バグチケットの機械学習から見抜く検証コスト削減ポイントとソースコードの改善ポイント

国分佑樹, 喜久里 陽
株式会社ディー・エヌ・エー

© DeNA Co.,Ltd.

00

この発表で伝えたいこと

© DeNA Co.,Ltd.

バグチケットの記入項目に修正差分への URL
を足すと検証項目のリスクを可視化できる。

そのための基盤の実現方法と機械学習を
用いた分析方法について解説する。

© DeNA Co.,Ltd.

目次

1 解決したい問題

2 考案した手法

3 データの収集手法

4 機械学習による分析

5 展望

© DeNA Co.,Ltd.

01

解決したい問題

本事例で解決したい問題を説明します

© DeNA Co.,Ltd.

1 解決したい問題

運用中のゲームアプリケーションにおいて高い開発コスト（検証コストを含む）が弊社の課題となっている。

一般に開発コストを低く抑える手法として品質の作り込みが重要と言われている。

しかし新規開発中のアプリケーションと比べると、運用中のアプリケーションで品質の作り込みを後から始めるのは難しい。

このような品質の作り込みの難しい状況でも開発コストを低くしたい。

© DeNA Co.,Ltd.

02

考案した手法

本事例で発表する手法について説明します

© DeNA Co.,Ltd.

2 方針

開発コストのうち検証コストの占める割合が大きいため
検証コストに着目した。検証コストを下げるためには
検証項目を減らせばよい。ただし闇雲に検証項目を
減らすと見逃される欠陥が増えてしまう。

減らす検証項目を欠陥の摘出可能性の低いものに
限定できれば、見逃される欠陥を大きくは増やさずに
検証項目を減らせると考えた。

そのため検証項目の欠陥摘出可能性の評価にフォーカスした。

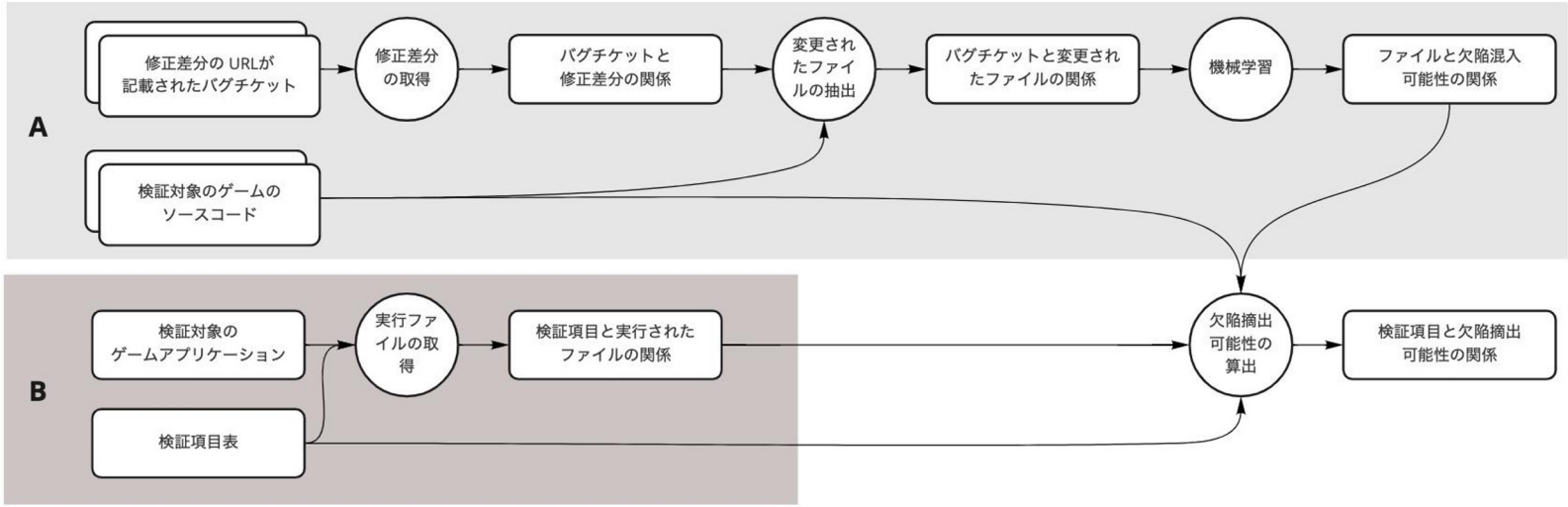
© DeNA Co.,Ltd.

2 手法の全体像

本事例では検証項目の欠陥摘出可能性を2段階で評価した：

A. ファイルごとの欠陥混入可能性を評価する

B. 検証項目の実施時に実行されるファイルを推測する



この2つを突き合わせると、検証項目の実施時に実行される

ファイルから欠陥混入可能性を得られる

© DeNA Co.,Ltd.

2 手法の全体像

ここで次のような検証項目を考える：

- 実行されるファイルの欠陥混入可能性が大きい検証項目
- 実行されるファイルの欠陥混入可能性が小さい検証項目

欠陥混入可能性の小さい方が大きい方より欠陥摘出可能性は小さいと単純化すると、欠陥混入可能性を検証項目の欠陥摘出可能性の指標として使えると考えた。

© DeNA Co.,Ltd.

03

データの収集手法

データの収集手法について説明します

© DeNA Co.,Ltd.

3 取得するデータの全体像

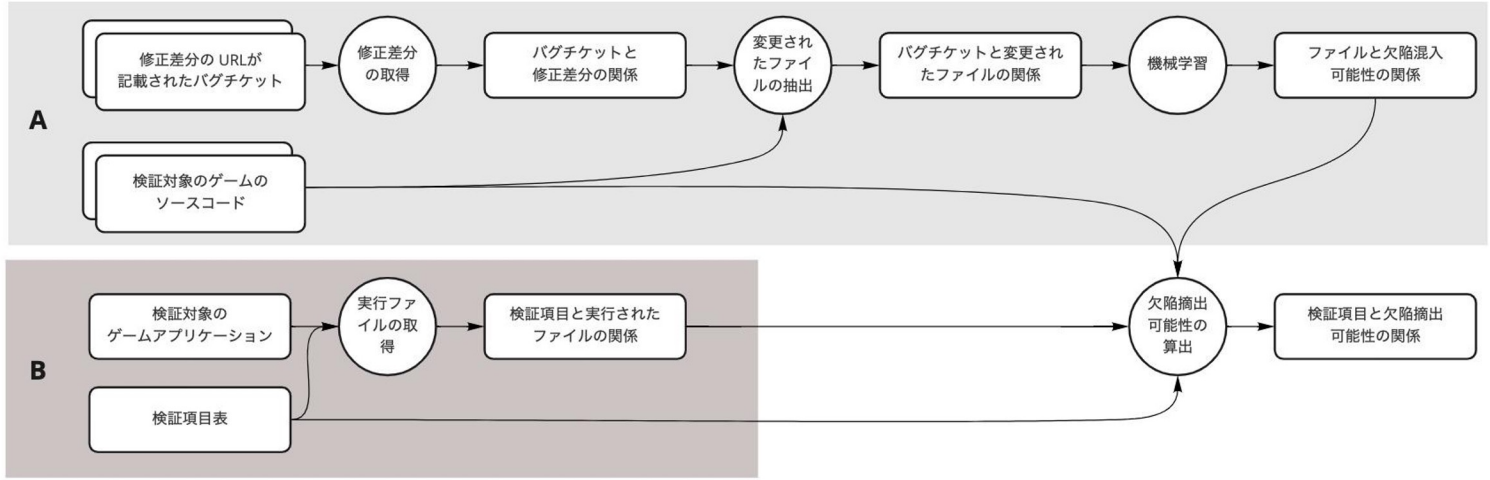
本事例で取得したデータは大まかに分けて2つ：

A. ファイルの欠陥混入可能性を評価するためのデータ

- Jira 上の修正差分への URL が付属したバグチケット
- GitHub 上の修正差分

B. 検証項目で実行されるファイルのデータ

- 検証項目表
- 検証項目ごとに実施時に実行されたファイル



© DeNA Co.,Ltd.

3 取得するデータの全体像

本事例で取得したデータは大まかに分けて2つ：

A. ファイルの欠陥混入可能性を評価するためのデータ

- Jira 上の修正差分への URL が付属したバグチケット
- GitHub 上の修正差分

B. 検証項目で実行されるファイルのデータ

- 検証項目表
- 検証項目ごとに実施時に実行されたファイル

© DeNA Co.,Ltd.

