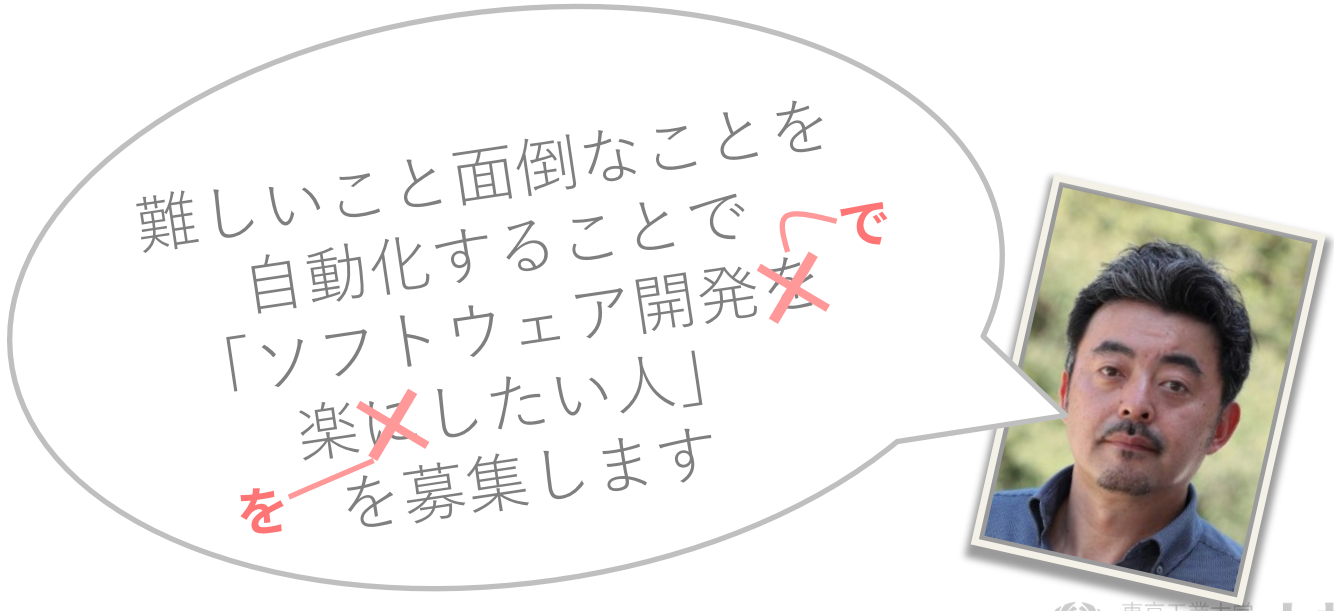
Call for Members for AY2024

小林研の研究内容の紹介



小林研の研究領域

- **ソフトウェア工学**
  - ソフトウェア保守・進化支援 [2007~]
    - **「既にある(難解な)ソフトウェアの変更・修正」**
    - プログラム理解支援，デバッグ技法，トレーサビリティ管理
  - ソフトウェア開発支援環境 [2007～]
    - **「面倒なこと・難しいことの自動化」**
    - プログラム解析ツール，デバッガ，DevOps, Infra. as Code
  - ソフトウェア分析・設計技法 [1997~]
    - **「コードより抽象度の高い表現とは？」**
    - 設計方法論，アーキテクチャ，デザインパターン
- 過去のデータに基づく**帰納的アプローチ**
  - リポジトリマイニング，機械学習に基づく手法

小林研の大きな研究目標

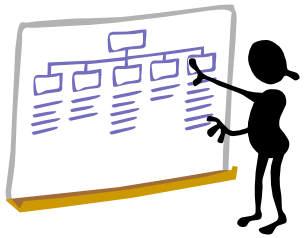
- ソフトウェア開発における属人性の軽減
  - 「できる人」を増やす
  - 熟練者の**知識・経験**を可能な限り「形式知」へ

なぜ「知識や経験」に着目するのか？

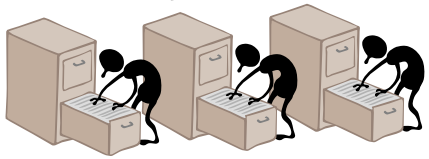
- ソフトウェア開発は高度な知的創造活動
  - 実現している機能 ⇔ ソースコード の
    変換作業は現在でも説明困難な活動
  - 開発者の理解・行動プロセスの補助が必要
- 2方向の支援アプローチ
  - 演繹的：プログラムの意味を解析し補助
    - 部分的なプログラムの意味を用いて理解・解析を支援
      - 命令単位での意味 → 関数単位での意味 → …
    - 支援の確度は高いが，適用範囲が限定的になりがち
  - 帰納的：事例(プログラム・活動)を解析し補助
    - 多数の事例から意味/意図を導出 : ML based AI的アプローチ
    - 適用範囲が広いが，支援の確度は事例数と共通性に依存

こちらを採用

研究アプローチ (基本的な考え方)



モデルを定義



高コスト

モデルに基づき
暗黙知を記述
(形式知化)

