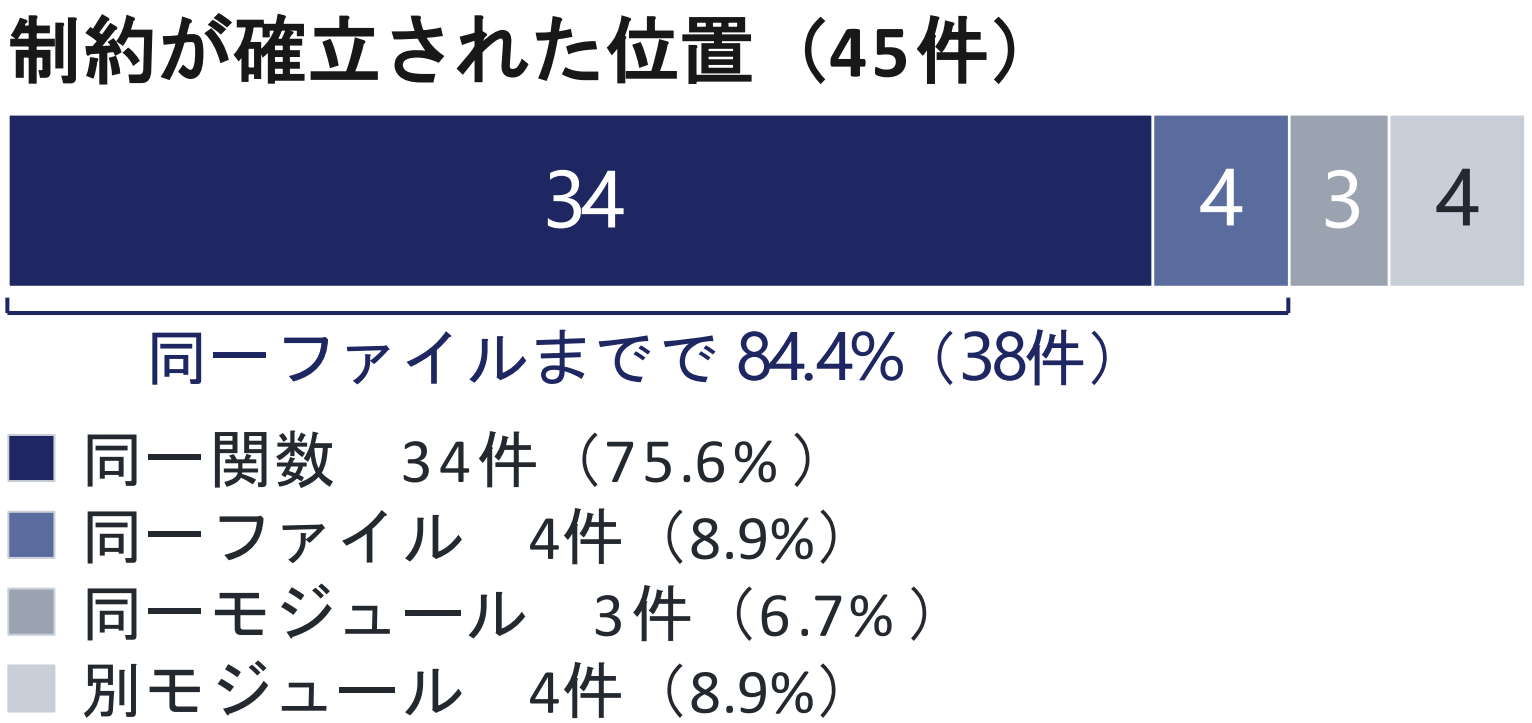
**6** 件　13.3%

既存コードとの一貫性

- API の使い方
- リソース管理
- フォーマットなど

# RQ2：制約が確立された位置

制約の**75.6%は同じ関数**で確立されていた



# RQ2：制約を誰が・いつ破ったか

制約が破られるまでの**期間は広く分布**し，**本人でさえ違反**していた

## 制約が破られるまでの期間


## 誰が破ったか

**42 %**

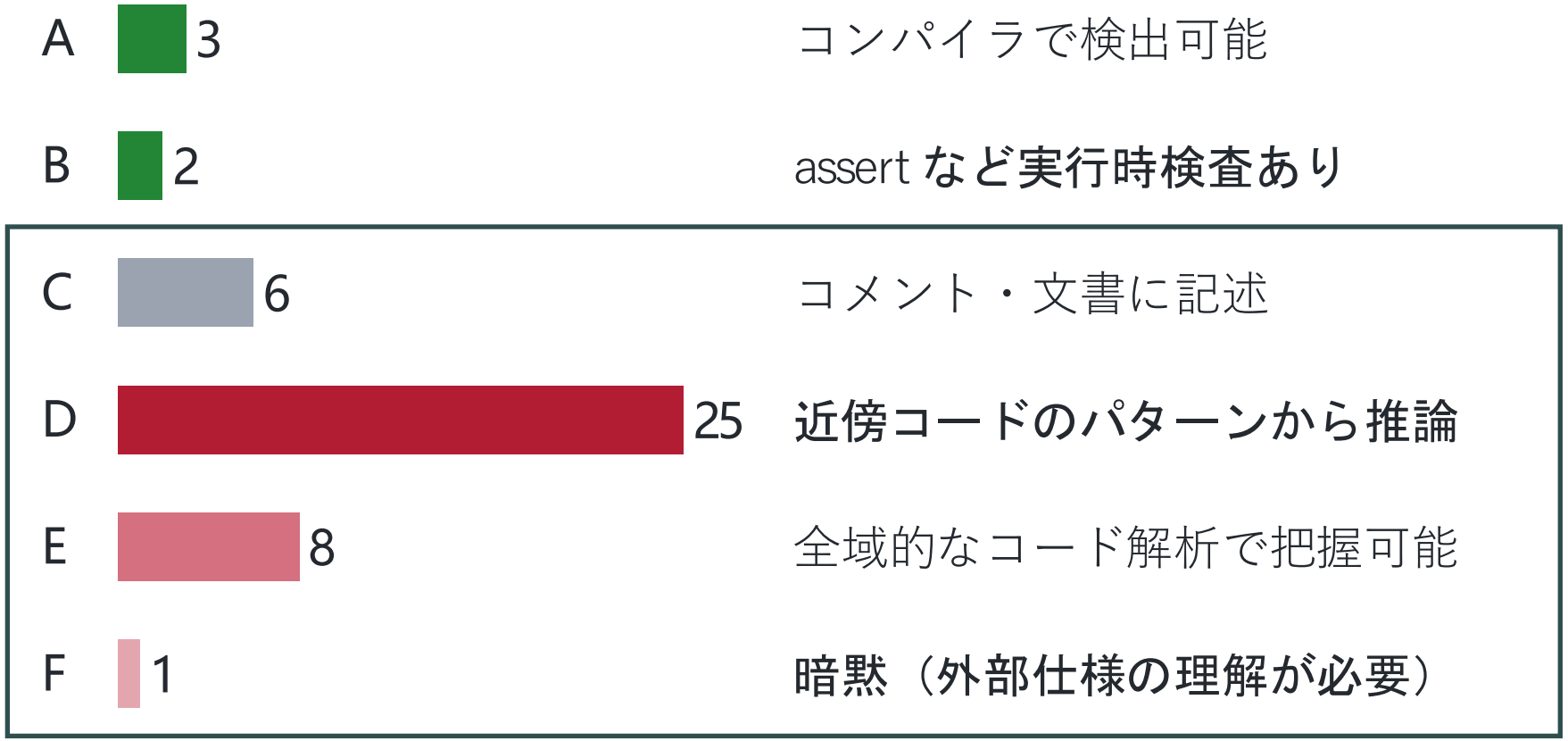
**制約を確立した本人**

(19/45件)

# RQ3：制約はどのように明示されていたか

制約の**88.9%はコンパイラやテストで検出できない**状態だった

**明示形式（45件）**




88.9%（40/45）
コンパイラやテストで検出不可

# 示唆：バグ混入時点と制約確立時点での予防策

**制約を抽出して警告**と**制約のテストを作成**する予防策が考えられる

| | |
|---|---|
| RQ1 | バグ混入コミットの59.2%が過去制約違反 |
| RQ2 | 75.6%の制約が同じ関数で確立 → **変更箇所の近くで確立** |
| RQ3 | 88.9%の制約はコンパイラやテストで不可 → **テスト不足** |

バグ混入時点：変更箇所の近傍コードから**制約を抽出して警告**

制約確立時点：後続の変更が満たすべき**制約のテストを作成**

## 課題：バグ混入コミットの変更内容だけでは原因分析困難

バグ混入コミットの変更差分やメタ情報など**変更内容**を分析[1]

- 何行変更したか