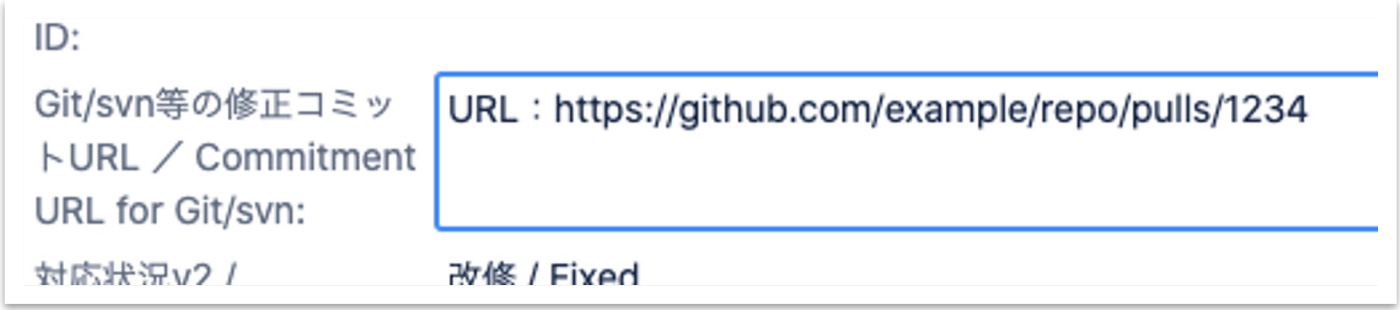
3 ファイルの欠陥混入可能性のデータ

バグチケットに GitHub 上の修正差分への URL を記入してもらった：

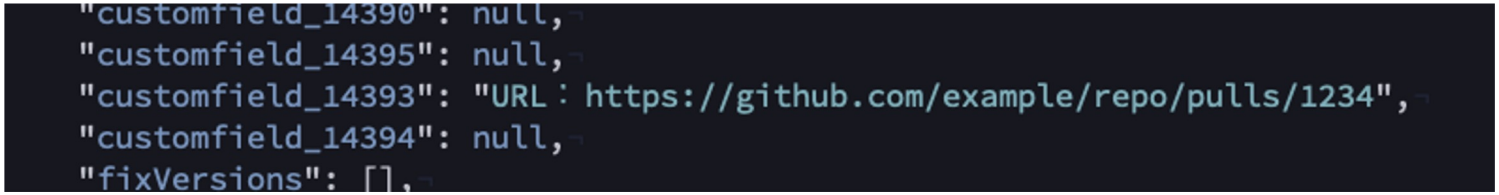
© DeNA Co.,Ltd.

3 ファイルの欠陥混入可能性のデータ

バグチケットには修正差分への URL が記載されている：



すべてのバグチケットは Jira API から取得できる：



© DeNA Co.,Ltd.

3 ファイルの欠陥混入可能性のデータ

修正差分への URL は三種類ある：

- Commit
  - 変更差分[1]
  - 例: https://github.com/example/repo/commits/1234abcd
- Pull request
  - 複数の commit をわかりやすい単位にまとめたもの
  - 例: https://github.com/example/repo/pulls/123
- Compare
  - 特定の範囲の commit を切り出したもの
  - 例: https://github.com/example/repo/compare/1234...5678

[1] 厳密にいうと commit は変更差分ではなくスナップショットであるが、直前の commit と比較すると一意な変更差分を得られるのでここでは変更差分と表記した。直前の commit がない場合でも何もファイルがないとみなせば同様に一意な変更差分を得られる

© DeNA Co.,Ltd.

3 ファイルの欠陥混入可能性のデータ

GitHub API でそれぞれの URL に紐づく commit の ID の一覧を取得する：

- Commit
  - https://github.com/example/repo/commit/<SHA1>
    → <SHA1> 部分から commit の ID を得る
- Pull request
  - https://github.com/example/repo/pulls/<数字>
    → https://api.github.com/repos/example/repo/pulls/<数字>/commits
    のレスポンスから commit の ID 一覧を得る
- Compare
  - https://github.com/example/repo/compare/<SHA1>…<SHA1>
    → https://api.github.com/repos/example/repo/compare/1/commits
    のレスポンスから commit の ID 一覧を得る

© DeNA Co.,Ltd.

3 ファイルの欠陥混入可能性のデータ

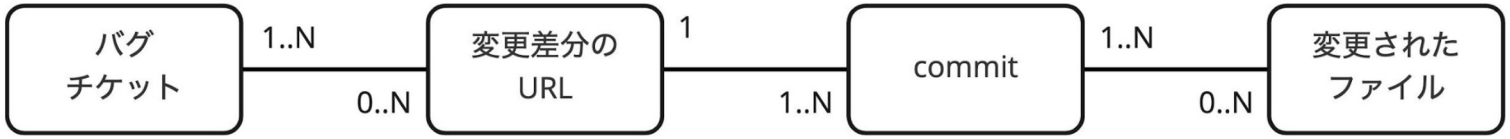
Commit の ID から変更されたファイルの一覧を得る。バージョン管理ツール git に commit の ID を与えると変更されたファイルと変更内容を得られる：

```
$ git show a3fef3bc3be41a965b8ea1c25a0acf94ee783668
commit a3fef3bc3be41a965b8ea1c25a0acf94ee783668 (HEAD -> master)
Author: Kuniwak <yuki.kokubun@dena.com>
Date:   Thu Dec 23 10:11:50 2021 +0900

    after

diff --git a/example.txt b/example.txt
index 60344e3..b9e0880 100644
--- a/example.txt
+++ b/example.txt
@@ -1,7 +1,6 @@
 Lorem ipsum dolor sit amet, consectetur adipiscing elit,
 sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
-Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris
-nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in
-reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
-Excepteur sint occaecat cupidatat non proident, sunt in culpa qui
-officia deserunt mollit anim id est laborum.
+Mauris pharetra et ultrices neque ornare aenean euismod elementum.
+Ultrices gravida dictum fusce ut placerat orci. Condimentum id
+venenatis a condimentum. Ut placerat orci nulla pellentesque dignissim.
+Consectetur libero id faucibus nisl. Sem integer vitae justo eget magna.
```

© DeNA Co.,Ltd.

3 ファイルの欠陥混入可能性のデータ

これまでのデータからバグチケットと修正によって変更されたファイルの対応関係が得られた：

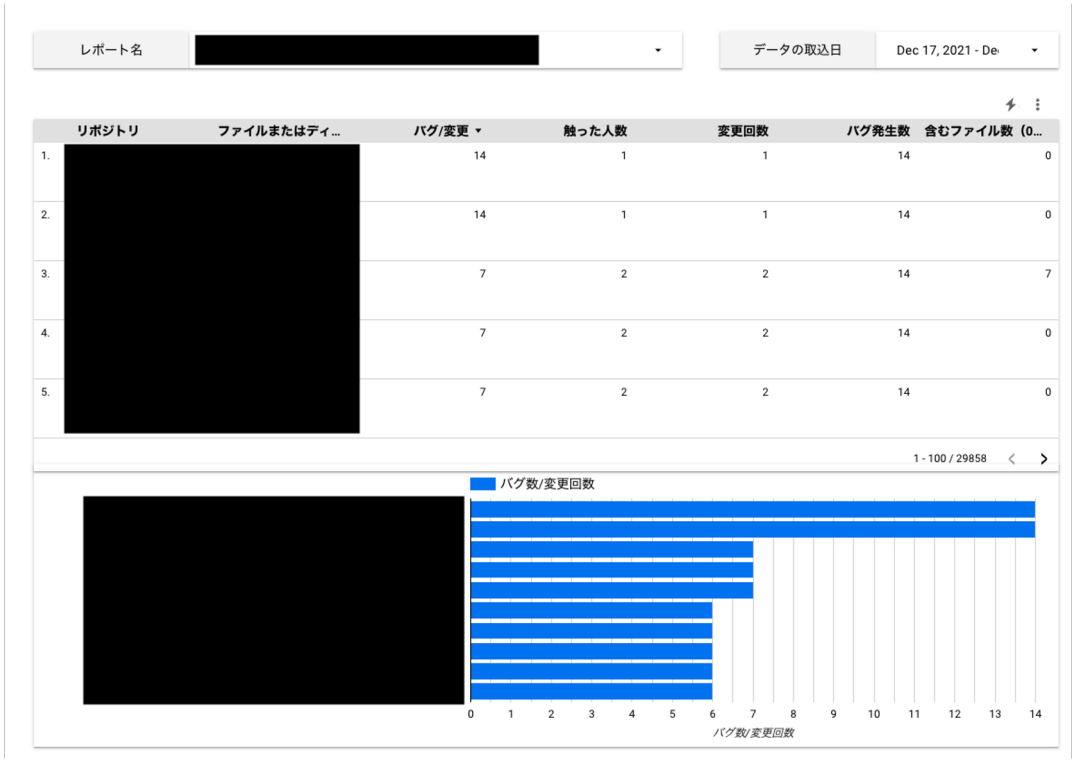


欠陥を修正するために変更されたファイルを欠陥があったファイルとみなせば、ファイルに紐づく欠陥がわかったことになる。
ここから機械学習によって欠陥混入可能性を推測させた（後述）。

© DeNA Co.,Ltd.

