



本資料公開場所

Call for Members for AY2026

小林研の研究内容の紹介

難しいこと面倒なことを
自動化することで
「ソフトウェア開発を
楽し~~た~~いたい人」
を募集します



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

本年度実施中のテーマ

- **不完全な変更の防止・変更情報の修正**
 - M2：言語モデルベースの類似変更検索に基づく不完全変更の推定
 - M2：言語横断学習に基づく多タスク混入コミットの自動分割
- **テスト支援**
 - M2：Reactアプリケーションを対象としたWeb UIテストの自動修復
 - B4：LLMテスト生成におけるJML仕様記述の効果調査
- **行動解析**
 - M2：入力デバイス操作の分析に基づく開発者の関心位置推定
- **デバッグ**
 - M2：制御依存解析に基づくバグ原因箇所推定
 - M1：データフロー解析に基づくバグ原因箇所推定
 - M1：バイトコード解析に基づくバグ原因箇所推定
- **ドキュメント生成**
 - M2：Graph-RAGを用いた マイクロサービス向けReadme生成手法
 - M1：LLMベースのコミットメッセージ生成手法
 - B4：Graph-RAGを用いた Vue.js アプリケーションの動作説明文書生成
- **要求分析**
 - D1：利害関係者間の要求の差異を考慮した要求分析手法

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

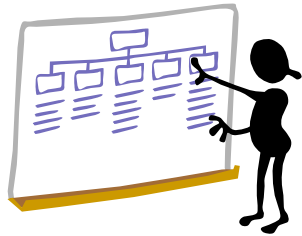


なぜ「知識や経験」に着目するのか？

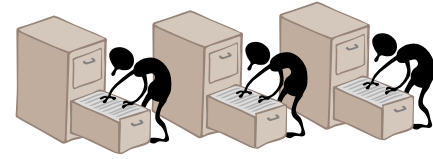
- ソフトウェア開発は高度な知的創造活動
 - 実現している機能 ⇔ ソースコードの変換作業は現在でも説明困難な活動
 - 開発者の理解・行動プロセスの補助が必要
- 2方向の支援アプローチ
 - 演繹的：プログラムの意味を解析し補助
 - 部分的なプログラムの意味を用いて理解・解析を支援
 - 命令単位での意味 → 関数単位での意味 → …
 - 支援の確度は高いが，適用範囲が限定的になりがち
 - 帰納的：事例(プログラム・活動)を解析し補助
 - 多数の事例から意味/意図を導出：ML based AI的アプローチ
 - 適用範囲が広いが，支援の確度は事例数と共通性に依存

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

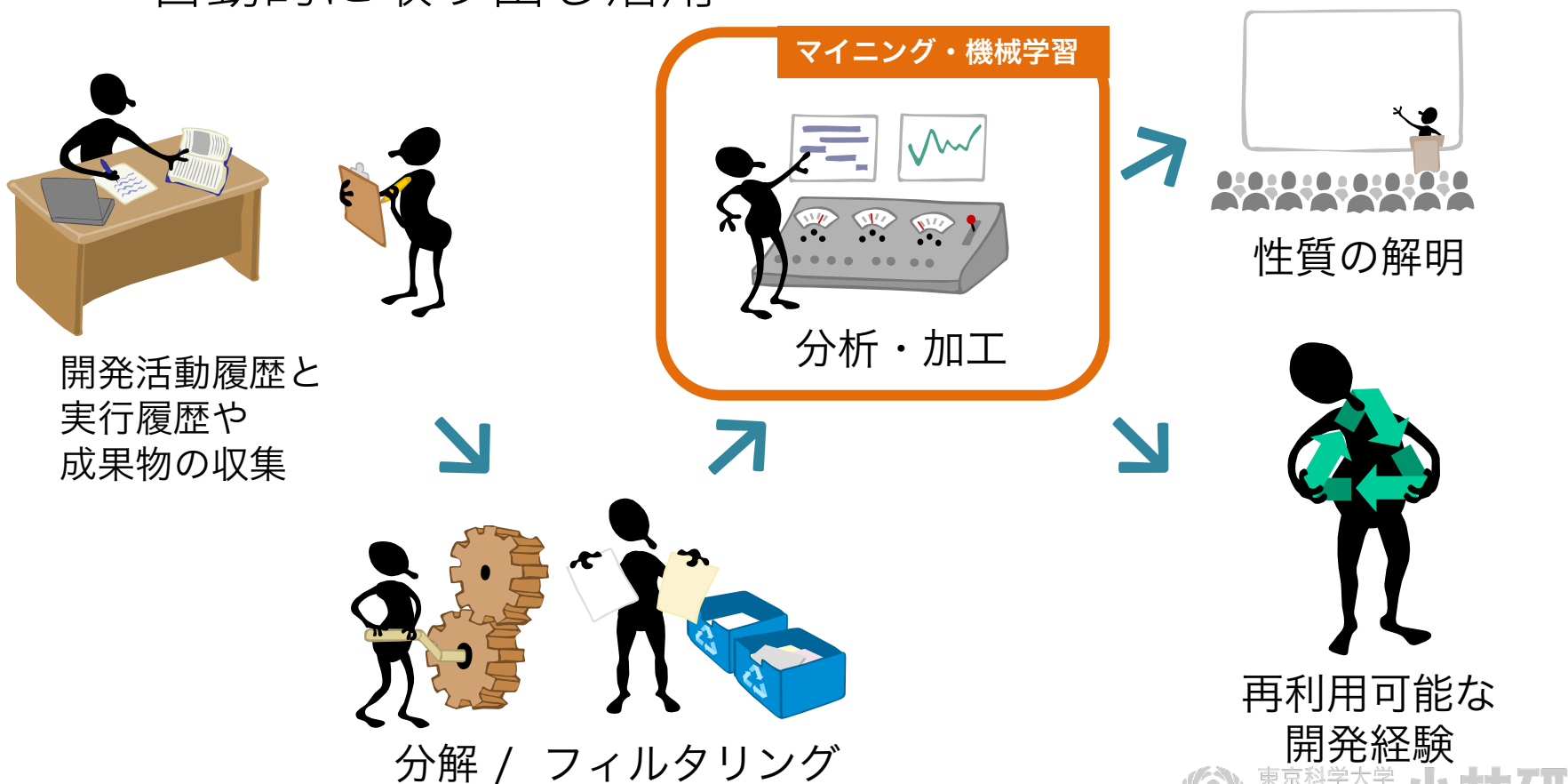
- 作業の自動化
- 作業結果の検証
- 作業内容の推薦

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)