- どのようなコードを追加・削除したか
- どのファイルを変更したか
- 誰が変更したか
- いつ変更したか

→ 変更内容だけではバグの原因が説明できない場合がある

[1] Śliwerski et al., “When Do Changes Induce Fixes?”, MSR, 2005



## 本研究の着眼：過去制約違反

バグ混入コミットの変更内容ではなく**違反された過去制約**に着目

### 過去制約違反

**制約**：ソフトウェアが満たすべき振る舞い・関係・前提
2条件を満たす**制約**を破ったBIC

**条件 1** 制約が BIC より前に確立されている（時系列）

**条件 2** 違反は diff 単体には現れない（位置）



## 研究設問

違反された過去制約の**特徴**，**位置**，**明示度**を明らかにする

**RQ1** **過去に確立された制約を違反する BIC はどの程度存在し，どのような制約が違反されているか？**

**RQ2** **違反された制約は，どこで・いつ・誰によって確立されたか？**

**RQ3** **違反された制約は，バグ混入時点でどのように明示されていたか？**



## 示唆：バグ混入時点と制約確立時点での予防策

**制約を抽出して警告**と**制約のテストを作成**する予防策が考えられる

**RQ1** バグ混入コミットの59.2%が過去制約違反

**RQ2** 75.6%の制約が同じ関数で確立 → **変更箇所の近くで確立**

**RQ3** 88.9%の制約はコンパイラやテストで不可 → **テスト不足**

バグ混入時点：変更箇所の近傍コードから**制約を抽出して警告**

制約確立時点：後続の変更が満たすべき**制約のテストを作成**