3 ソースコードの改善ポイントの共有

このファイルごとの欠陥混入数をファイルに対する変更数で除算したスコアの上位のファイルは、危険なファイルとして開発チームへフィードバックしている：

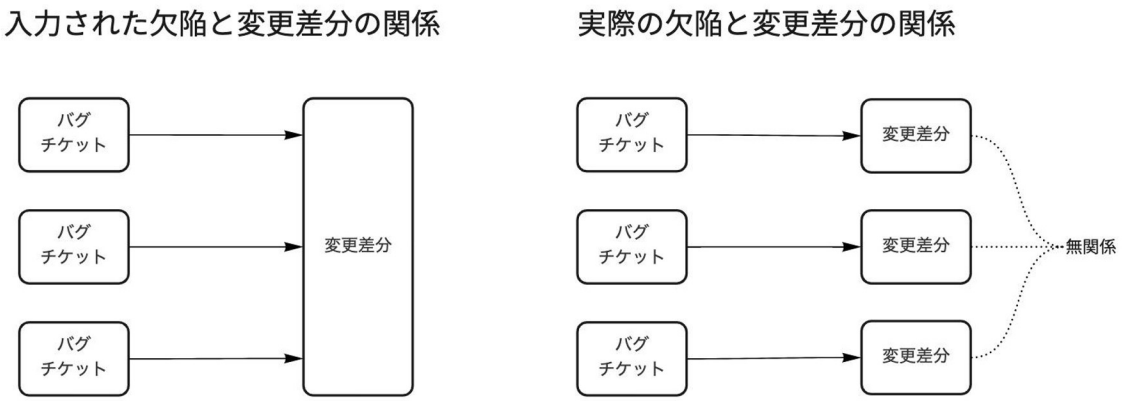


© DeNA Co.,Ltd.

3 得られたデータの信頼性

定性的ではあるが、欠陥混入数が多かったファイルは開発者の経験と照らして危険なファイルであったことを確認した。そのためそれなりの信頼性があると判断している。

ただし修正箇所の推測を誤らせる入力も観察された：



実際の運用ではこのような入力を許容しないルール設定が必要になるだろう。

© DeNA Co.,Ltd.

3 取得するデータの全体像

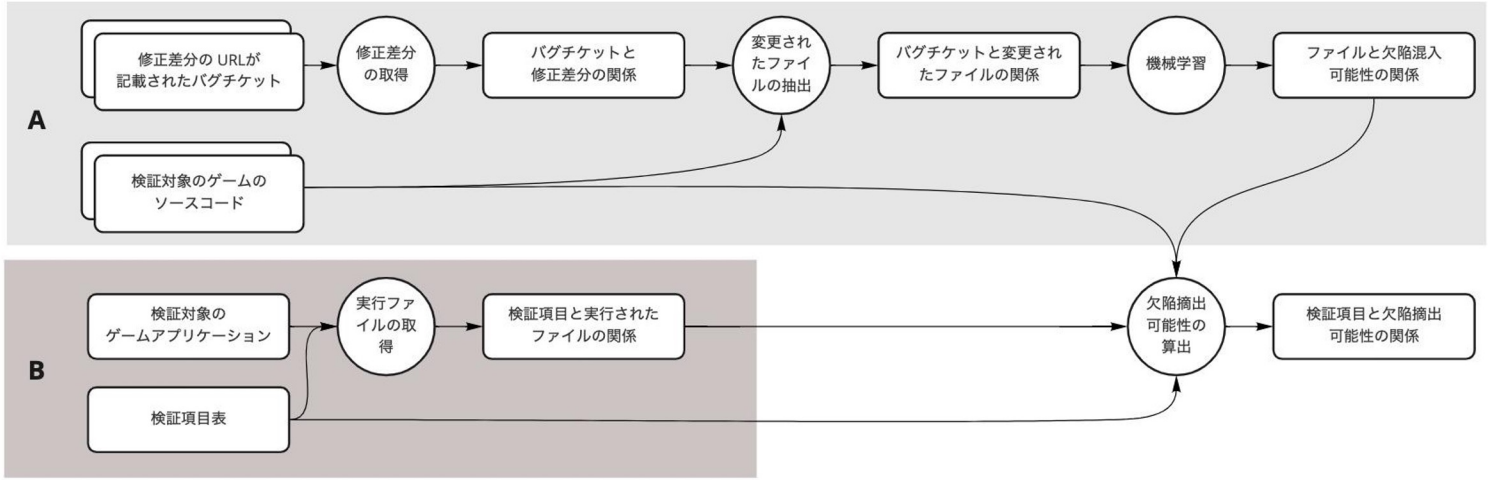
本事例で取得したデータは大まかに分けて2つ：

A. ファイルの欠陥混入可能性を評価するためのデータ

- Jira 上の修正差分への URL が付属したバグチケット
- GitHub 上の修正差分

B. 検証項目で実行されるファイルのデータ

- 検証項目表
- 検証項目ごとに実施時に実行されたファイル



© DeNA Co.,Ltd.

3 取得するデータの全体像

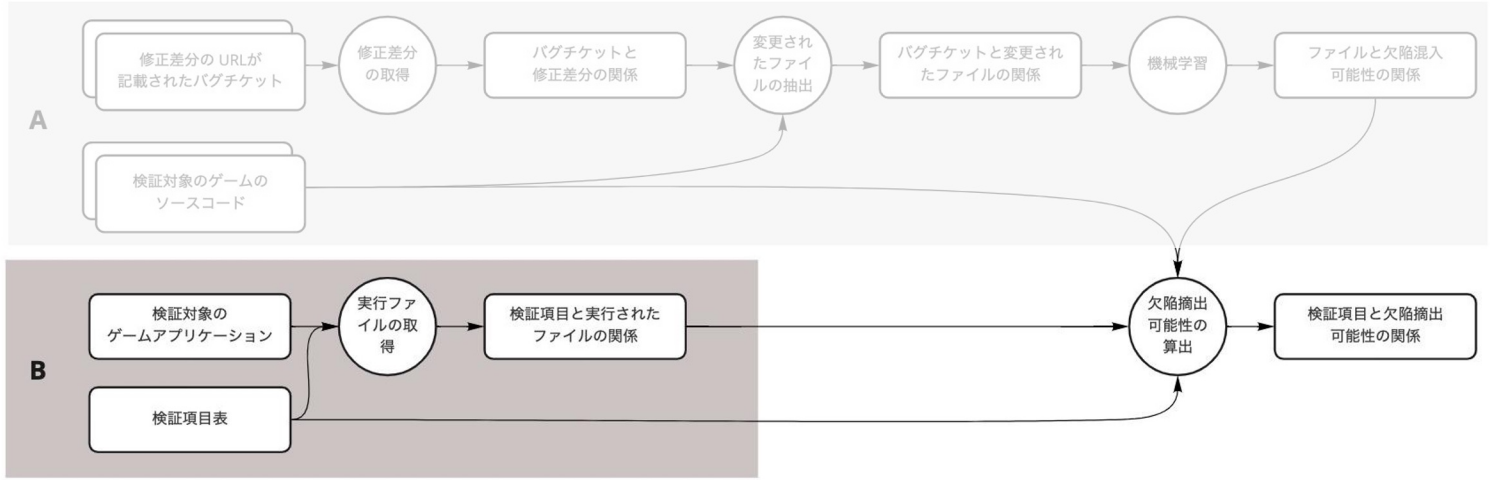
本事例で取得したデータは大まかに分けて2つ：

A. ファイルの欠陥混入可能性を評価するためのデータ

- Jira 上の修正差分への URL が付属したバグチケット
- GitHub 上の修正差分

B. 検証項目で実行されるファイルのデータ

- 検証項目表
- 検証項目ごとに実施時に実行されたファイル



© DeNA Co.,Ltd.

3 検証項目で実行されるファイルのデータ

これまでのデータでリスクの高いファイルがわかっている。ここで次のような検証項目を考える：

- 実行されるファイルの欠陥混入可能性が大きい検証項目
- 実行されるファイルの欠陥混入可能性が小さい検証項目

前者より後者の方が欠陥の摘出可能性が小さいと考えるのが自然。

そこで検証項目の実施時に実行されるファイルの欠陥混入可能性を検証項目の欠陥摘出可能性の指標として利用する。

そのため、検証項目ごとに検証時に実行されるファイルを記録したい。

© DeNA Co.,Ltd.