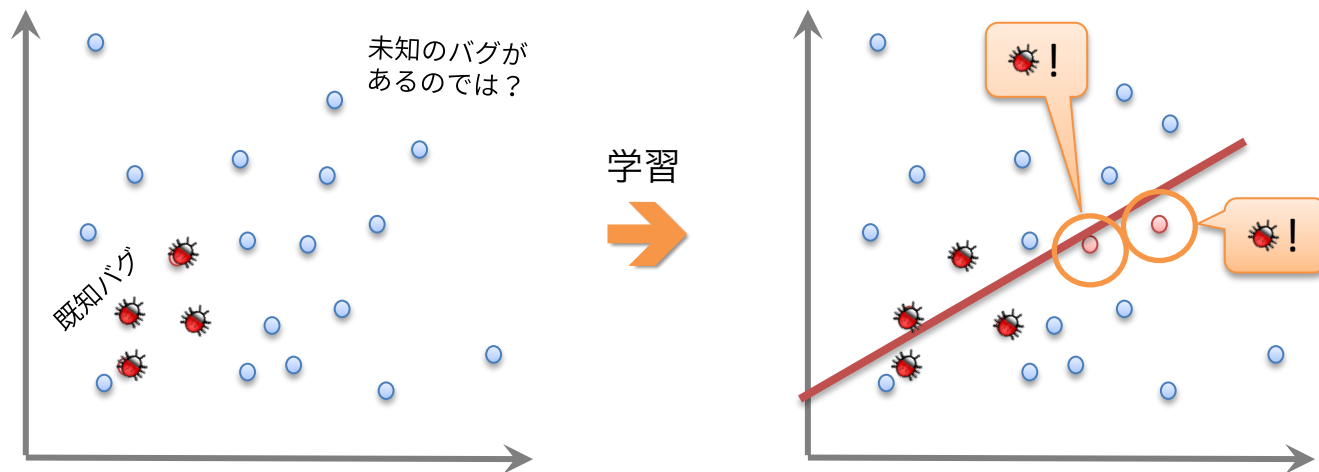
○ リポジトリマイニング・ML4SE

- 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

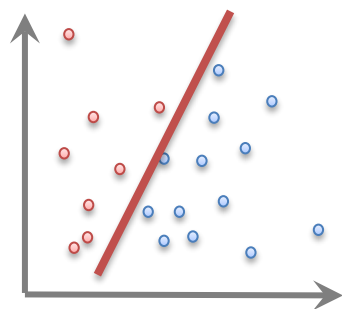


既存データからの経験抽出

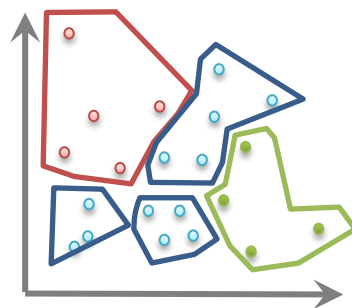
- マイニング・機械学習・統計処理を応用



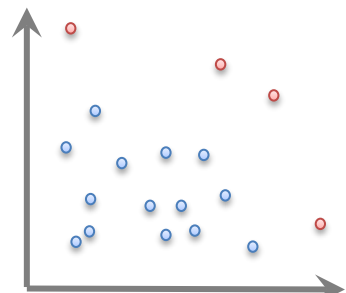
- 様々な技法を用いて経験を抽出



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

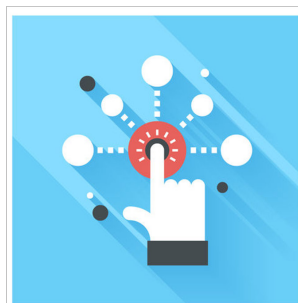
- ソフトウェア保守・進化
 - 保守＝既存のソフトのデバッグ・改修
 - バグの箇所を特定し，取り除くための支援
 - 進化＝既存のソフトへの機能追加・改善
 - 明示的・潜在的なバグを生み出さない変更を支援
 - 有益な情報のドキュメント化，情報提示



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



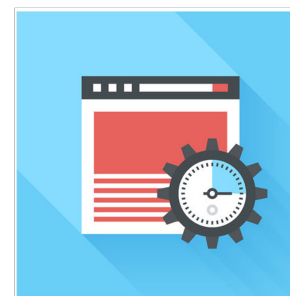
Just-in-Time 変更推薦
開発者の行動を分析し，次に変更すべき箇所を推薦



FAULT LOCALIZATION
デバッグ支援



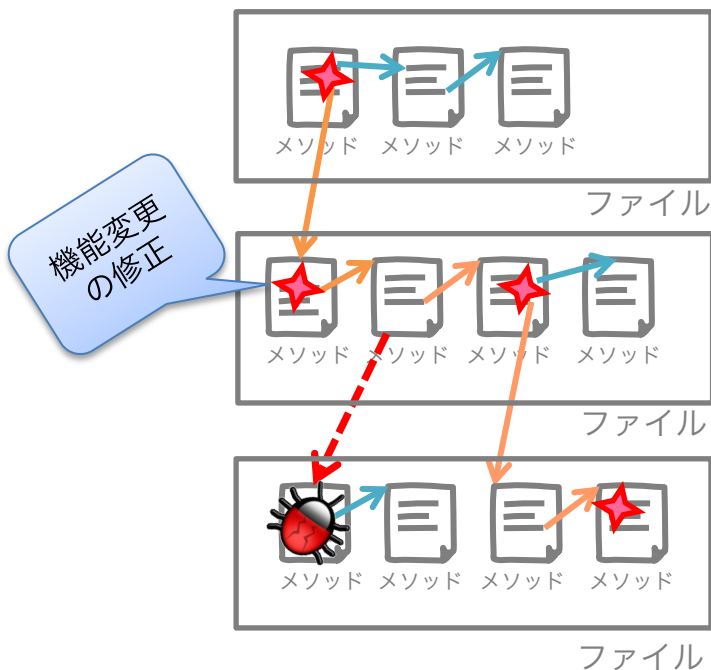
バグ箇所の自動特定
テスト実行の成否状況に基づいてバグの場所を推定



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

バグが混入する1つの要因 = 変更漏れ

- なぜ変更漏れが見逃されるのか？
 - その影響伝播が起こるケースの把握不足
 - 十分なテストケースが用意されていれば防げる(はず)
- 全てのケースをテストすることはほぼ不可能
 - 実開発においては時間・コスト制約が強い
 - 自動テストでも「テスト実行時間」が問題に
- Agentic coding でも回避は難しい
 - AIによる修整の検証はテスト実行
 - テスト生成が不十分なら変更漏れが起こりうる
 - テスト生成には(文書化されない)仕様の正確な把握が必要

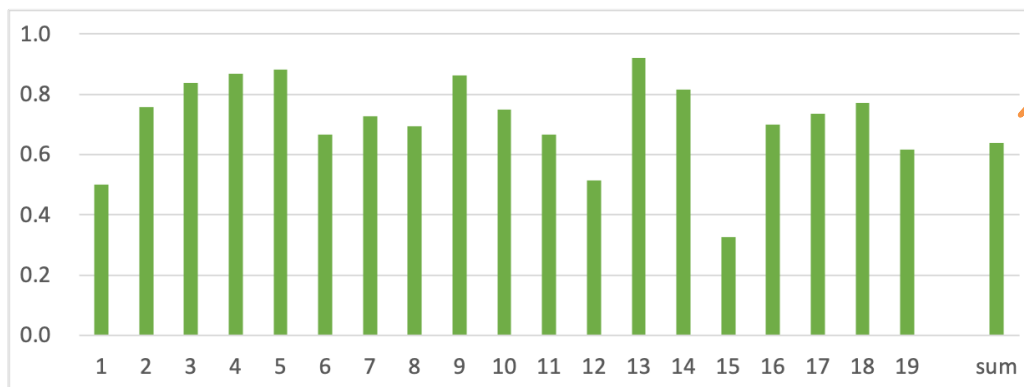
「変更漏れバグ」の現状分析 [修論2020]