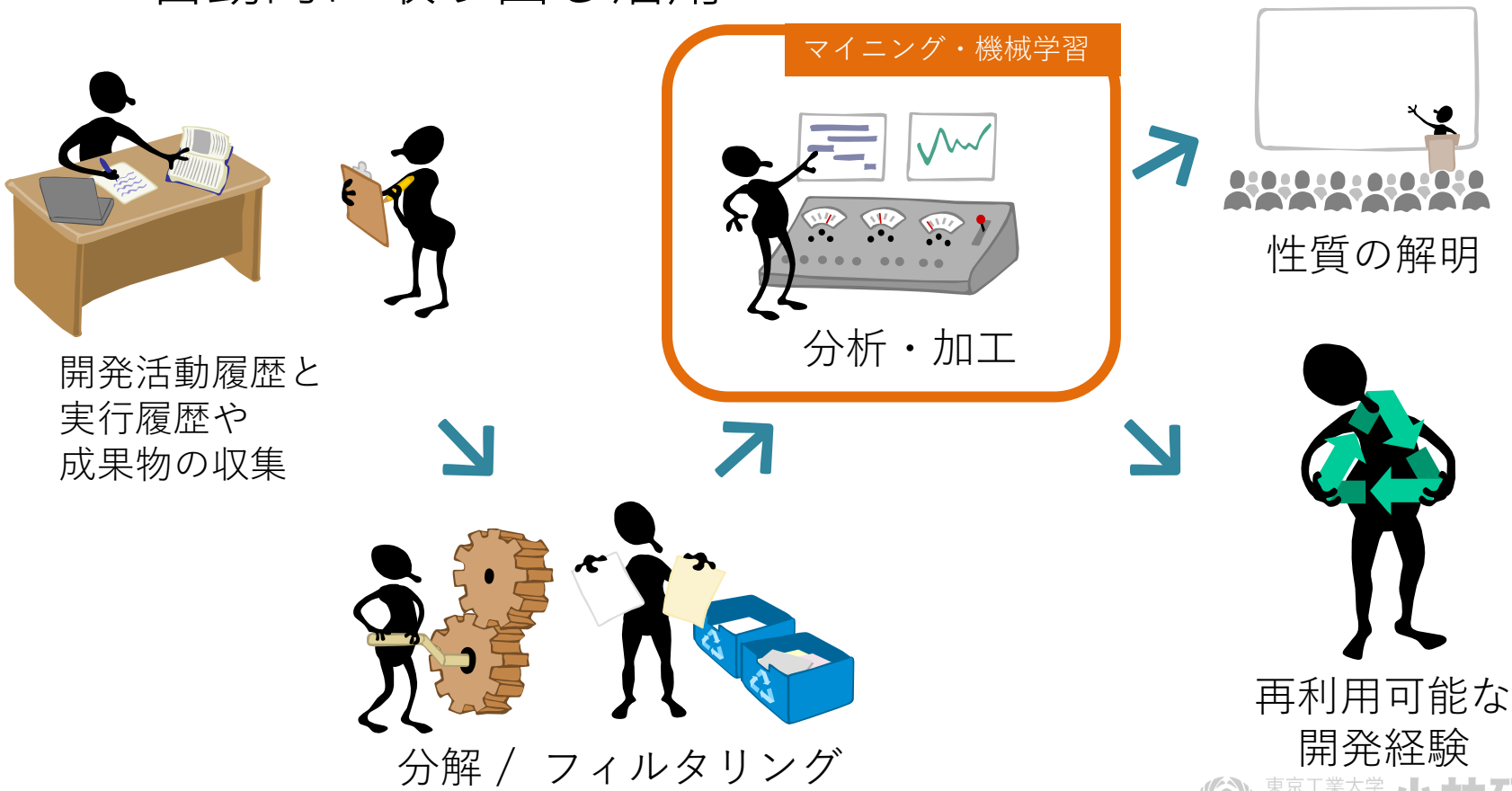
(リポジトリに格納した)
形式知に基づくツール支援

- 作業の自動化
- 作業結果の検証
- 作業内容の推薦

自動運転でいうと:
- 完全自動（手放し）
- ハンドルアシスト
- ブレーキアシスト
- ナビゲーション

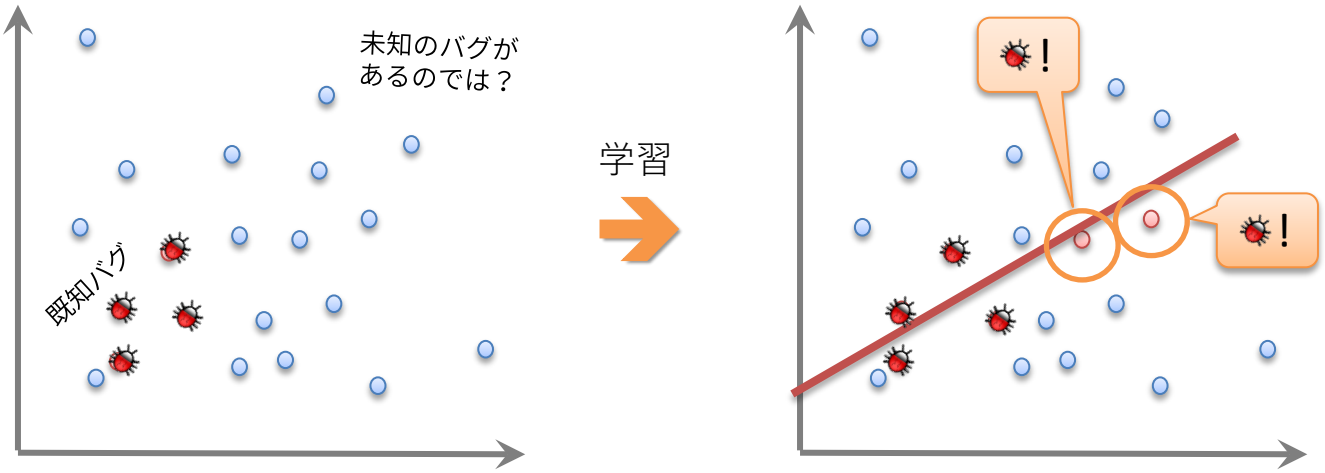
研究アプローチ (帰納的アプローチ)

- リポジトリマイニング・ML4SE
  - 過去の成果物・履歴から再利用可能な経験を自動的に取り出し活用

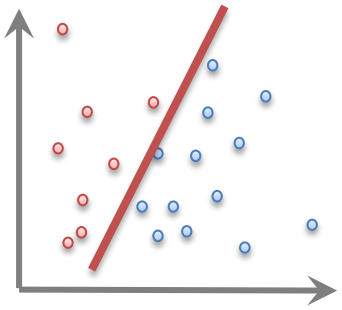


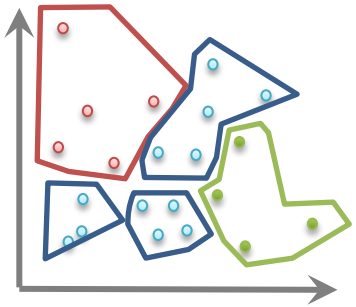
既存データからの経験抽出

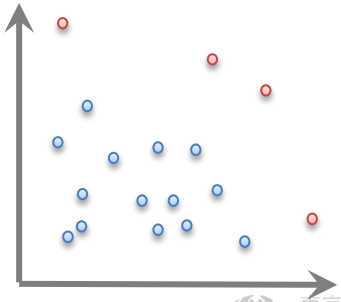
- マイニング・機械学習・統計処理を応用



- 様々な技法を用いて経験を抽出


分類


クラスタリング


外れ値検出

小林研が特に力を入れている領域

- ソフトウェア保守・進化
  - 保守＝既存のソフトのデバッグ・改修
    - バグの箇所を特定し，取り除くための支援
  - 進化＝既存のソフトへの機能追加・改善
    - 明示的・潜在的なバグを生み出さない変更を支援
    - 有益な情報のドキュメント化，情報提示

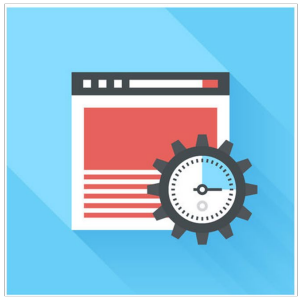
BUG PROTECTION
バグ防止

変更漏れの発見
開発リポジトリをマイニングし変更漏れを検出

Just-in-Time 変更推薦
開発者の行動を分析し，次に変更すべき箇所を推薦

FAULT LOCALIZATION
デバッグ支援

バグ箇所の自動特定
テスト実行の成否状況に基づいてバグの場所を推定

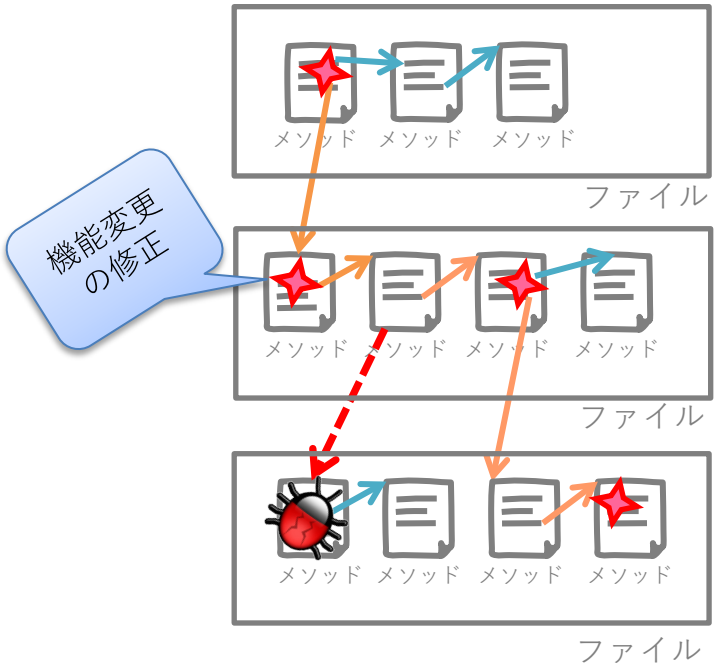


実行情報の抽象化・可視化
実行時のプログラム挙動を分かりやすく提示

バグ防止：変更漏れ発見

バグが混入する1つの要因 = 変更漏れ

- プログラムは複雑な依存関係をもつ
  - 変更波及：変更は様々な場所に伝播する
  - 変更波及を扱う手法＝波及解析 (Impact Analysis)



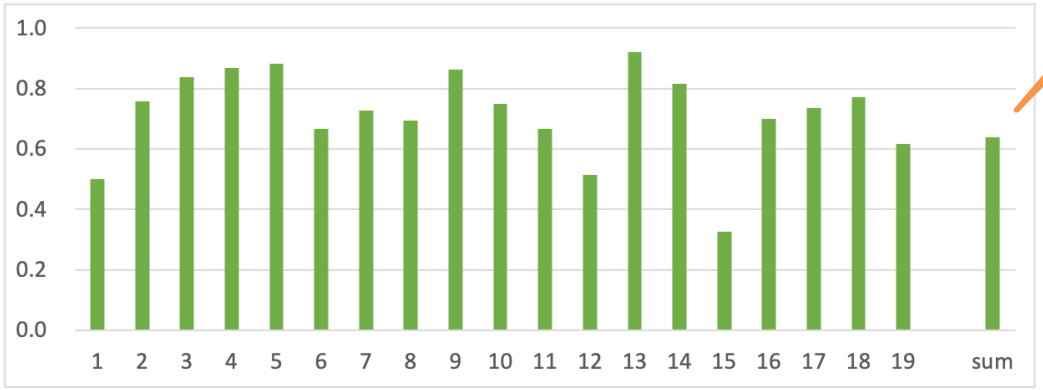
- 静的解析で波及先の特定は(ある程度)可能
  - コードが必須＝フレームワークなどを介すると分断される
  - ポインタ・動的束縛は「可能性」
- 全ての静的依存に波及するわけではない
  - **不要な関係が多い**という報告

「変更漏れバグ」の現状分析 [修論2020]

- 実際に「変更漏れ」はどの程度・どんな規模で起こる？
  - 19 のOSSから 計2840件の バグ管理情報とリポジトリを調査
  - バグチケットから {Bug混入の編集，その修正} ペアを抽出