3 検証項目で実行されるファイルのデータ

弊社で主に用いているゲームエンジン Unity では Code Coverage パッケージ[5] を使って手動実行時に実行された関数を得られる。

本事例では Code Coverage パッケージを使うために、
実環境であるモバイル端末ではなく Unity エディタ（開発環境）上で計測した。

そのためここで得られたデータは実環境で実際に
実行される関数と正確に一致しないことがある[6] ので注意。

[5] https://docs.unity3d.com/Packages/com.unity.testtools.codecoverage@latest/manual/index.html

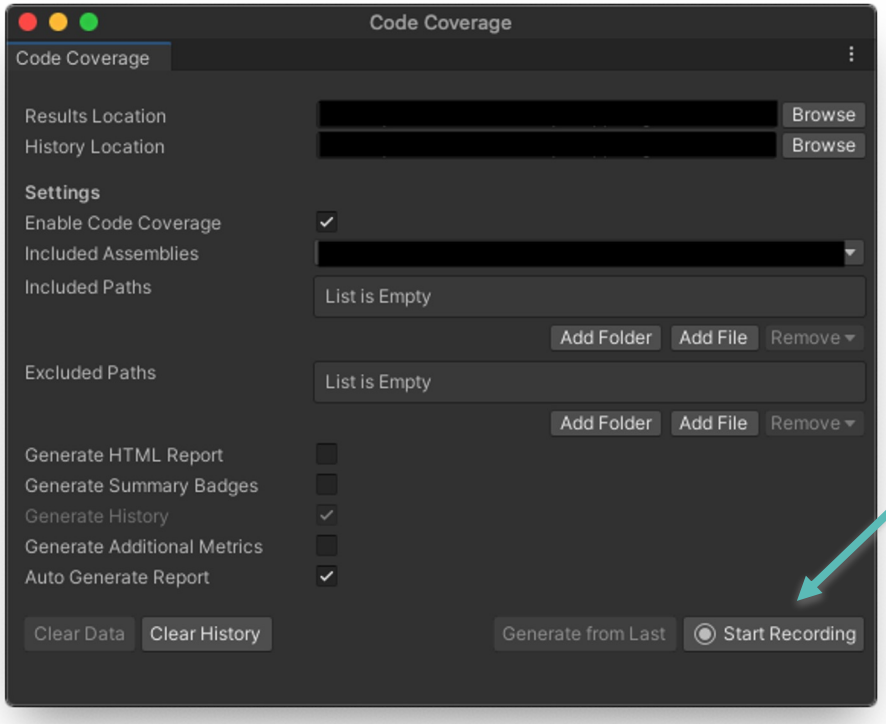
[6] #if !UNITY_EDITOR のようなプリプロセッサディレクティブで囲まれたコードは実行されない

© DeNA Co.,Ltd.

3 検証項目で実行されるファイルのデータ

Code Coverage パッケージをインストールした Unity プロジェクトでは

Window > Analysis > Code Coverage で以下の window を開ける：



記録開始ボタンを押すと、

記録終了ボタンが出現する

記録終了すると、指定した位置に

XML ファイルが出力される

© DeNA Co.,Ltd.

3 検証項目で実行されるファイルのデータ

Code Coverage パッケージが出力する XML ファイルには
メソッドごとの実行状況が visited 属性の値に記録されている。

XML ファイルの中から visited属性が true な <Method> 要素を探すと
この子要素に <FileRef> 要素が見つかる。

この uid 属性の値と同じ uid 属性の値を持つ <File> 要素を探して
実行されたファイルのファイルパスを得る。

この手順で検証項目の実施時に実行されたファイルを得られる。

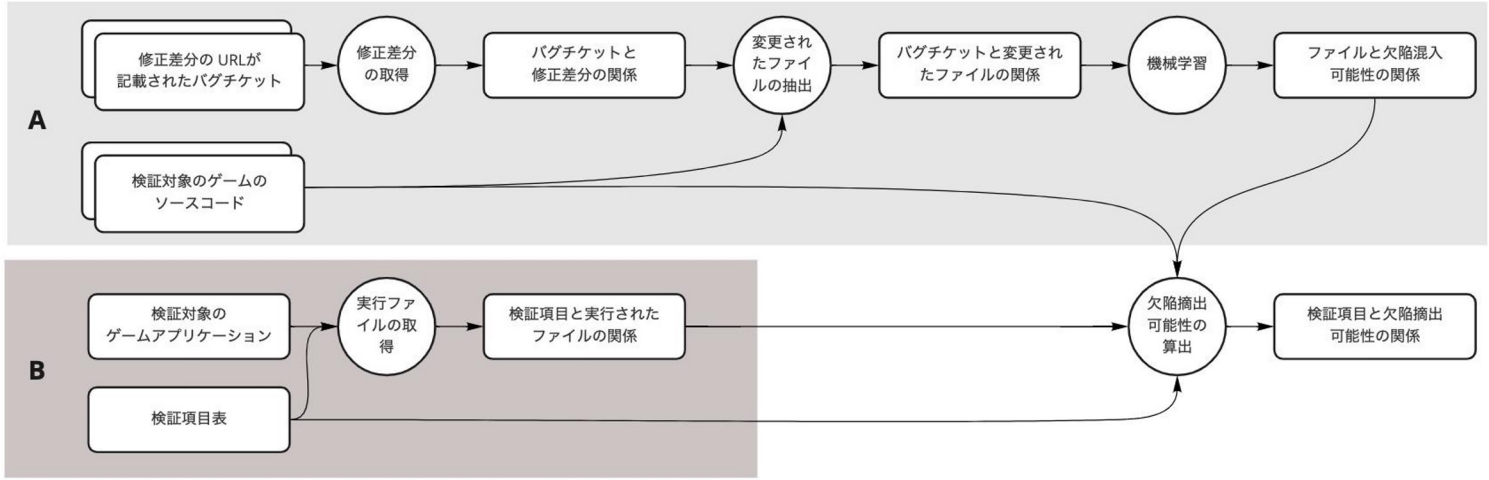
© DeNA Co.,Ltd.

3 取得するデータのまとめ

本事例で取得したデータは大まかに分けて2つ：

A. ファイルの欠陥混入可能性を評価するためのデータ
- Jira 上の修正差分への URL が付属したバグチケット
- GitHub 上の修正差分

B. 検証項目で実行されるファイルのデータ
- 検証項目表
- 検証項目ごとに実施時に実行されたファイル



© DeNA Co.,Ltd.

04

機械学習による推定と分析

機械学習による推定と分析の説明をします

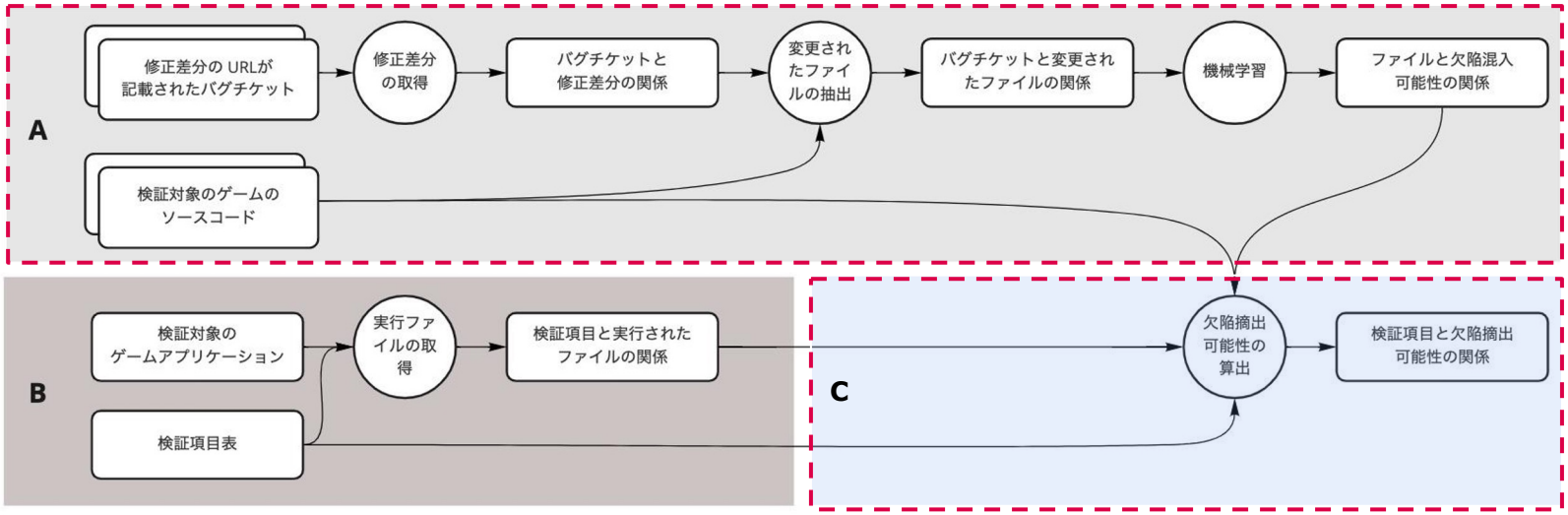
© DeNA Co.,Ltd.