- 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**

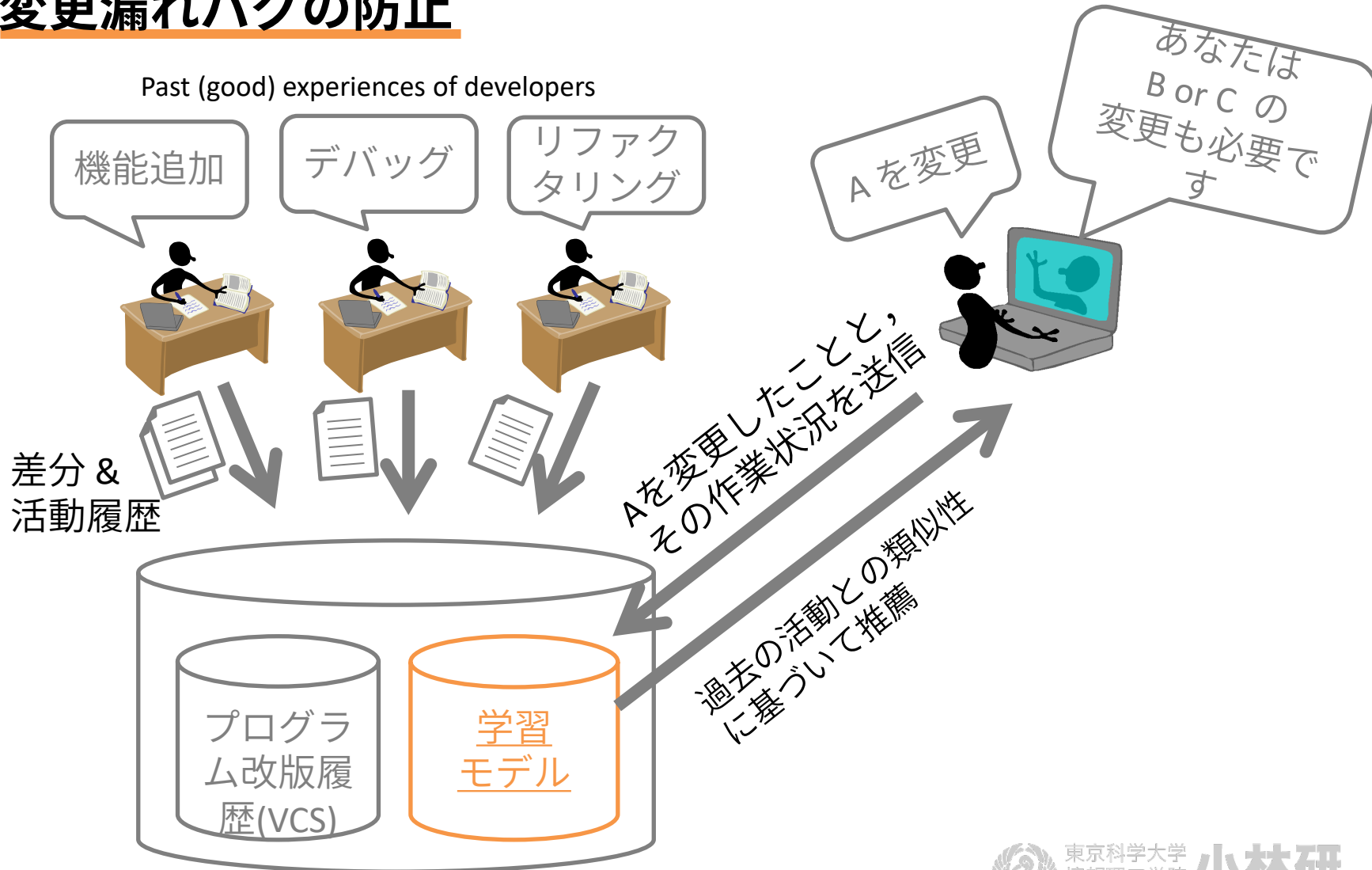
- 改版履歴の分析でどの程度発見できる？

63.9%



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 with ...
+-----+
|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