!! **平均 44.7%** のバグ原因に変更漏れが含まれる
!! 90.7%で 5ファイル以下，**53.2%は1ファイル**

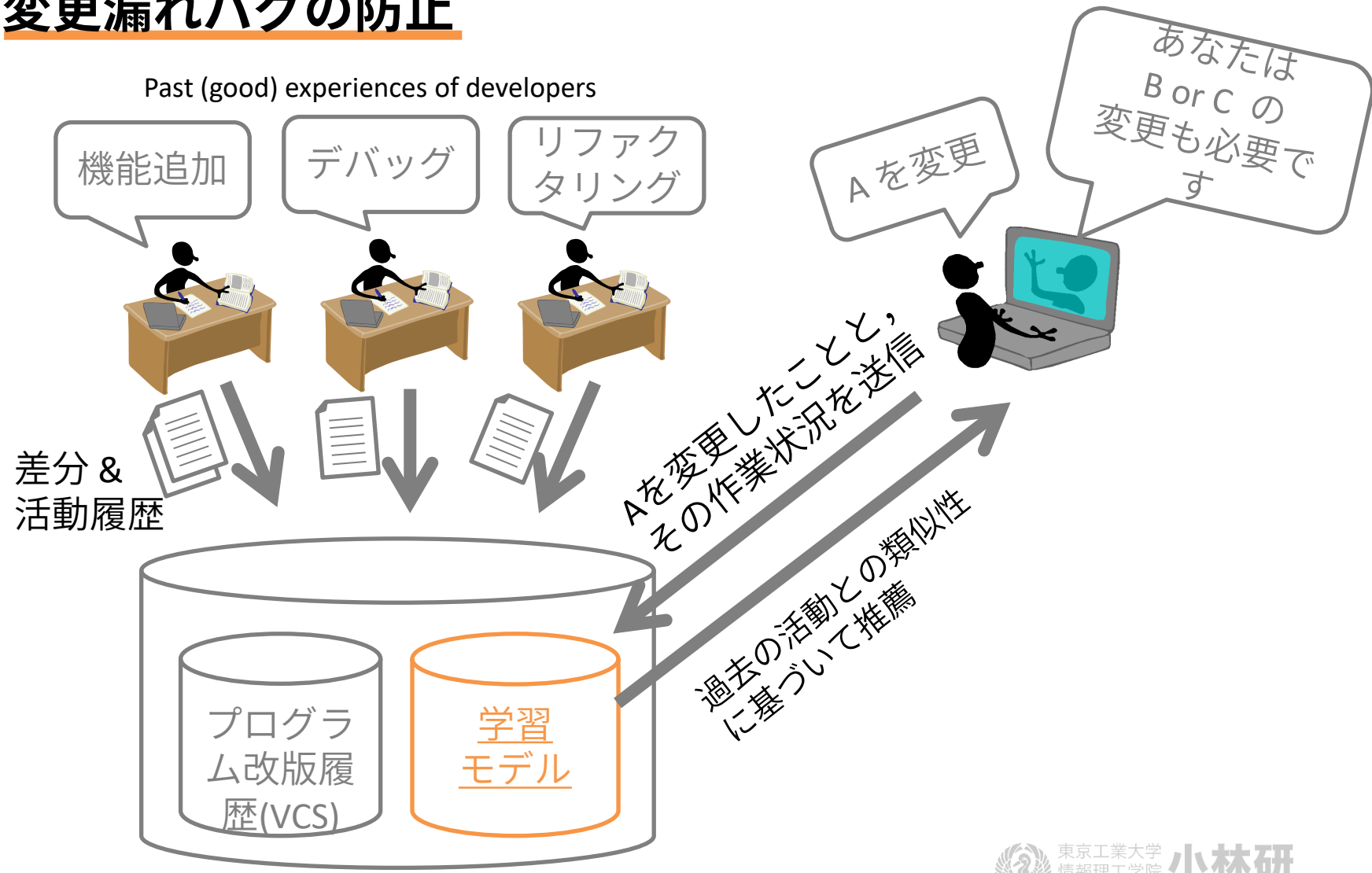
- 改版履歴の分析でどの程度発見できる？



指摘可能な変更忘れファイルの割合

活動の分析による経験(マイクロプロセス)の再利用

変更漏れバグの防止



変更支援手法：ツールの出力例

```
Report for the commit 763a9fba93b91eb730e563eadb08de0444e80980
Changed files in the commit:
  - java/org/apache/catalina/ha/session/LocalStrings.properties
  - java/org/apache/catalina/core/LocalStrings.properties
  - java/org/apache/tomcat/util/descriptor/web/LocalStrings.properties
[1] Top 1 candidate of overlooked change is :
  - java/org/apache/jasper/resources/LocalStrings_ja.properties
because it was 66 % changed when the following your modified files were changed.
  - java/org/apache/catalina/ha/session/LocalStrings.properties
  - java/org/apache/catalina/core/LocalStrings.properties
It happened 2 times in past. For instances :
  +-------------------------------------------------------------------------
  |CommitID  : #4ead17
  |AuthorDate: Thu Mar 30 18:06:46 JST 2017
  |Author    : Mark Thomas<markt@apache.org>
  |Message   : Replace the use of (...) to delimit values in messages wit...
  +-------------------------------------------------------------------------
  |CommitID  : #d46012
  |AuthorDate: Thu Mar 30 06:34:18 JST 2017
  |Author    : Mark Thomas<markt@apache.org>
  |Message   : Fix https://bz.apache.org/bugzilla/show_bug.cgi?id=60932 Corr...
  +-------------------------------------------------------------------------

[2] 2nd  candidate of overlooked change is :
  - java/org/apache/jasper/resources/LocalStrings_fr.properties
…
```

開発者の変更内容

変更忘れ候補

理由

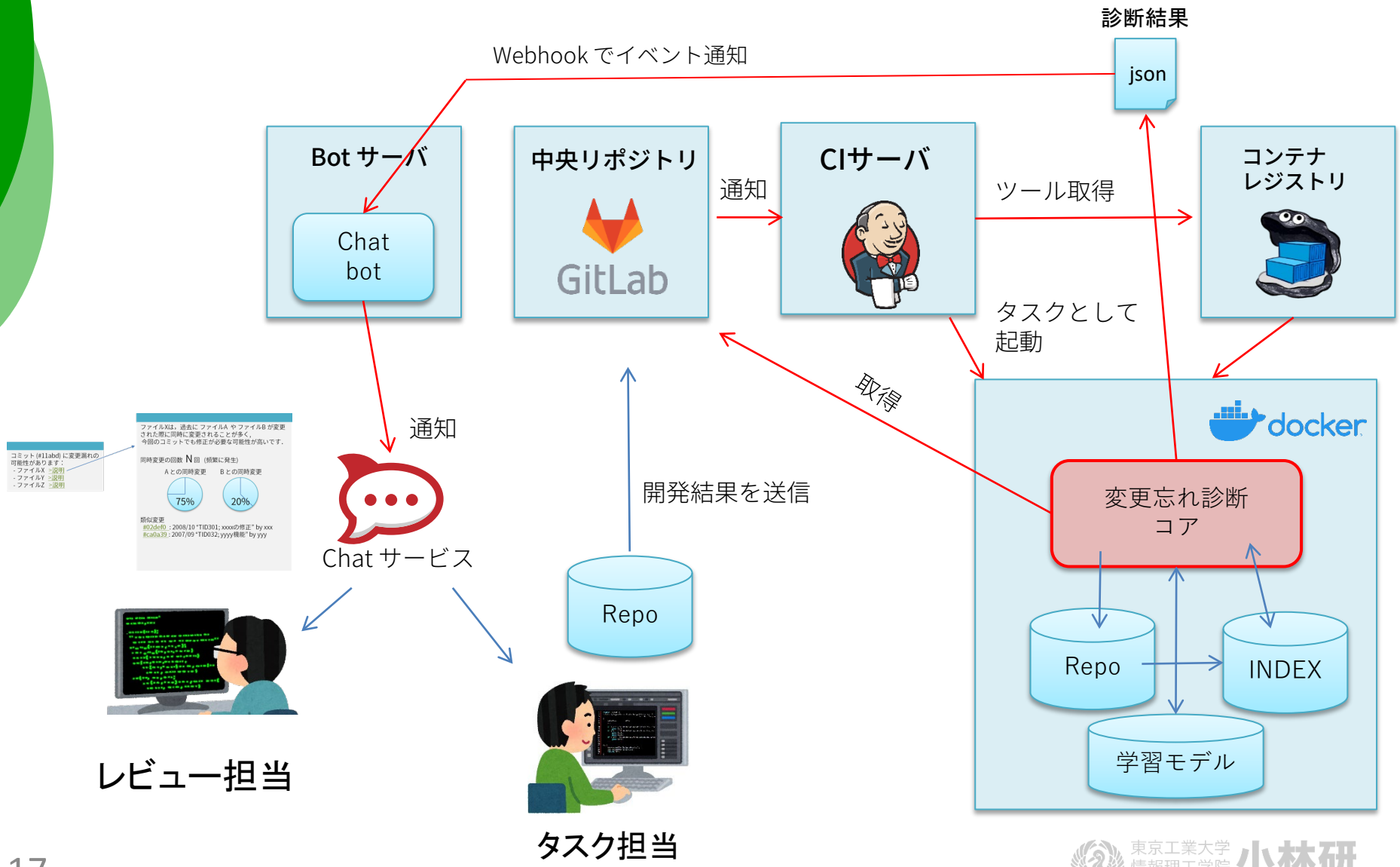
条件付き確率 66%

過去の類似変更

変更支援手法：我々の研究成果

- 変更履歴の質と推薦精度の関係を分析 
  - 「古い履歴は使うべきではない」「分割コミットは悪影響」
  - EclipseやFirefox等の大規模OSSリポジトリで検証
    - 2017年度情報処理学会論文誌 論文賞受賞
- Configurable Software(CS) 向け拡張
  - 前処理の影響と変更箇所の関係を考慮した推薦
- マイニング範囲の動的決定 
  - マイニングする履歴の量を推薦毎に動的に自動決定
  - 12のOSSで固定値より優れた精度を実現
- 機械学習を用いた変更忘れ指摘手法の開発
  - データマイニングではなく機械学習での解決方法を提案
  - 古典的ML，DNNを用いた複雑な関係性の推定

変更支援手法：ChatOps環境への組み込み (共同研究)



複合コミットの問題

- 複合コミット(CC)：複数の目的を含むコミット
  - a.k.a Tangled Commit, Multi-purpose change…
- 問題
  - 複数の変更が「絡まる」と分かりづらい
    - レビューや再利用が困難