4 推定・分析の流れ

これ以降、2種類の欠陥混入可能性の推定や分析について説明する

- ファイルの欠陥混入可能性
- 検証項目の欠陥混入可能性

ファイルの欠陥混入可能性の評価



検証項目と紐づくファイルを推定

検証項目の欠陥混入可能性の評価

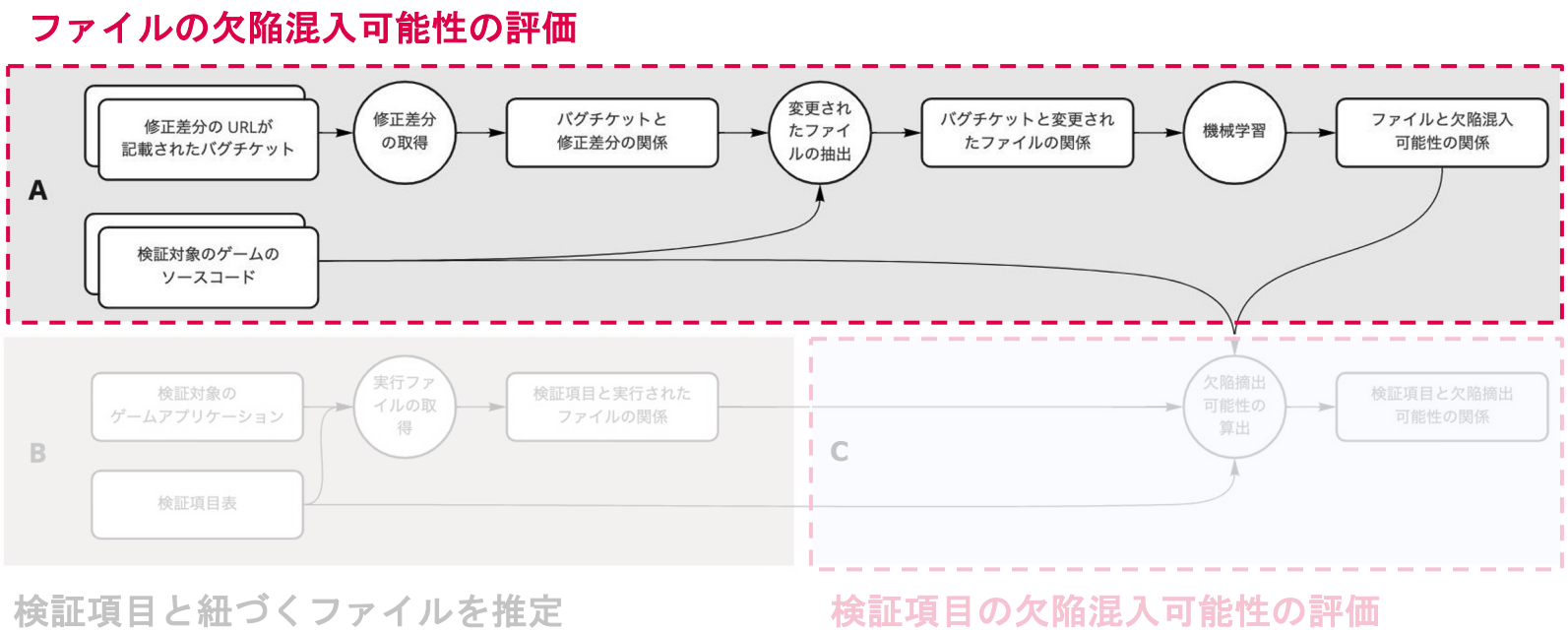
ファイルの欠陥混入可能性	運用中ゲームサービスのうち、変更履歴にアクセス可能なレポジトリを対象に推定・分析（15レポジトリ）
検証項目の欠陥混入可能性	検証項目のトレースを実施可能なサービスのレポジトリに絞って推定（1レポジトリ）

© DeNA Co.,Ltd.

4 ファイルの欠陥混入可能性 - 推定の流れ

ファイルの欠陥混入可能性は、2種類のデータを元に機械学習モデルを使って推定している

- 修正差分と紐付けられたバグチケット一覧
- 修正差分の内容とその履歴



© DeNA Co.,Ltd.

4 ファイルの欠陥混入可能性 - 推定の流れ

ファイルの欠陥混入可能性は、2種類のデータを元に機械学習モデルを使って推定している

- 修正差分と紐付けられたバグチケット一覧
- 修正差分の内容とその履歴

バグチケット一覧
・欠陥混入の有無を表すデータ
・これを推定対象(目的変数)とする


```
"customfield_14390": null,
"customfield_14395": null,
"customfield_14393": "URL : https://github.com/ex
"customfield_14394": null,
"fixVersions": [],
```


ファイル一覧
・更新状況の統計値を算出
・特徴量として機械学習モデルに入力


```
"file_stat_map": {
  "\r": {
    "num_authors": 5,
    "num_files_containing": 0,
    "commits": [
      "4cf225cf5cf13882f94ae533cee05fb18641458d",
      "9323a2403ec9c8fb5b3371298ca0e43b371a4406",
```


修正差分URL, commit ID で紐付け

機械学習モデルで推定

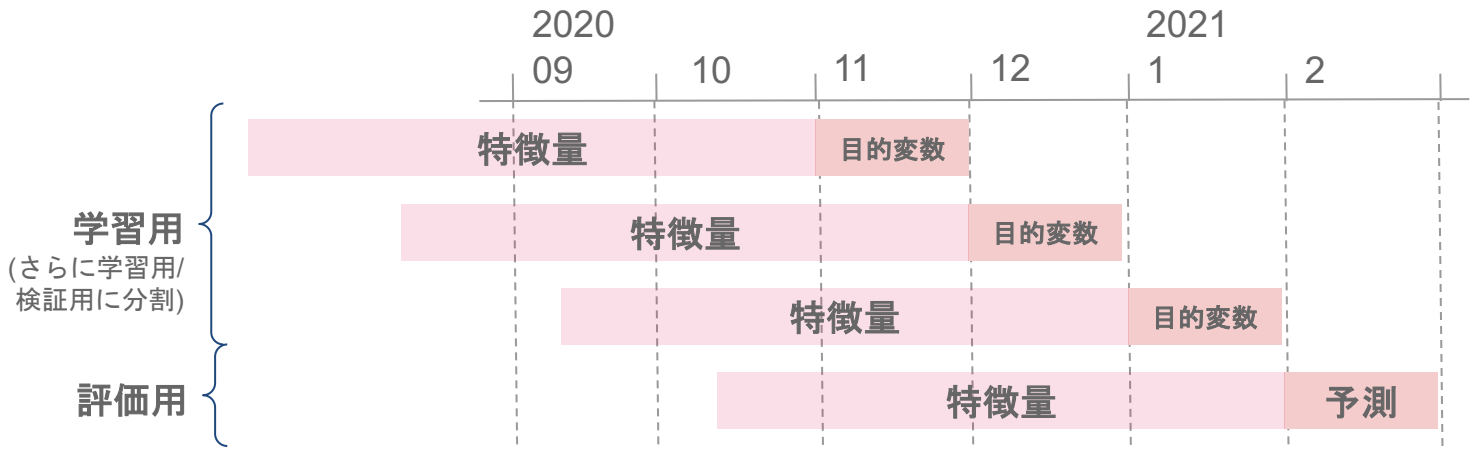
ファイルの欠陥混入可能性

	file_name	num_authors	num_files_containing	commits
0	Classes	53	5575	f089e74bf3ad9b773b7e56eaa3eb5740ac5576b7
1	Classes/AppDelegate.cpp	17	0	6a15f69e2578ecf70aadb95d009a2cd5bcb321dc

© DeNA Co.,Ltd.