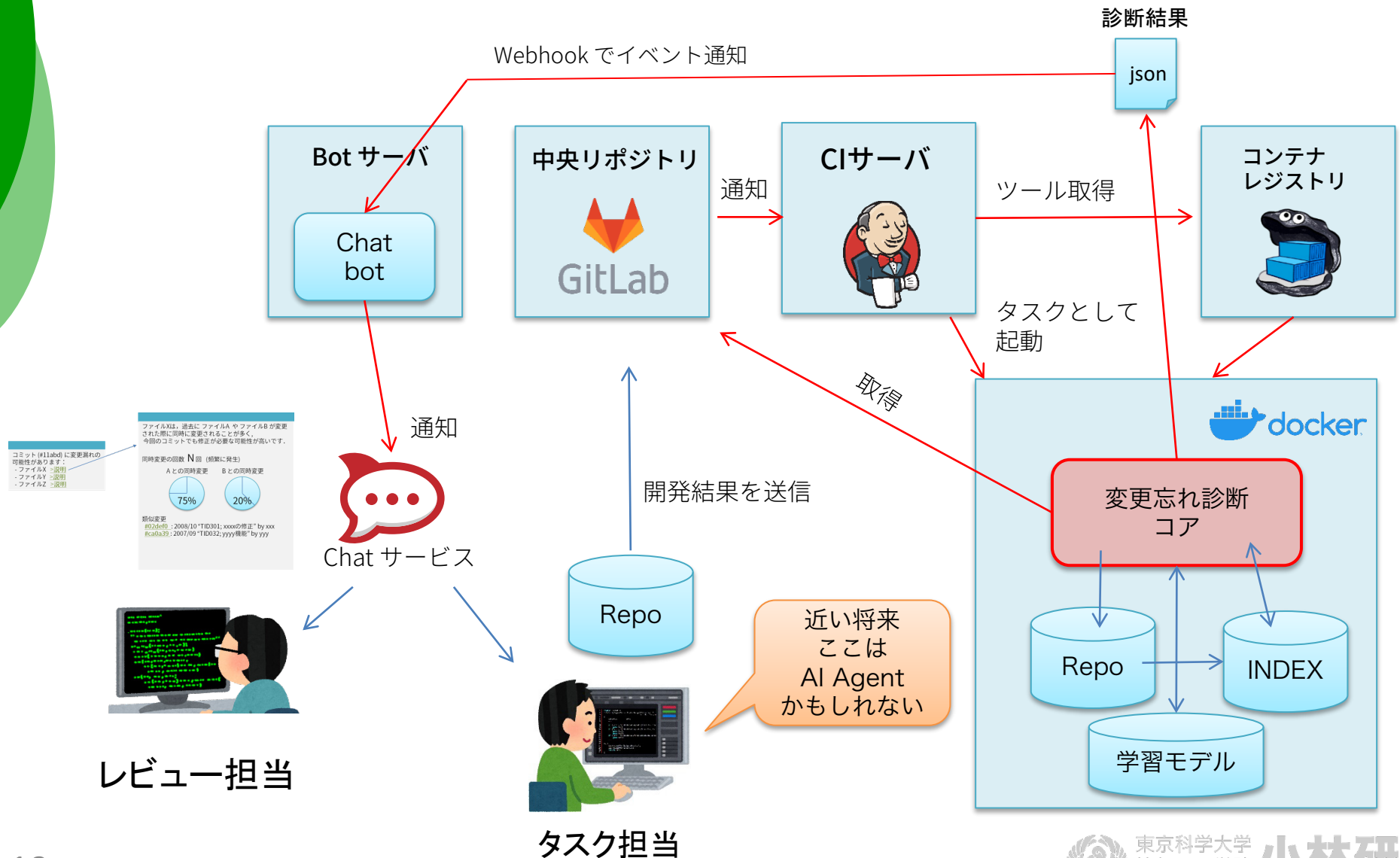
変更忘れ
候補

理由

条件付き確率
66%

過去の
類似変更

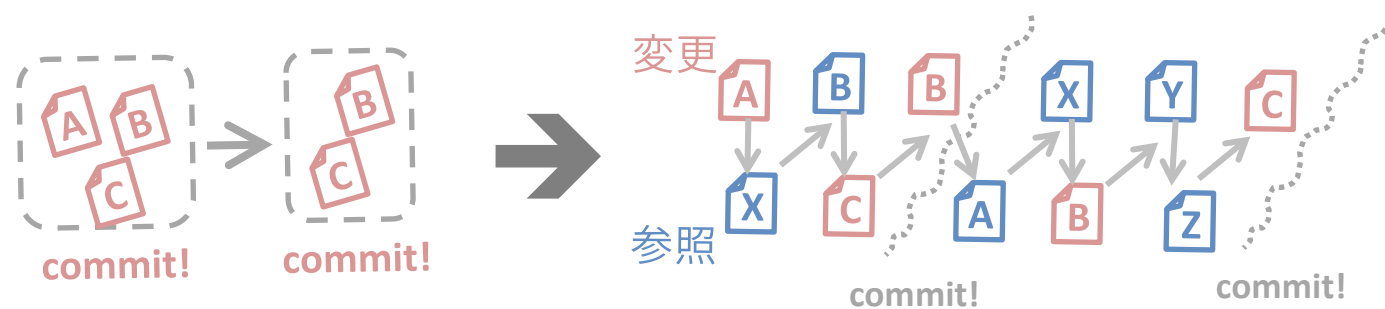
変更支援手法：ChatOps環境への組み込み (共同研究)



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



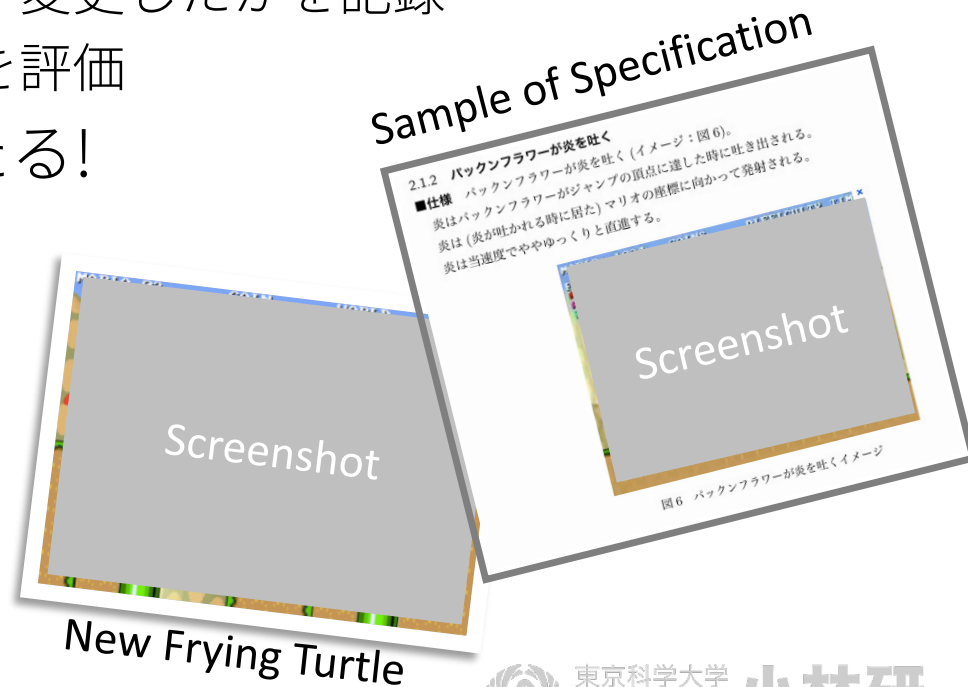
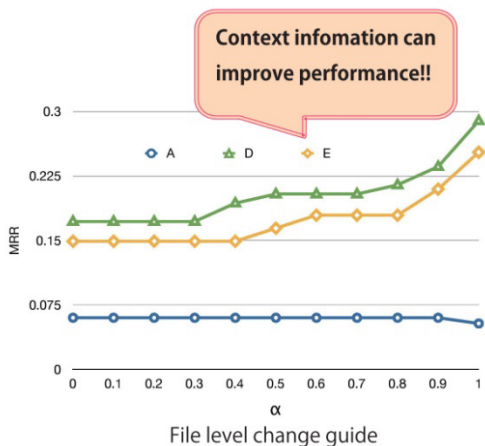
変更履歴 = スナップショット

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 MSR 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，パクショ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！



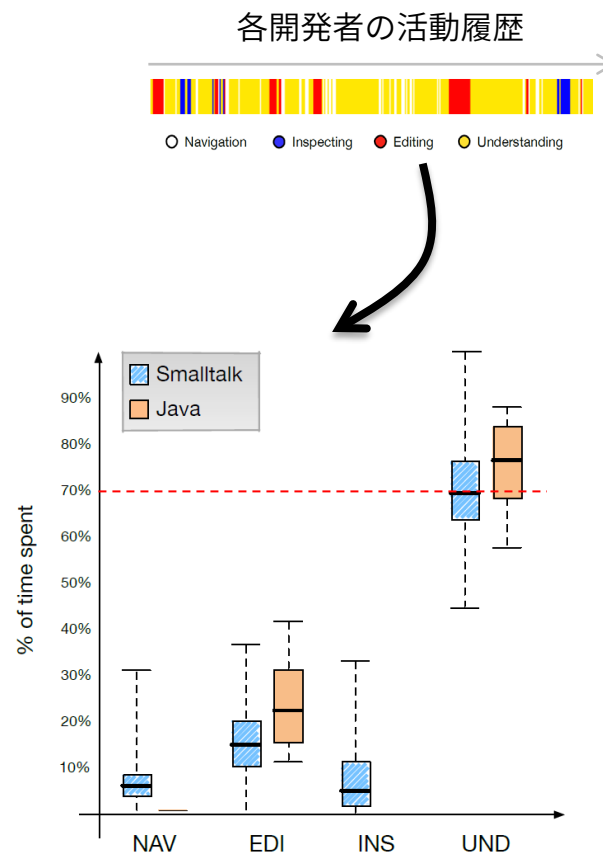
変更支援手法：国際共同研究への発展

○ 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

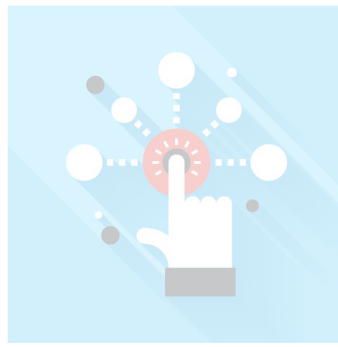


BUG PROTECTION
バグ防止



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

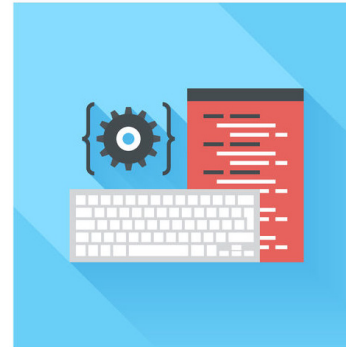


Just-in-Time 変更推薦

開発者の行動を分析し，次に変更すべき箇所を推薦

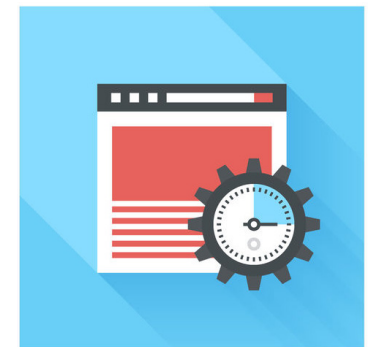


FAULT LOCALIZATION
デバッグ支援



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) 