  - コミット解析に基づく手法に悪影響 [5, 21]
    - 「大きな変更は除外」等の経験則で対応
    - 有効な共変更関係を見落とす可能性

本研究のきっかけ

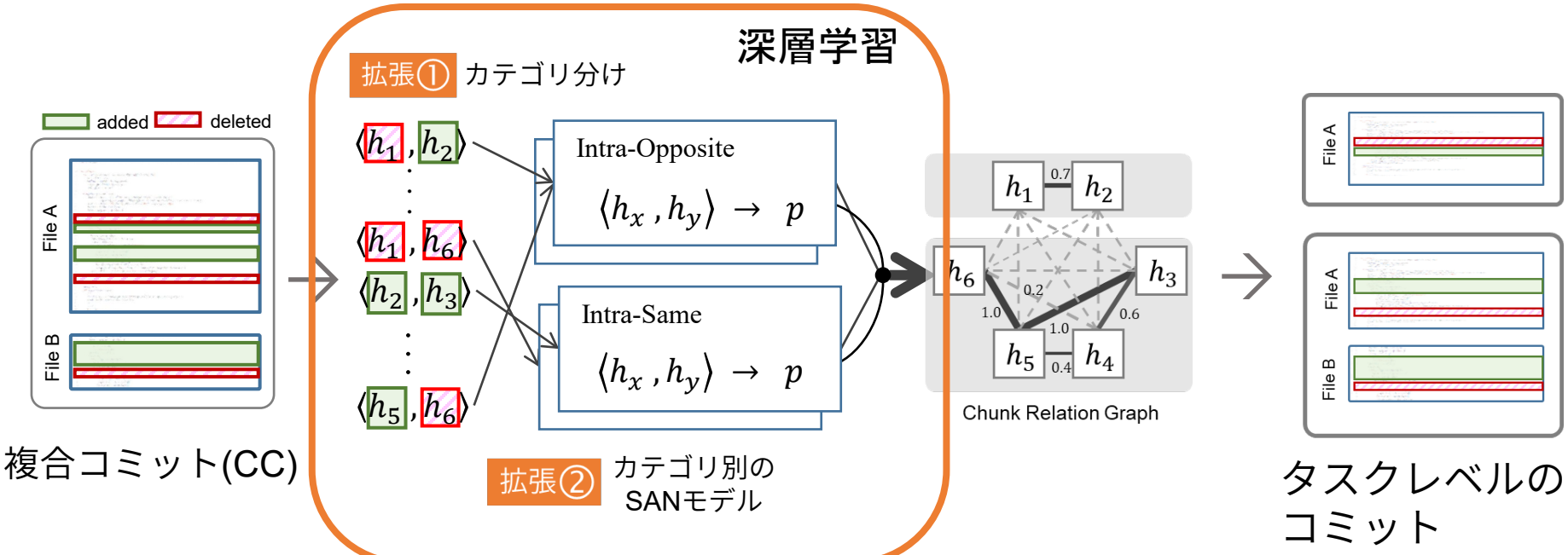
[5] H. A. Nguyen et al., “Filtering noise in mixed-purpose fixing commits to improve defect prediction and localization”, Proc. ISSRE, 2013

[21] K. Herzig et al, “The impact of tangled code changes on defect prediction models”. Empirical Software Engineering 21(2), 2016

変更支援手法：精度改善にむけて

- コミットを適切な粒度に分解 (’19〜)
  - 「連続する変更行」の単位で抽出
  - DNN (SAN, Tree-based CNN)による特徴学習

→従来手法より高精度にコミットを分割

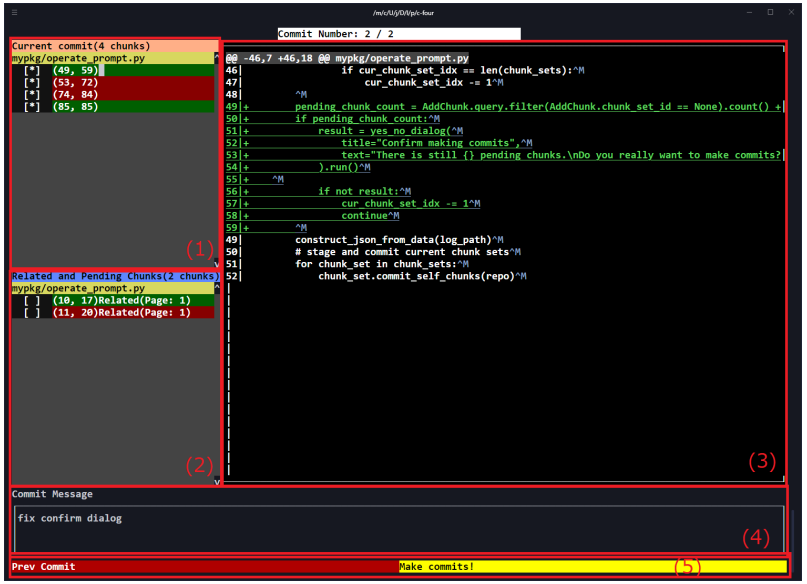


変更支援手法：精度改善にむけて

- コミット分割の応用：自動ステージング

- CLIツール を開発
  1) コミット忘れてた!
     → 起動
  2) 分割案提示
  3) チャンク単位で
     自動分割案を調整
  4) 連続コミット

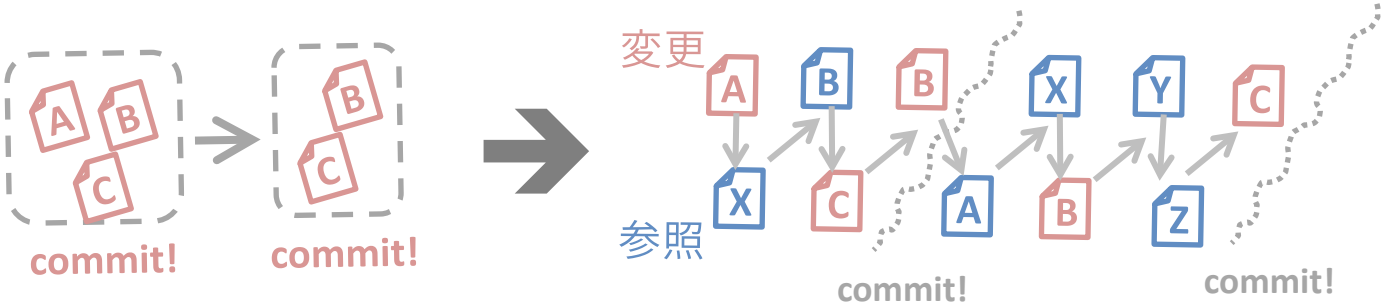
CLIツール c-four スクリーンショット



Available at
github.com/tklab-group/c-four

より詳細なマイクロプロセスの抽出

- **操作履歴** の解析に基づく変更推薦
  - 操作履歴：Commit間に「どんな操作をした」か



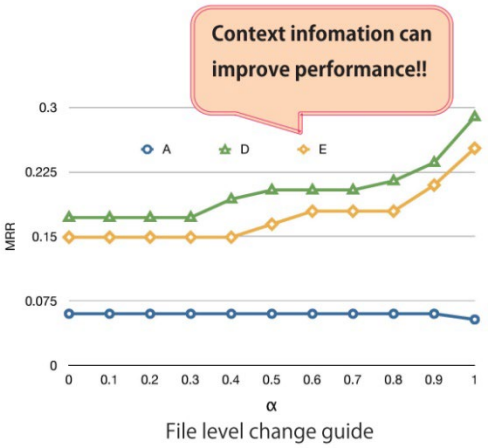
変更履歴 ＝ スナップショット　　操作履歴 ＝ 開発者がどのように変更したか

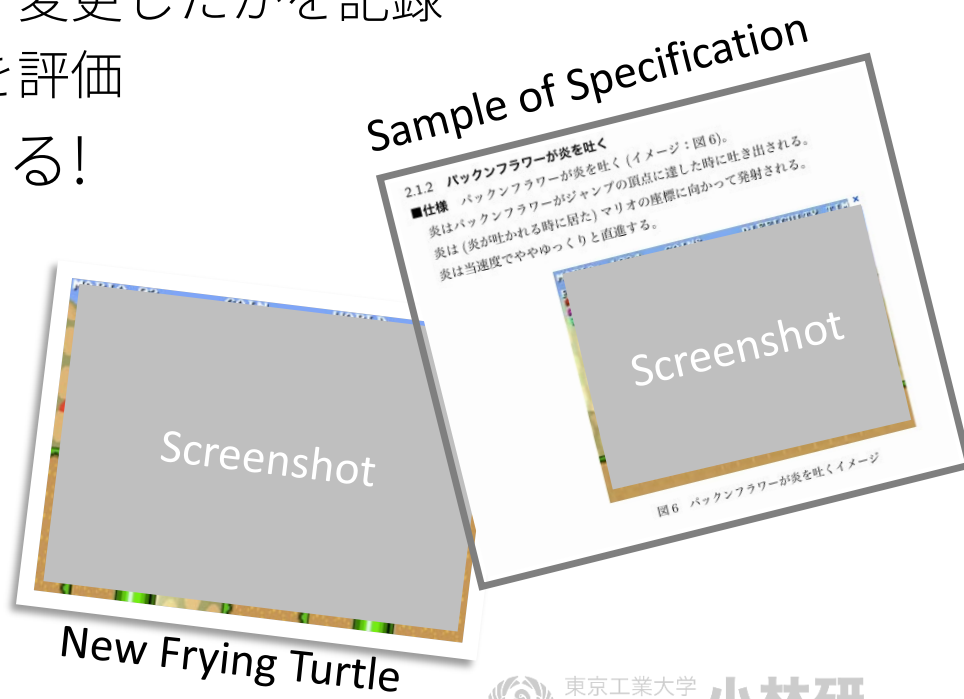
  - 参照先や変更の順序から，「変更の文脈」を抽出
    - 「ファイルXを変更する」には複数の意図がある
    - 仮説：文脈が類似するならば変更意図も類似する