4 ファイルの欠陥混入可能性 - データセットの設計

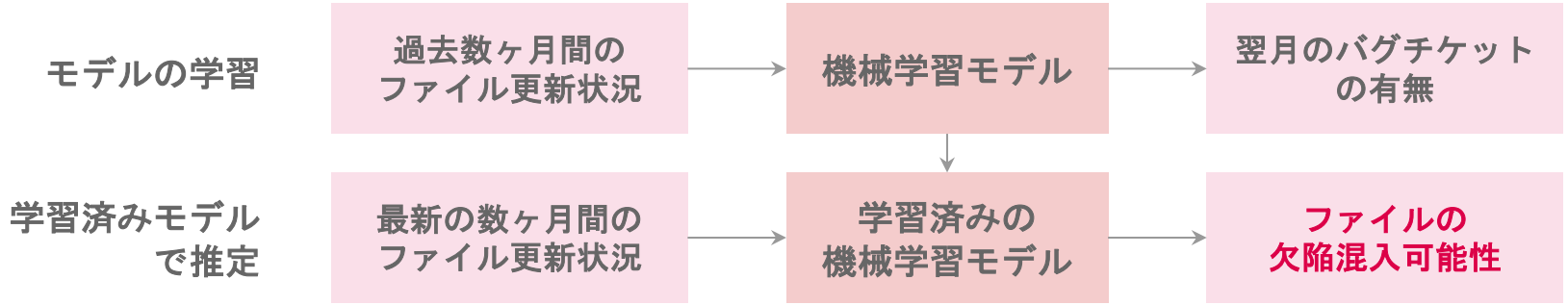
- 数ヶ月間のファイル更新状況を集計して、特徴量として使用
- 翌月のバグチケットの有無を、目的変数として使用
- 検証項目トレースを実施したレポジトリの最新時期のデータセットを評価用、それ以外を学習用として使用



© DeNA Co.,Ltd.

4 ファイルの欠陥混入可能性 - モデルの学習/推定および評価方法

- 過去数ヶ月間のデータセットで学習して、モデルを作成
- 最新のデータセットを学習済みモデルに入力して、欠陥混入可能性を推定
- 機械学習モデルには、LightGBM を使用
  - 決定木を多数重ねた上で、精度や速度を改善したモデル
- 予測値上位の欠陥混入の検知状況を評価するために、評価指標として Recall@K を使用
  - 予測値の上位K個に含まれる正解の割合
  - 今回は、Kを予測値の上位 [1%, 10%, 20%, 40%] の4パターンとして比較



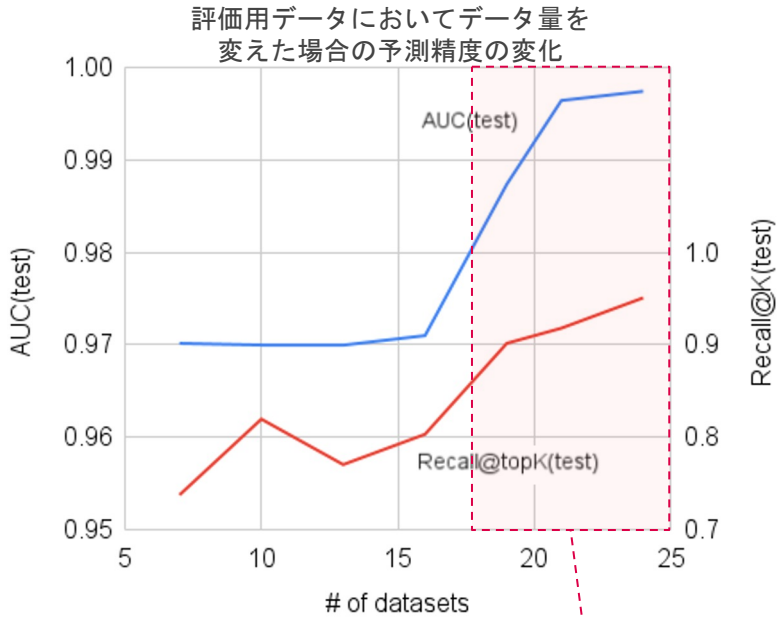
機械学習モデルの評価指標 ▶ Recall@K

- 「欠陥混入可能性の高いファイルに絞ることでどれだけ実際の欠陥混入を捕捉できるか」を評価する指標
- 0~1.0 の範囲を取り、1.0に近いほど高精度
- Kを予測値の上位 [1%, 10%, 20%, 40%] の4パターンとした

© DeNA Co.,Ltd.

4 ファイルの欠陥混入可能性 - 推定・分析結果

- そもそも欠陥やバグチケットは多くないため、正例割合が極端に低い（正例:負例 ≒ 1:1000）
- データ量、特に正例件数の増加のために、より多くの期間・タイトルのデータセットをまとめて学習に使用した
- およそ18ヶ月分以上のデータセットを重ねることで精度が安定する傾向



学習用データとして、この程度の量を
使うこととした

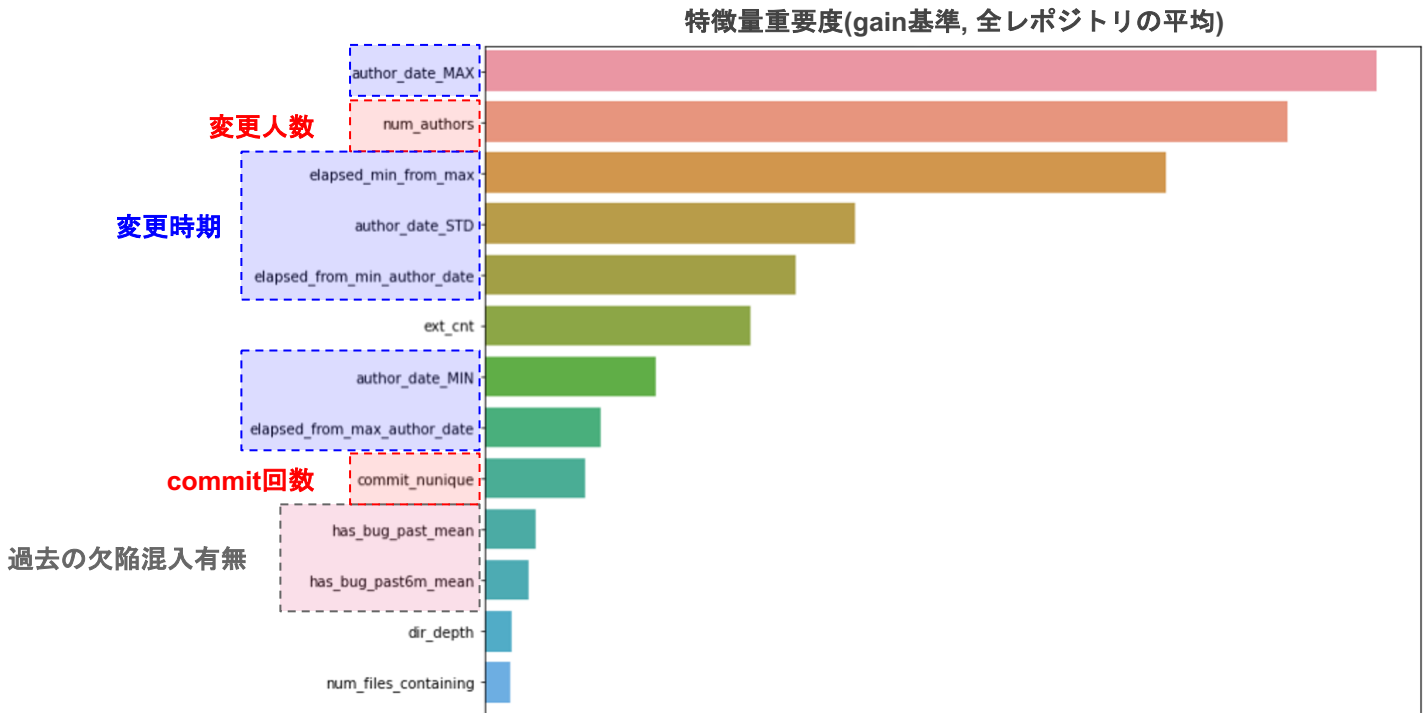
© DeNA Co.,Ltd.