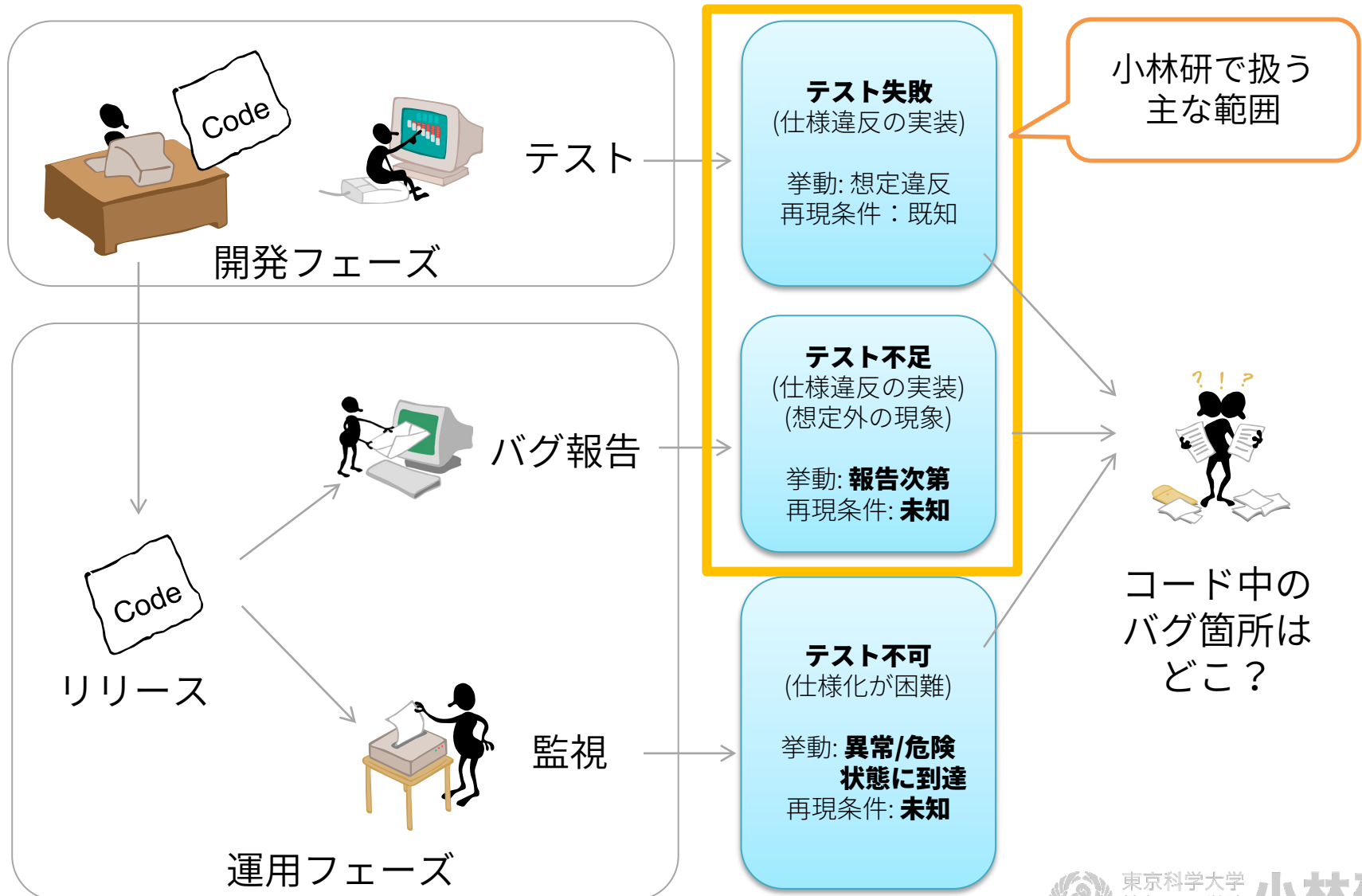
○ 新しいデバッグ環境の開発

- 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

	Test Cases						
	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3	
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);	•	•	•	•	•	•	
	Pass Status: P P P P P F						

どの行が実行されたか

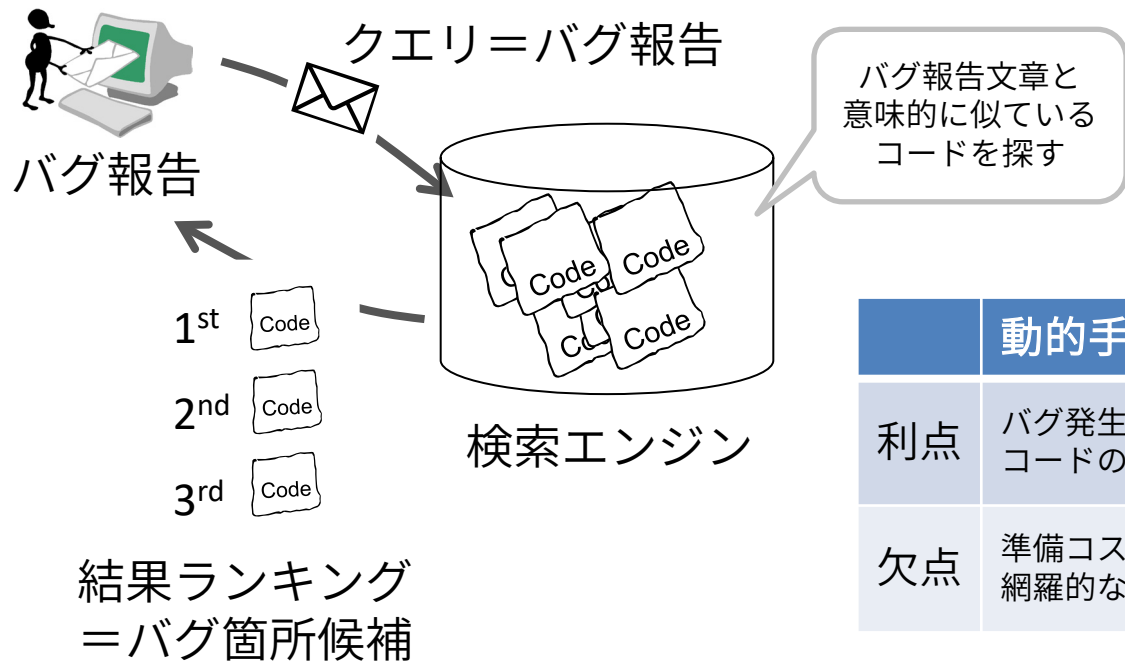
[Jones+2002] Jones et al. "Visualization of test information to assist fault localization" in Proc. ICSE2002