- 国際会議 MSR2018 のコンテスト題材になるなど，操作履歴分析は（ようやく）着目されてきた

より詳細なマイクロプロセスの再利用実験

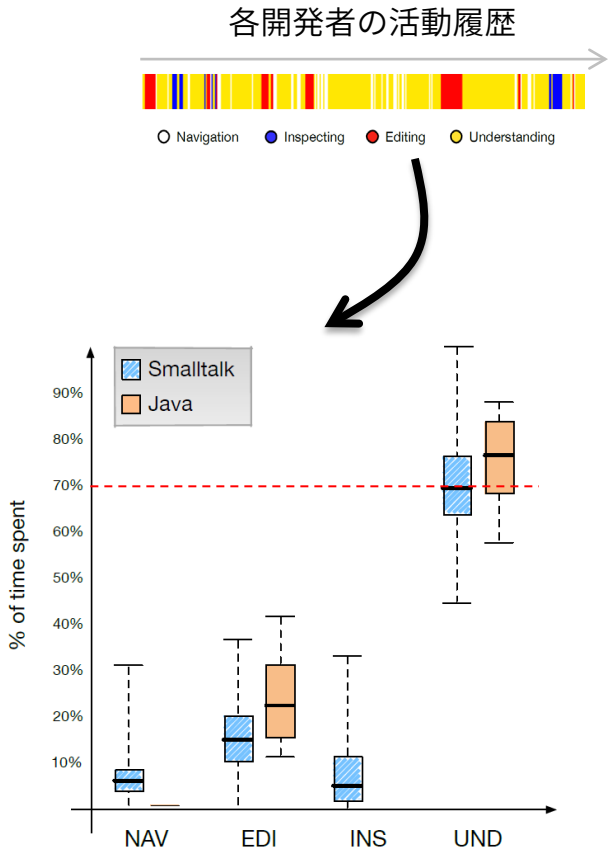
- 過去の開発経験を応用できるか？
  - 学生 17 名で「陽気な配管工が活躍するゲーム」を拡張
  - 例：Hammer Marioの導入，パックンフラワーが炎をはく
- 開発活動を記録して学習
  - どのファイル・関数を参照・変更したかを記録
  - 次の変更を予測できるか？を評価
- 現在までの結果：結構当たる!





変更支援手法：国際共同研究への発展

- 開発者の活動分析 (2014)
  - スイス Lugano大 と実施
  - 操作ログを持ち寄り分析
    - Java: 15名，SmallTalk 7名
  - 発見：70+%を 理解に費やす
    - それまで定説では 35%程度



- 長期開発データの分析 (2017)
  - ノルウェー Oslo大と実施
  - 4名の企業開発ログを分析
    - 先方は別の目的でログを取得

小林研が特に力を入れている領域 [再掲]

- ソフトウェア保守・進化
  - 既存のソフトへの機能追加
    - 明示的・潜在的なバグを生み出さない変更を支援
  - 既存のソフトのデバッグ
    - バグの箇所を特定し，取り除くための支援

BUG PROTECTION

バグ防止

変更漏れの発見

開発リポジトリをマイニングし変更漏れを検出

Just-in-Time 変更推薦

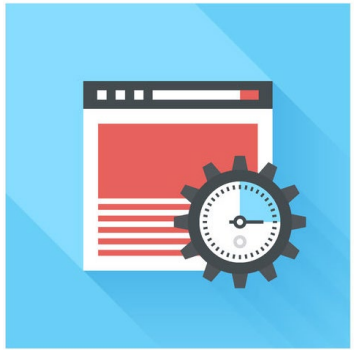
開発者の行動を分析し，次に変更すべき箇所を推薦

FAULT LOCALIZATION

デバッグ支援

バグ箇所の自動特定

テスト実行の成否状況に基づいてバグの場所を推定



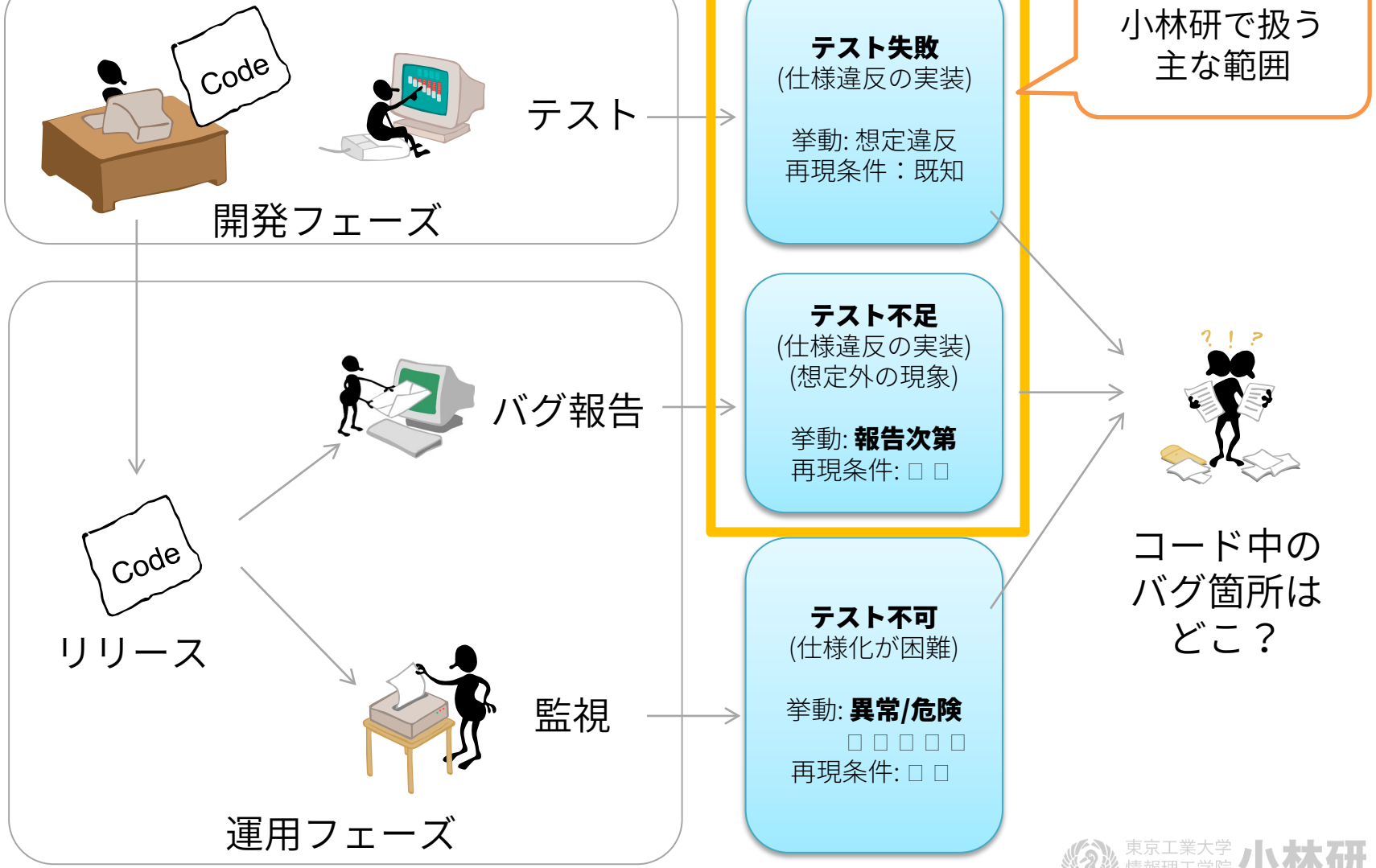
実行情報の抽象化・可視化

実行時のプログラム挙動を分かりやすく提示

デバッグ支援の取り組み

- バグ発見・個所特定
  - 情報検索によるバグ箇所特定 (2020〜)
  - 深層学習によるバグ含有予測 (2019~)
  - 動的解析を用いたバグ箇所特定 (2011~) 
  - 大規模実行トレース解析による異常検知 (2019) 
- 新しいデバッグ環境の開発
  - 仮想ファイルシステムを用いたデバッグ環境 (2017~) 
  - Jupyterを用いたデバッグ環境 (2020)
- 振る舞い・機能に特化したプログラム理解支援
  - 実行履歴の抽象化・可視化(2009~) 
  - 機能実現個所の特定 (2012~)
  - プログラム要約 (2017〜)