4 ファイルの欠陥混入可能性 - 推定・分析結果

推定結果と関連が強い変数として、大きく分けて次の3種類が常に上位に現れてくる

- **変更の複雑さ（commit回数、変更人数）**
- **変更時期（初回or最終変更時期の近さ、初回変更~最終変更の経過日数）**
- **過去の欠陥混入有無**

※ 各変数の関連の度合いは時期ごとに変動することに注意



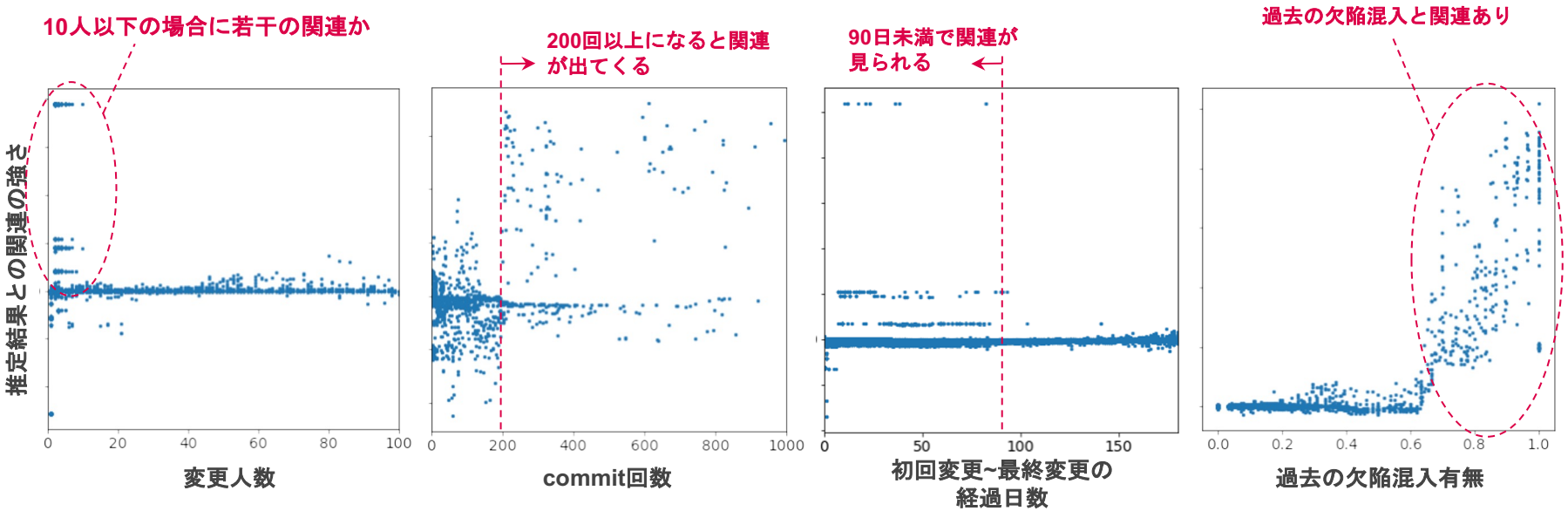
© DeNA Co.,Ltd.

4 ファイルの欠陥混入可能性 - 推定・分析結果

推定結果と関連が強い変数として、大きく分けて次の3種類が常に上位に現れてくる

- **変更の複雑さ（commit回数、変更人数）**
- **変更時期（初回or最終変更時期の近さ、初回変更~最終変更の経過日数）**
- **過去の欠陥混入有無**

※ 各変数の関連の度合いは時期ごとに変動することに注意

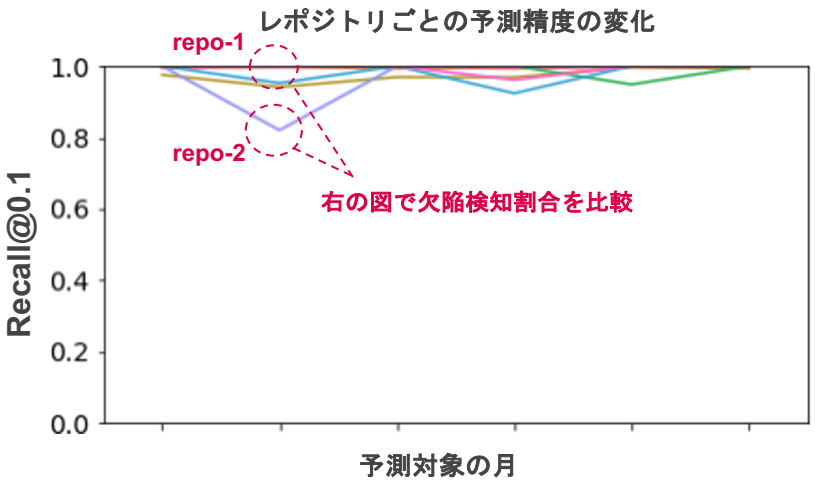


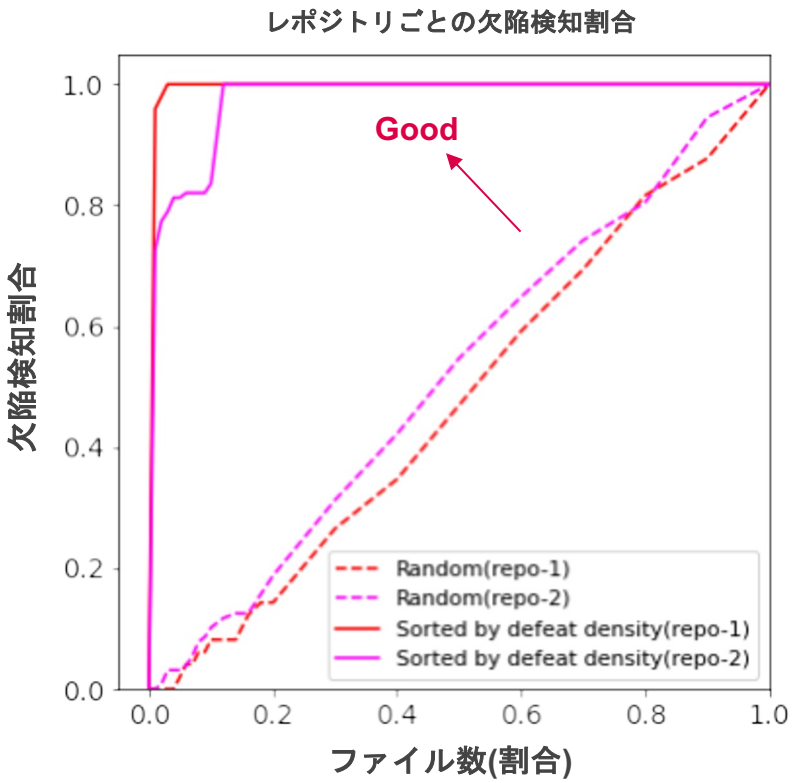
※各グラフで縦軸の尺度は異なります

© DeNA Co.,Ltd.

4 ファイルの欠陥混入可能性 - 推定・分析結果

- 予測精度が高く安定するレポジトリと、時期によって低下するレポジトリがある
- 全ての欠陥を検知するために必要なファイル数(割合)は次の通り
  - 予測精度が高く安定するレポジトリ：全体の10%未満
  - 予測精度が時期によって低下するレポジトリ：全体の20%程度





© DeNA Co.,Ltd.

4 ファイルの欠陥混入可能性 - まとめ

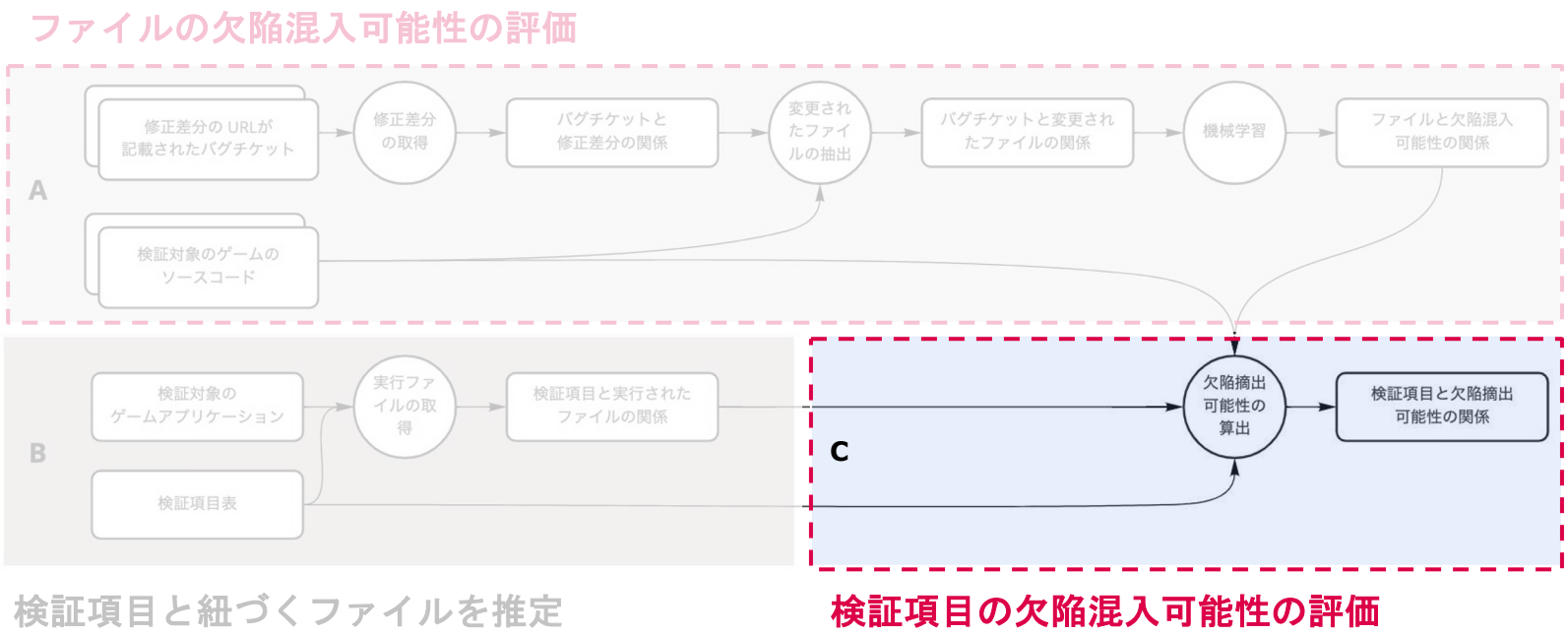
- 大半のレポジトリにおいて、**推定結果の上位10%で欠陥混入の90%以上を検知可能**であり、ファイルごとの欠陥混入可能性の推定精度としては及第点