図は <http://pleuma.cc.gatech.edu/aristotle/Tools/tarantula/vizworkshop.ppt> Slide p12 より. 引用

バグ箇所の候補の絞り込み

- 情報検索ベースのバグ局所化手法
 - 「ソースコードの検索問題」とみなす
 - ヒント(クエリ)=バグレポート記述
- 誤検出をどのように減らすかがポイント

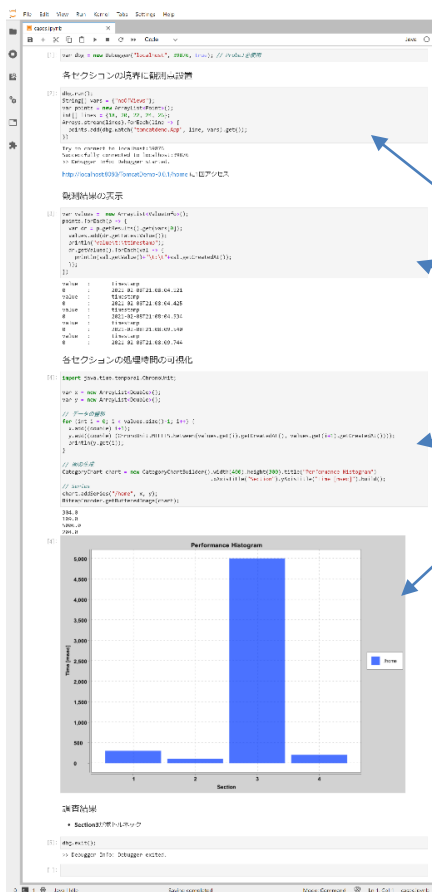
この部分が
研究領域



	動的手法	静的手法
利点	バグ発生に関するコードのみが対象	導入が容易 (準備コストが低い)
欠点	準備コストが高い 網羅的な実行が必要	誤検出が多い

デバッグ支援：新しいデバッグ環境 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

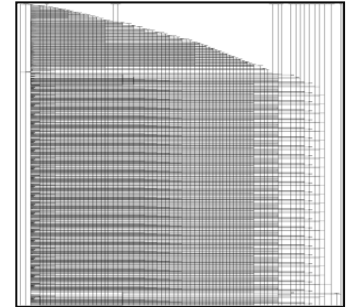


東京科学大学
情報理工学院
小林研
Software Analytics Research Group

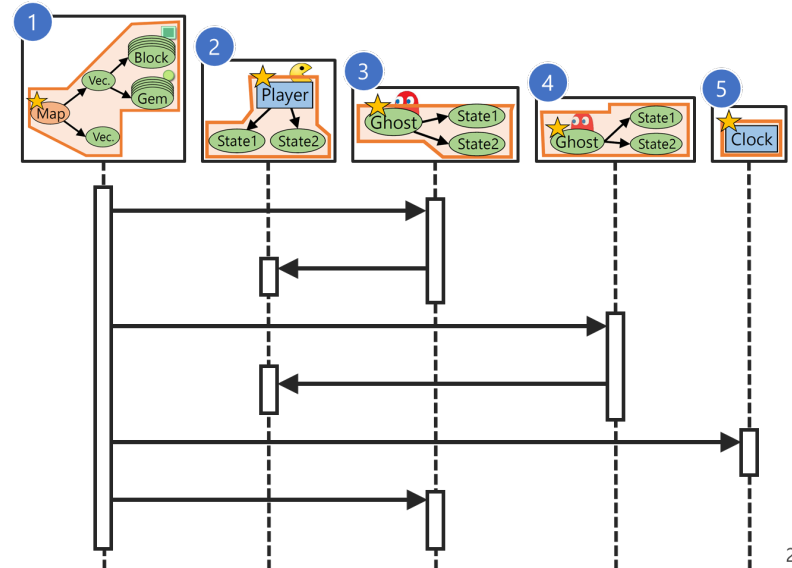
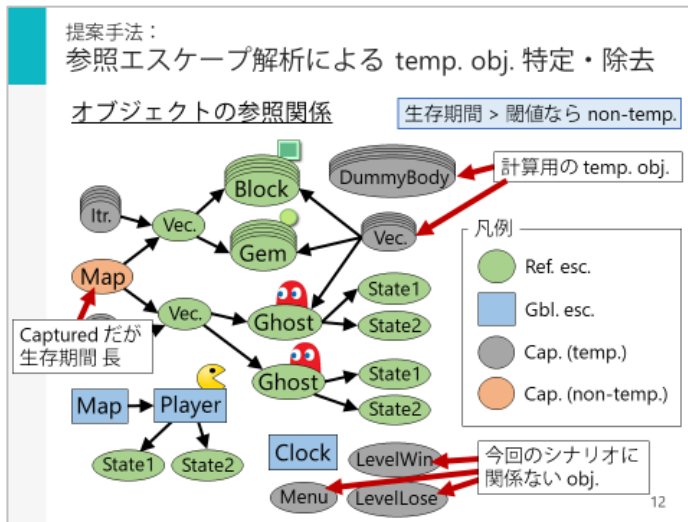
実行履歴のシーケンス図による可視化

○ 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



21

提案手法で生成する実行概要シーケンス図

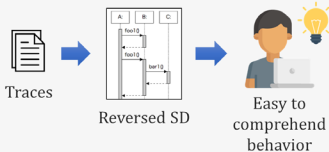
デバッグ支援：プログラム振る舞い理解支援 シーケンス図の活用

○ SDExplore: 大規模SD表示ツールキット

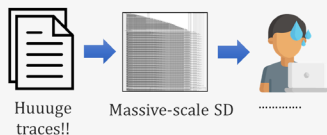
- Won Best Tool Demo Award @ IEEE/ACM ICPC 2018

Program Comprehension with Reversed SD

Ideal



Real



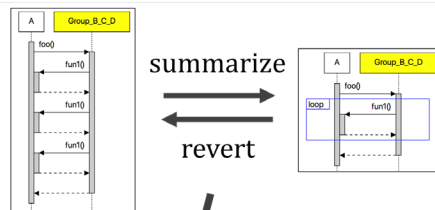
Existing tools...

- Sometimes **Crash**
- **Hard to re-use**
- ... etc.

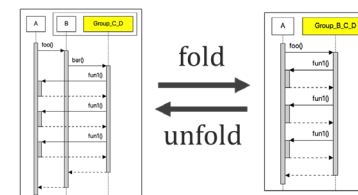
Browser-based tool

- Easy to use
- Platform-free
- Embed to Web page

Interactive Loop Summarization



Interactive group folding/unfolding



Rich features for exploration

- Searching
- Filtering
- Zooming
- ... etc.

Operation Logging

Records interactions to support user activities analysis.