デバッグ支援：
バグ発見・原因箇所特定



バグ箇所の自動特定

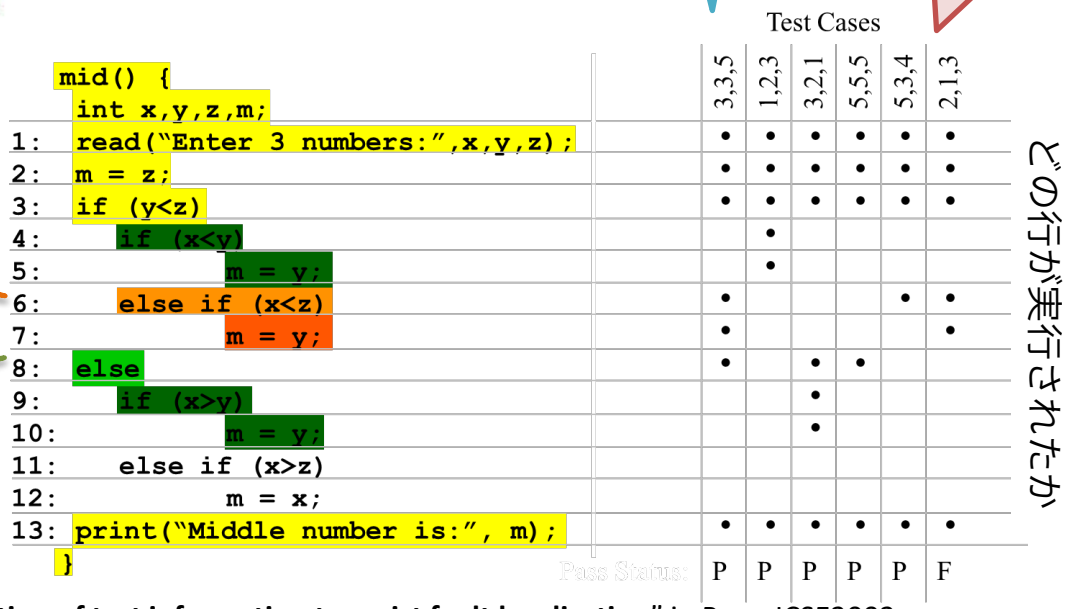
- Spectrum-based Debugging
  - 成功実行と失敗実行の差分から，
    失敗の原因となった「バグ原因箇所」を推定
  - コード解析・理解なしに推定可能
- 詳細な実行トレースの分析による
  新しい手法を考案

(x,y,z) = (3,3,5)
の時はPass(成功)

(x,y,z) = (2,1,3)
の時はFail(失敗)

成功実行で通ることが少なく、
失敗実行で通ることが多い

成功実行で通ることが多く
失敗実行で通ることが少ない

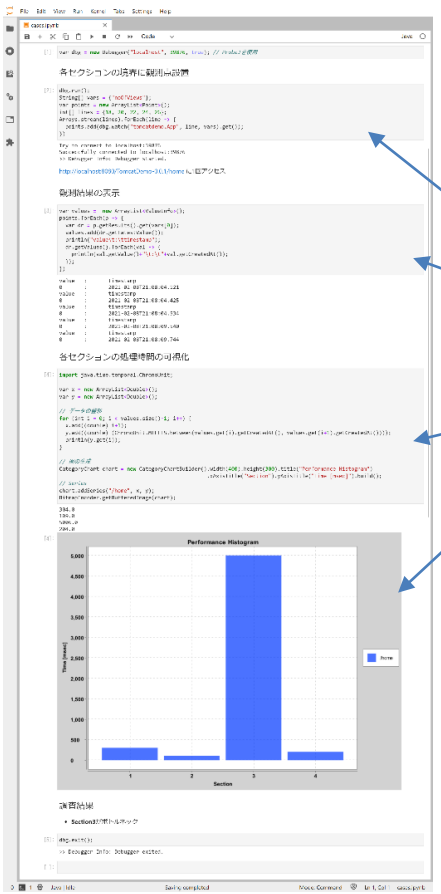



```
mid() {
  int x,y,z,m;
1:  read("Enter 3 numbers:",x,y,z);
2:  m = z;
3:  if (y<z)
4:    if (x<y)
5:      m = y;
6:    else if (x<z)
7:      m = y;
8:  else
9:    if (x>y)
10:     m = y;
11:   else if (x>z)
12:     m = x;
13: print("Middle number is:", m);
}
```

[Jones+2002] Jones et al. "**Visualization of test information to assist fault localization**" in Proc. ICSE2002
図は http://pleuma.cc.gatech.edu/aristotle/Tools/tarantula/vizworkshop.ppt Slide p12 より. 引用

文芸的デバッグ環境

- 新しいデバッグ支援環境 JISDLab を開発



- 観測の影響を最小限に
  - 実行を停止せずに観測
  - 実行中に「観測したい値」を変更可能
- Scriptable Debugging + Literate Program
  - デバッグ作業をスクリプト記述
  - 観測結果の加工・保存も可能
- “実行できるバグレポート”の実現
  - 不具合の再現・状況確認・解析を可能に

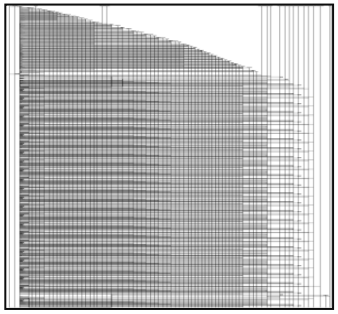
Webブラウザから
Web App @Tomcat をデバッグ

Available at
github.com/tklab-group/JISDLab

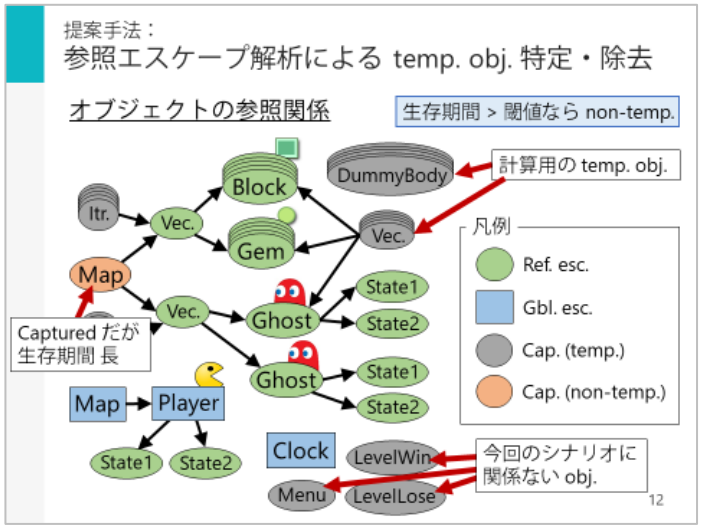
東京工業大学 情報理工学院 小林研
Software Analytics Research Group

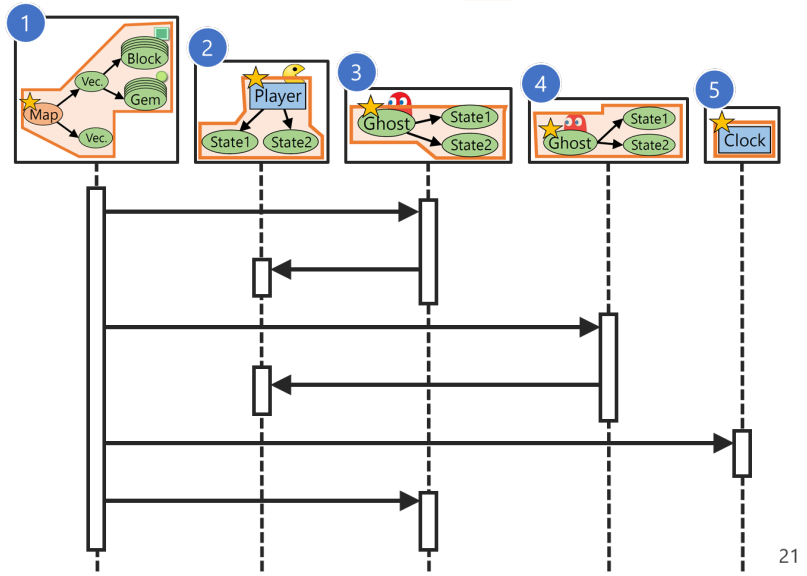
実行履歴のシーケンス図による可視化

- 実行履歴の抽象化・可視化
  - 一般的なソフトウェアの実行 ＝ 可視化困難
  - プログラムを解析しオブジェクトをグループ化
  - 全体像が把握しやすい「実行概要」を提供



生成したシーケンス図





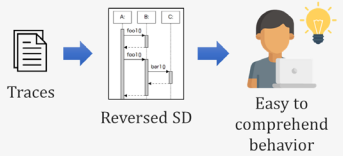
提案手法で生成する実行概要シーケンス図

シーケンス図の活用

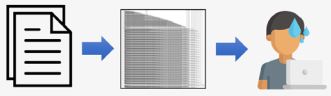
- SDExplore: 大規模SD表示ツールキット
  - Won Best Tool Demo Award @ IEEE/ACM ICPC 2018

Program Comprehension with Reversed SD

Ideal



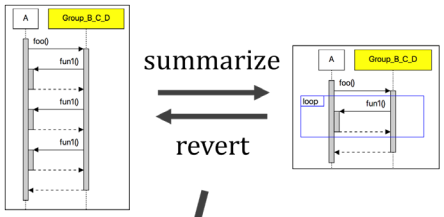
Real



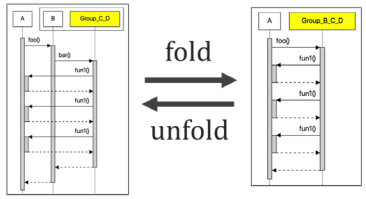
Existing tools…
- Sometimes **Crash**
- **Hard** to **re-use**

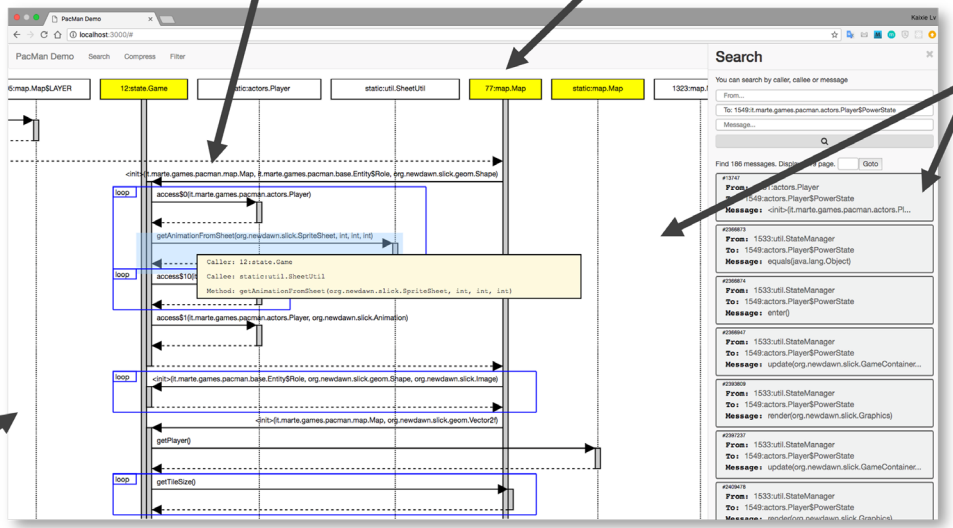
… etc.