- ただし、当初の目的であった**確認する検証項目を減らす**ことを達成するためには、検証項目ごとの欠陥混入可能性の推定精度の検証が必要

© DeNA Co.,Ltd.

4 検証項目の欠陥混入可能性 - 評価の概要

- 検証項目の欠陥混入可能性は、ファイルの欠陥混入可能性とファイル-検証項目の関係を紐付けることで推定している



© DeNA Co.,Ltd.

4 検証項目の欠陥混入可能性 - 評価の概要

- 検証項目の欠陥混入可能性の推定については**現在実現性を検証中のため、いずれも WIP** の内容になります
- ここでは仮にファイル-検証項目の紐付け状況が既知であるとして、推定精度を確認
  - ファイルの欠陥混入可能性の推定には、検証項目のトレースを実施したサービスや期間のデータを使用
  - トレースには工数を要するため、100項目をサンプリングして評価を実施
  - トレースの仕組み上、未検証項目の紐付け情報は取得できないため、紐付け状況の推定も検討中
- 検証項目単位の欠陥混入可能性の算出方法として、次の3種類を試した

①最も深刻な欠陥混入可能性を考慮

$$p_{item} = MAX(p_{file})$$

②欠陥混入可能性にファイル数増加の影響を考慮

$$p_{item} = 1 - \prod^{N_{item}} (1 - p_{file})$$

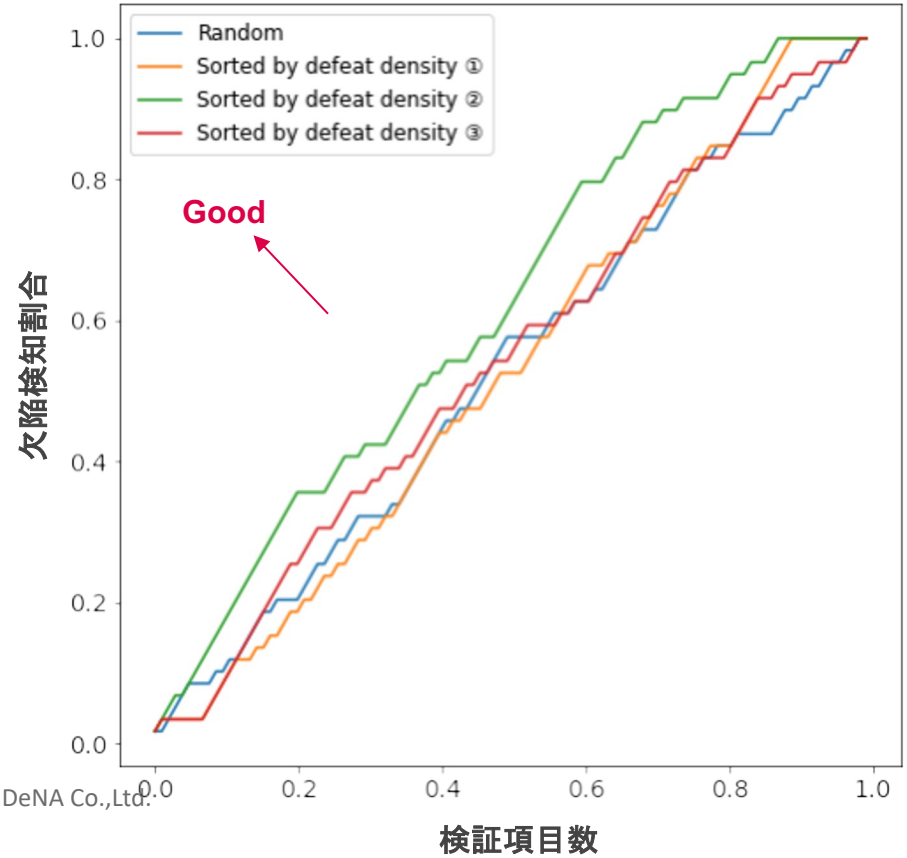
③②にしきい値(0.5)を設定

$$p_{item} = 1 - \prod^{N_{item}} [1 - MAX(threshold, p_{file})]$$

© DeNA Co.,Ltd.

4 検証項目の欠陥混入可能性 - 推定精度の検証

- 欠陥混入可能性の各算出方法における推定精度を、検証項目数に対する欠陥検知割合のカーブ下面積で比較
  - カーブ下面積は 0.5~1.0 の範囲を取り、1.0に近い(カーブがより左上を通る)ほど高精度と見なせる
- 既存手法に対して改善が見られ、特に②の場合に比較的良い推定精度が得られるものの、**業務導入にあたっては改善の余地あり**



欠陥混入可能性の算出方法	カーブ下面積
①最も深刻な欠陥混入可能性を考慮	0.5273
②欠陥混入可能性にファイル数増加の影響を考慮	0.6186
③②にしきい値(0.5)を設定	0.5422

© DeNA Co.,Ltd.

4 検証項目の欠陥混入可能性 - 推定精度の検証

- ②において推定精度が比較的高いが、個々のファイルの欠陥混入可能性より、単純にファイル数増加による欠陥遭遇リスクの増加の影響が強く出ている可能性がある
  - この現象の良し悪し含めて、今後検証を進める
- 精度改善に向けて、検証項目のトレース粒度を適正化することも検討
  - 機械学習の推定単位(=ファイル単位)に近づけると、機械学習の推定精度に近づくはず

欠陥混入可能性の算出方法	カーブ下面積	
①最も深刻な欠陥混入可能性を考慮	0.5273	→ 直感的には最も合いそうだが精度は上がらず (危ないファイルが1つでもあれば欠陥有り)
②欠陥混入可能性にファイル数増加の影響を考慮	0.6186	ファイル数増加による欠陥遭遇リスク増加の影響が強く出ている可能性がある
↓		
③②にしきい値(0.5)を設定 = ②で可能性が高いもののみに絞って考慮	0.5422	

© DeNA Co.,Ltd.

4 検証項目の欠陥混入可能性 - まとめ

- 既存手法に対して改善が見られ、考案した手法の有効性は確認できた
  - ファイル数増加の影響を考慮すると比較的良い推定精度が得られる

- しかし、業務導入をふまえると推定精度の改善は必要で、機械学習の推定単位と検証項目のトレース粒度の適正化が求められる

© DeNA Co.,Ltd.

05

まとめと展望

本事例の展望を説明します

© DeNA Co.,Ltd.

5 まとめと改善方針

ファイルの欠陥混入可能性

- 大半のレポジトリにおいて、**推定結果の上位10%で欠陥混入の90%以上を検知可能**であり、ファイルごとの欠陥混入可能性の推定精度としては及第点

検証項目の欠陥混入可能性

- 既存手法に対して改善が見られ、**考案した手法の有効性は確認できた**
  - ファイル数増加の影響を考慮すると比較的良い推定精度が得られる
- しかし、**業務導入をふまえると推定精度のさらなる改善は必要**

改善に向けたアイデア

- 機械学習の推定単位をファイルごとから関数やモジュールごとに変更
  - やり方次第で開発プロセスやソースコード管理の変更も伴うため、実施コストは要精査
- 検証項目における項目粒度やトレース粒度の調整
  - 機械学習の推定単位に近づけられるような項目粒度を探る必要がある

© DeNA Co.,Ltd.

一人ひとりに　想像を超えるDelightを

© DeNA Co.,Ltd.