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：ドキュメント生成

- 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・コミットから「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」の生成がトレンド
 - 欲しい情報は多種多様
 - このコミットで何を変更したの？
 - 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？
 - このアプリどんな機能があるの？
 - この機能どこでどうやって実現されているの？

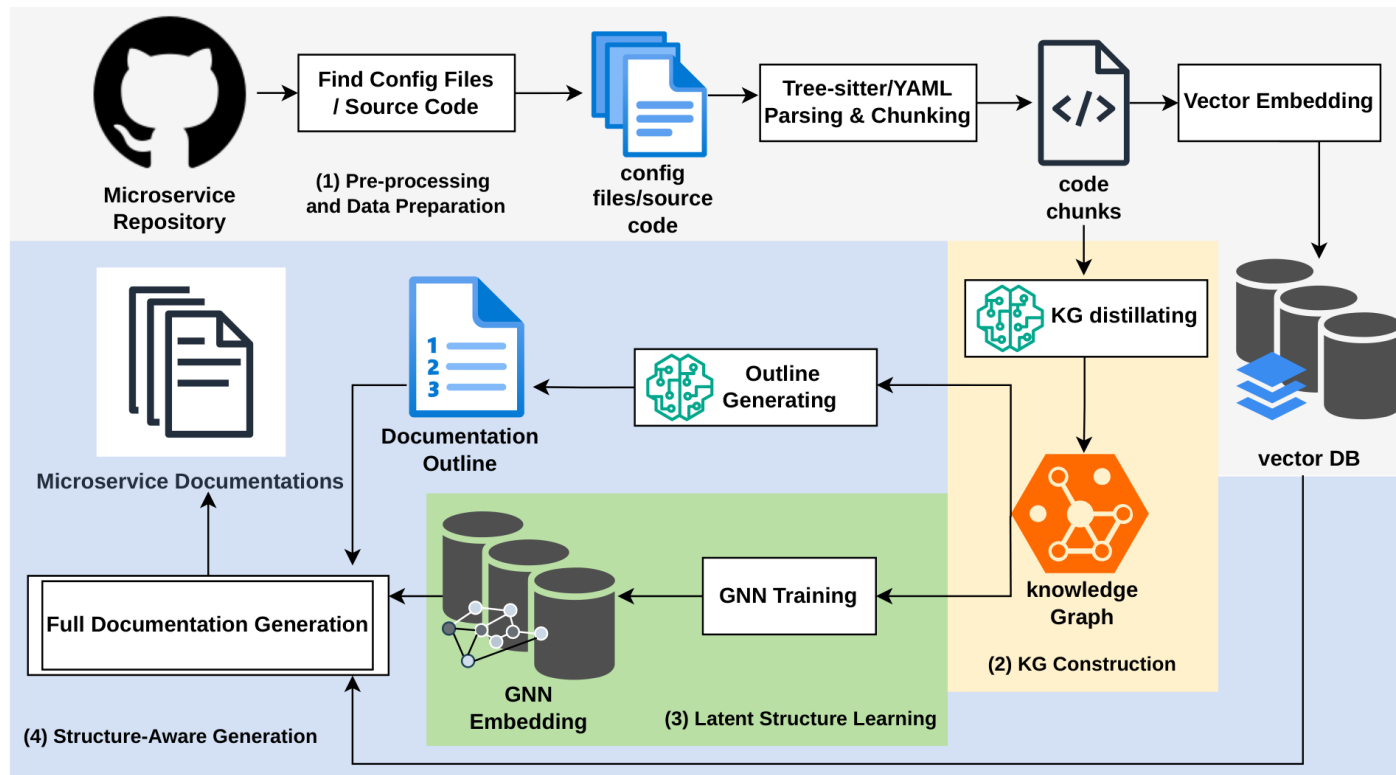
小林研での取り組み

- GraphRAGを用いたドキュメント生成
 - 対象：マイクロサービスアプリケーション(MSA)
 - 特徴：ハイブリッド特徴量に基づく生成
 - MSAの要素間関係グラフ特徴 + 設定内容の意味的特徴
- 概要
 - Step1: ファイル → 分解しLLMで関係抽出 → グラフ化
 - ファイル = ソースコード + 設定ファイル
 - 分解したファイル断片は文書として別途 Embedding
 - Step2: KGを直列化してアウトライン文書を生成
 - Step3: 文書断片Embeddingを使って詳細内容を補完
 - グラフEmbeddingの情報で RAGでの検索精度を改善

GraphRAGを用いたドキュメント生成

○ 特徴：

- セクションごとの文書生成
- GNNベクタで文書ベクタのランキングを改善

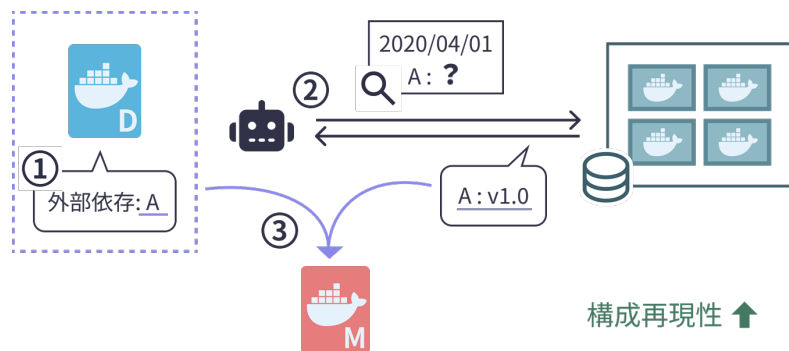
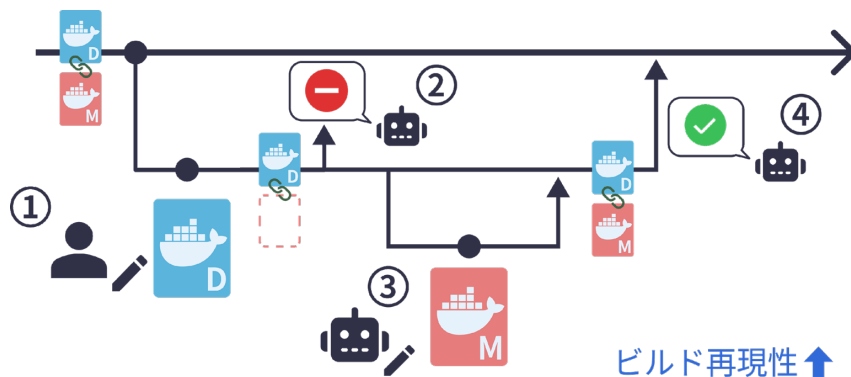


これまでの取り組み：SE 4 IaC

- 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
 - ...

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



ベースイメージをubuntu:24.04に変更 #14

Open oribe1115 wants to merge 1 commit into main from demo-new-pr

Conversation 1 Commits 1 Checks 1 Files changed 1

oribe1115 commented 2 minutes ago

No description provided.

ベースイメージをubuntu: 24.04に変更

github-actions bot commented 1 minute ago

Merge #15 to this branch to update Dockerfile

github-actions bot mentioned 1 minute ago

Update Dockerfile.mold with for

Add more commits by pushing to the demo-n

All checks have failed

1 failing check

github/workflows/demo.yaml / forge-action-job (pull_request) Failing af... (Required) Details

Required statuses must pass before merging

All required statuses and check runs on this pull request must run successfully to enable automatic merging.