Interactive Loop Summarization



Interactive group folding/unfolding





Rich features for exploration
- Searching
- Filtering
- Zooming

… etc.

Operation Logging

Records interactions to support user activities analysis.

Browser-based tool
- Easy to use
- Platform-free
- Embed to Web page

Smooth exploration for massive-scale SD

< 1 sec responses time for SD with **3K** objs, **2.5M** msgs

そのほかの取り組み：コード要約

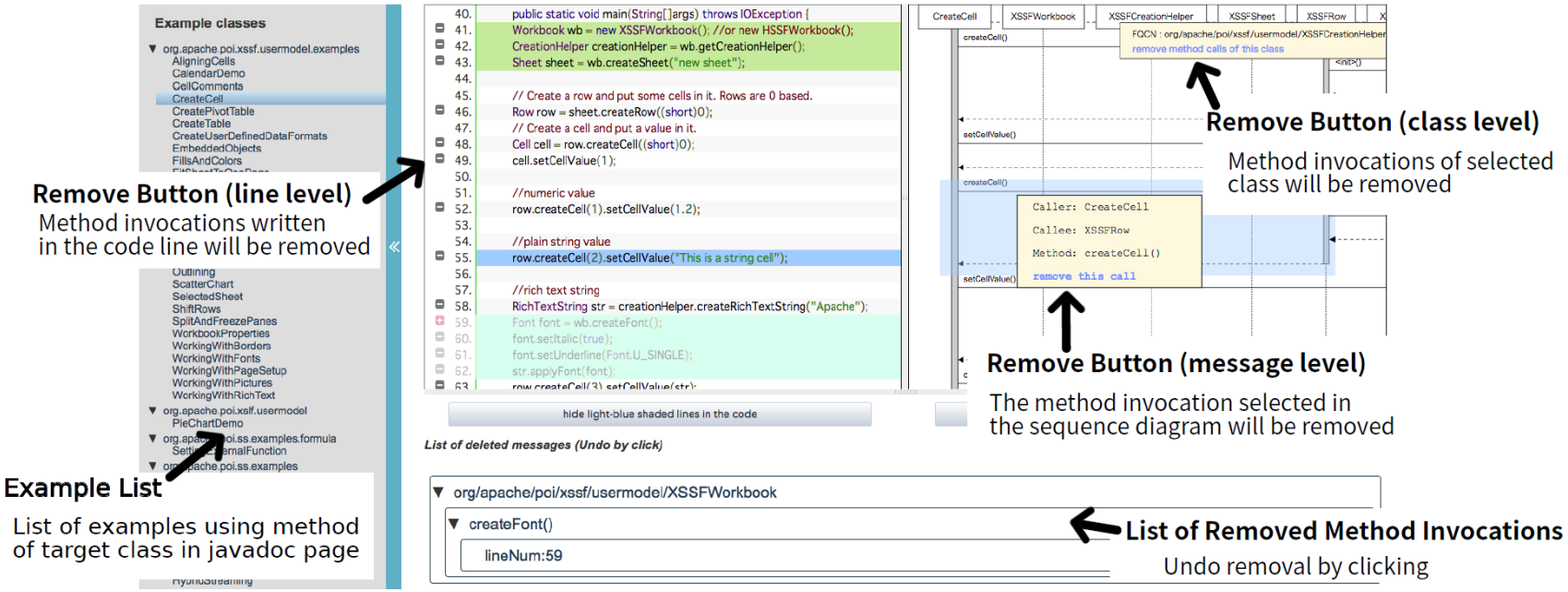
- 保守工程の大半は「コードの理解」
  - ドキュメントは欠損・整備不良が多い
  - 「可読性の高いコードを書く」が唯一の対応策
- 新しい流れ：ドキュメント生成・コード要約
  - コードからドキュメントを生成
    - 既存手法の多くは抽象度が低い (例: Javadoc API reference)
    - 文書要約技術を応用した「要約」を生成がトレンド
  - 欲しい情報は多種多様
    - このクラス何しているの？
    - この機能どこでどうやって実現されているの？
    - このパラメータをONにすると何が起こるの？
    - このAPIどう使うの？
    - この実装なんでこうなってるの？

‘19修論

‘18 学部課題 x2

そのほかの取り組み：コード要約

- API利用テンプレートを対話的に生成
  - 振る舞いをSDで確認しサンプルコードを修正
- JavaDoc ページに埋め込み可能



※ 18年度 B4課題(2か月)，DocGen2018@Madrid で発表

マクロスイッチ・フラグの説明文生成

- マクロスイッチ・フラグ
  - 設定ファイルを用いた機能切り替えは広く用いられる
  - 特に Configurable Software (IoT, 組込み) で多用される
  - 明確な説明書・コメントは少ない → 暗黙知となりがち

- これまでの取り組み
  - C言語前処理マクロの大規模コーパス作成
    - 1.41億行 のC言語コード → マクロ変数 630万種類，5000万出現
  - 変数コメントの存在・意味的情報の分析 [**MSR ’21 採択**]
    - Java の基本型 + String 変数 の コメントを抽出 (1.7Mコメント)
    - コメントが不足していることを実証的に示した
    - DySDoc のメンバーとの国際共同研究

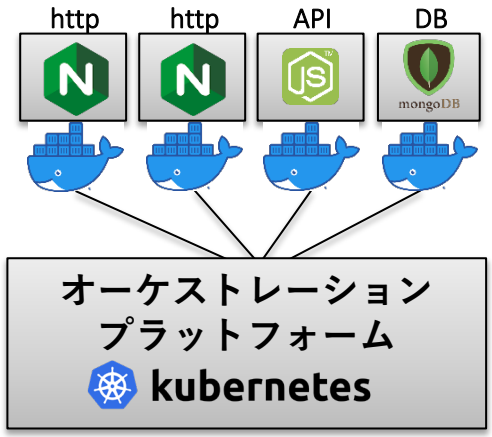
これまでの取り組み：SE 4 IaC

- VM・コンテナによる環境(インフラ)構築
  - 例：レンタルサーバ, Amazon AWS, Dockerの利用
  - 手作業による間違い防止・効率化が必要
  - IaCスクリプト記述による自動化が主流に
- IaCスクリプトも「コード」
  - 多量のIaCスクリプト ＝ やはり理解しづらい
  - ソースコードと同様にソフトウェア工学が必要
- SE 4 IaC
  - 設計支援：記述支援，推薦・警告，性能見積もり
  - 保守支援：理解支援，バグ発見，リファクタリング
  - …

19年度 卒論

21年度 卒論

これまでの取り組み：IaCの可視化



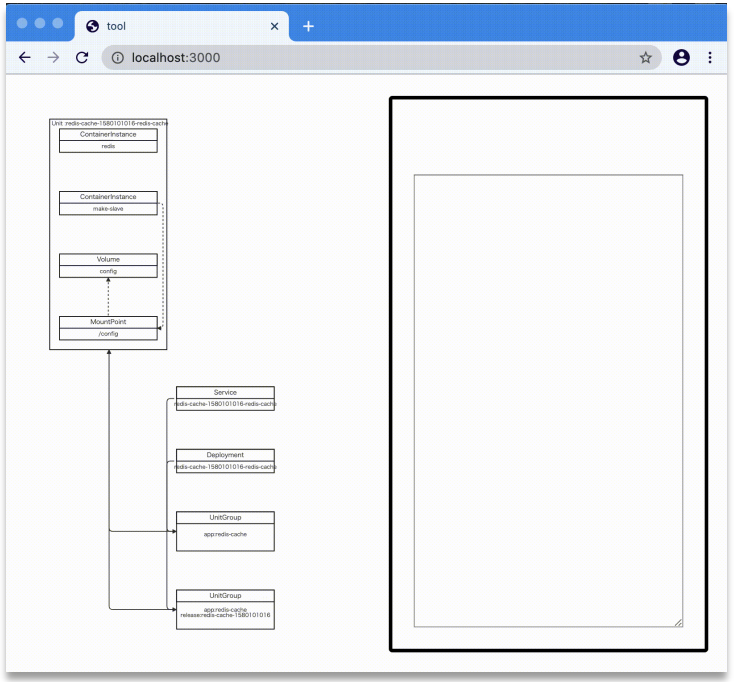
構築時に記述

理解困難

```
kind: Deployment
spec:
  replicas: 2
...
containers:
      - name: nginx
        image: docker/nginx...
        imagePullPolicy:
Always
...
      - name: web
        image: docker/.....
```

IaC スクリプト記述

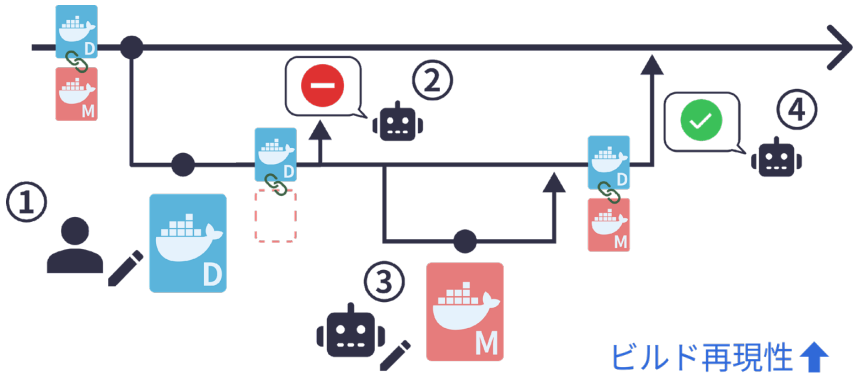
- **ソースコードと同様に**
  リバースエンジニアリング手法を適用
- 実際のサービス設定ファイルに適用

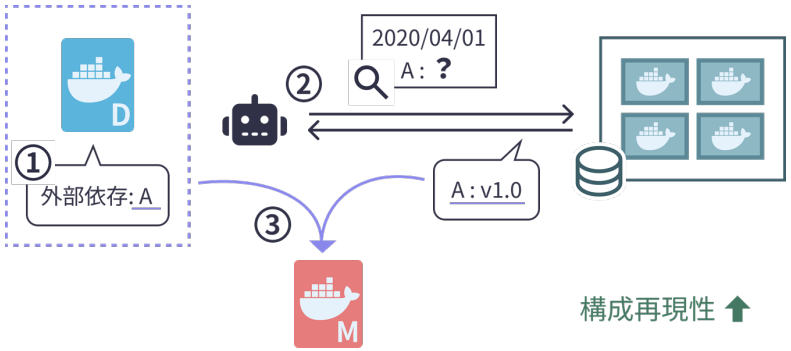


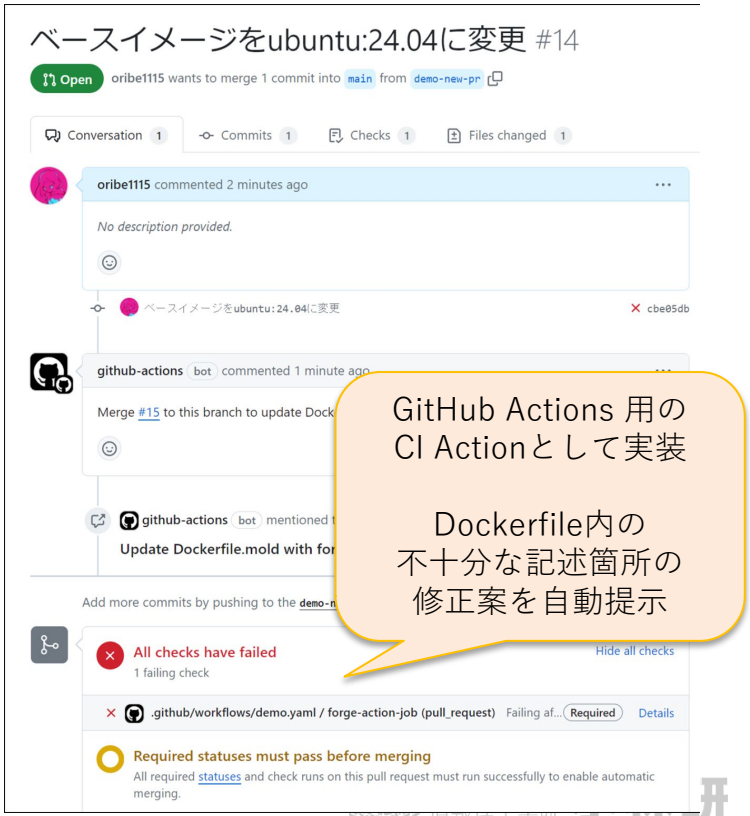
Kubernetes用可視化ツール (19年度 卒論)

これまでの取り組み: Dockerfileの保守支援

- CDCM: Continuous Dockerfile Configuration Management
  - 構成再現性を担保：正確な外部依存先を記録
    - 意図しない外部依存先の変化も検知・指摘







2024年度メンバー

- 学生は13名前後
  - 教員：小林
  - M2：4名
    - バグ箇所検索 x 2，変更パターン，リポジトリ診断
  - M1：6名
    - WebUIテスト共進化，デバッガ，変更分割，未定x3
  - 学部：2名
    - 未定 x2
  - 短期滞在・交換留学： 若干名

例年修士は 3〜4名/学年
他大学からの
入学も多数実績あり

- 連携研究室・大学・企業
  - 林研@大岡山とはいろいろ連携
  - 大学：京大，はこだて未来大，スイス Lugano大，濠 Adelaide大
  - 企業：IT/SI企業 3社

必要な知識・関連講義など

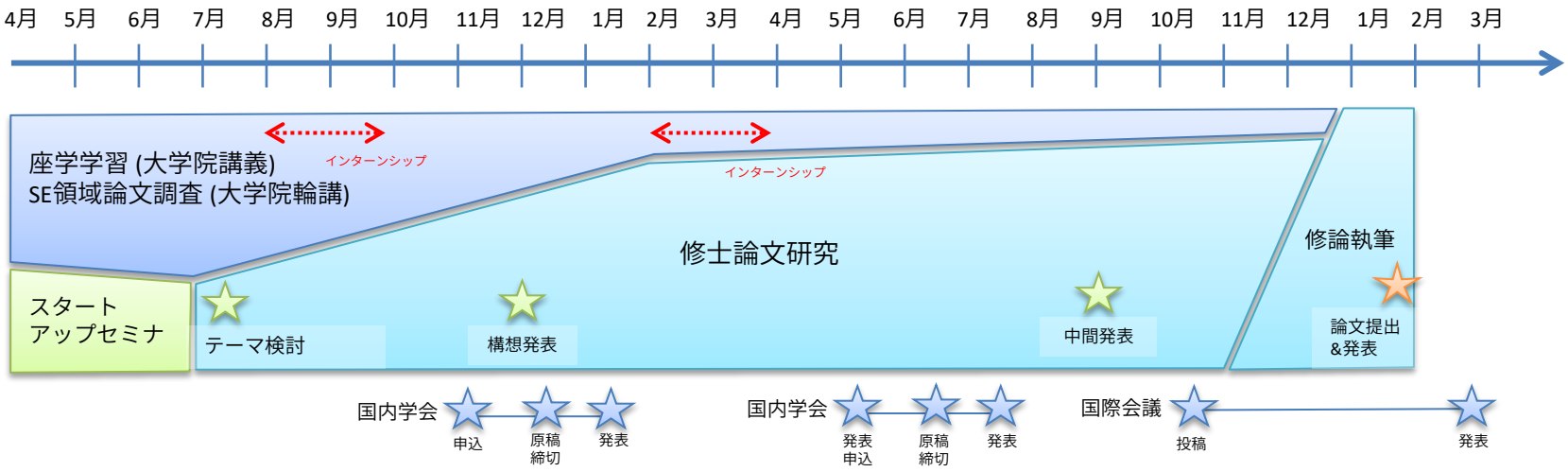
- 必須知識ではありませんが以下は関連します
  - 手続き型・オブジェクトプログラミング言語
  - 数理論理学，オートマトン，コンパイラ
  - データベース，情報検索，データマイニング，機械学習
- ソフトウェア開発・プログラミングで
  困ったことがある人はすごく向いています
  - その内容が研究のきっかけです
  - 開発環境に不満 → 支援ツール は良くある話です
  - 開発者を楽に・幸せにすること＝SEのゴールの1つ
- 開発スキルを得たい人も向いています
  - その動機に関連する書籍・事例はたくさん提供できます

最近の言語事情@小林研

実装言語：Java / python / Go
解析対象：Javaが多い

C/C++/C# も扱う．珍しいところだと
COBOL, Simulink, (Docker, Kubernetes)

４月入学修士のスケジュール



- 基本スケジュール
  - **2024年度から週3日間のコアタイム制をとります**
    - 研究室にいれば研究が進むわけではないですが，来ないと全く進みません．
  - 大学院輪講(週2時間)，研究会 (週2時間) に参加
    - 前期はこれに加えて スタートアップセミナー(1.5h*8回程度)に参加
  - 大学院輪講は論文紹介が基本
  - M1後期から研究会で毎週進捗を確認

東京工業大学
情報理工学院 **小林研**
Software Analytics Research Group

2022年から **すずかけ台キャンパスに移転** しました
最寄り駅： 東急 田園都市線 すずかけ台 駅

問合わせ先：

小林隆志 <**tkobaya@c.**titech.ac.jp>

研究室Webページで過去の修士論文や
外部発表論文も参照してみてください
https://www.**sa.cs.**titech.ac.jp/