GitHub Actions 用の CI Actionとして実装

Dockerfile内の不十分な記述箇所を修正案を自動提示

2026年度メンバー

○ 学生は13名前後

- 教員：小林，Partachi助教
- D：3名
 - 要求文書生成，コミット分割，不足変更箇所推薦
- M2：3名
 - デバッガ/IDE拡張，ドキュメント生成，バグ箇所特定
- M1：4名
 - ドキュメント生成，テストケース生成，未定 x 2
- 学部：1-3名
- 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
入学は例年半数程度

○ 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 4社

必要な知識・関連講義など

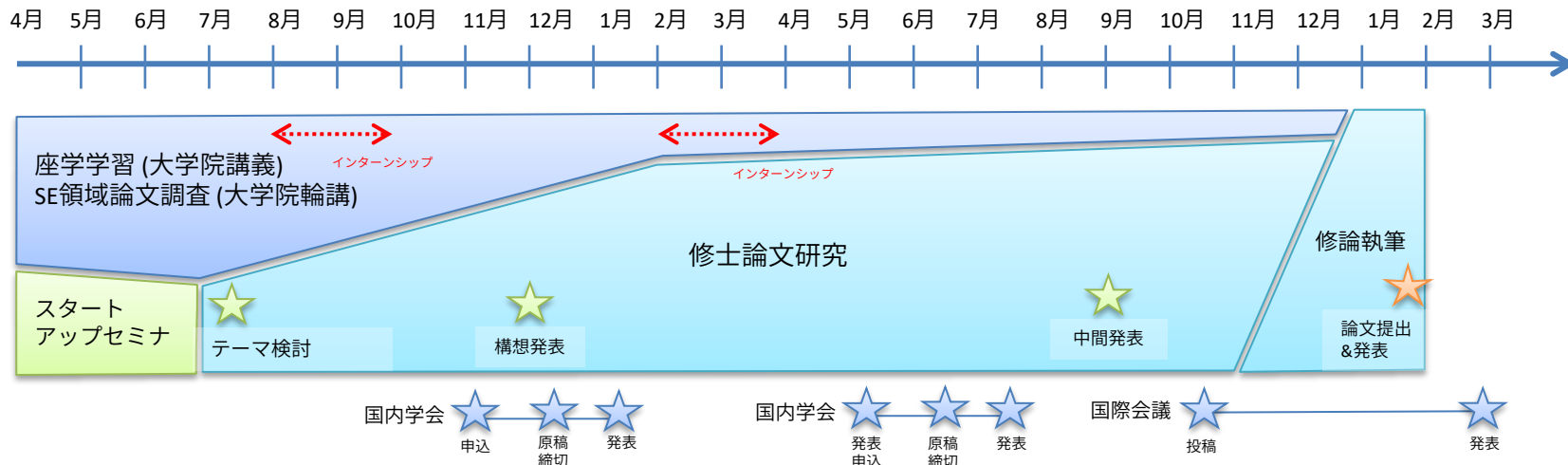
- 必須知識ではありませんが以下は関連します
 - 手続き型・オブジェクトプログラミング言語
 - 数理論理学, オートマトン, コンパイラ
 - データベース, 情報検索, データマイニング, 機械学習
- ソフトウェア開発・プログラミングで困ったことがある人はすごく向いています
 - その内容が研究のきっかけです
 - 開発環境に不満 → 支援ツールは良くある話です
 - 開発者を楽に・幸せにすること = SEのゴールの1つ
- 開発スキルを得たい人も向いています
 - その動機に関連する書籍・事例はたくさん提供できます

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

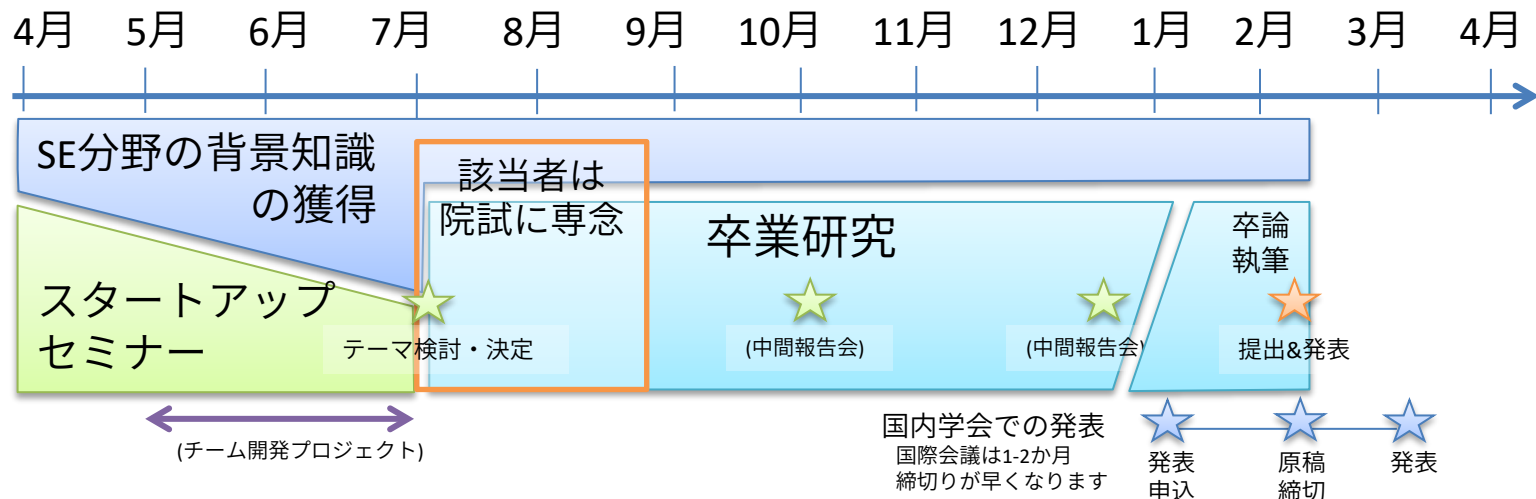


○ 基本スケジュール

● 週3日間のコアタイム制をとっています

- 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

特課研究生のスケジュール



○ 基本スケジュール

- **週3日間のコアタイム制をとってます**
- 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
- 後期の輪講は洋書輪読か論文紹介
- 8月~9月中旬程度までは輪講・研究会はなし(≠夏休み)
 - 上の図の通り, 9月一杯「休む」と研究は2か月程度しかできない
- 後期から研究会で毎週進捗を確認

小林研の「特課研」に対する考え

- 「研究」に関する基礎能力の向上が主目的
 - 文献調査・実験（計画と評価方法）・論文執筆・発表
→ テクニカルライティング，ロジカルシンキング，プレゼンテーション
 - 上記能力に関連するスタートアップセミナーを予定
 - 座学→体験による定着を行いたい
 - 卒研発表会だけでなく，研究成果の学外発表を推奨
 - 当然，研究内容はおろそかにしません
 - 多くの卒論生は3月～修士在学中に学会発表
 - 国際会議で発表・学会で受賞したケースも少なくない
- SE分野，特に設計～保守に関する専門知識の獲得
 - 実質4か月程度しかないが，背景知識の獲得は目標としたい。
 - 研究指導，関連論文・書籍の提供を通じた学習がメイン



東京科学大学
情報理工学院 **小林研**
Software Analytics Research Group

2022年から **すす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 **すすかけ台 駅**

問合わせ先：

小林隆志 <tkobaya@comp.isct.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.comp.isct.ac.jp/>