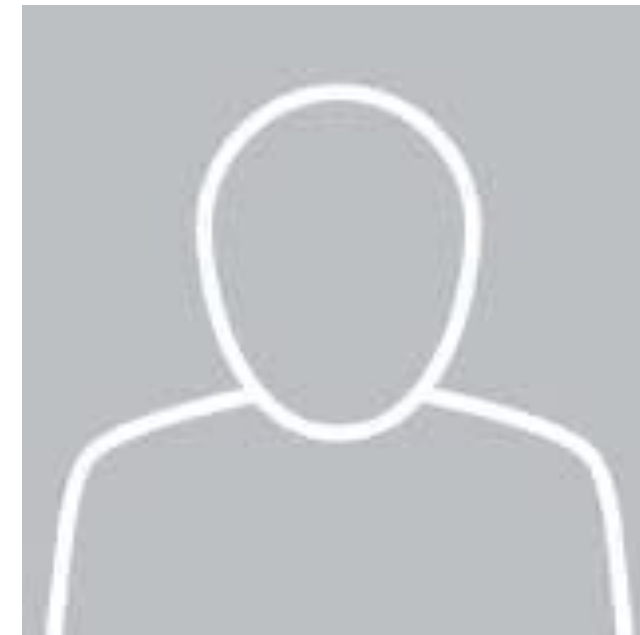


# 深層学習による不具合混入コミットの予測と評価

PSJ/SIGSE ソフトウェアエンジニアリングシンポジウム2017



近藤 将成



森 啓太



水野 修



崔 銀恵

紹介者：柏 祐太郎

# 背景

---

- 品質保証活動としてテストやレビューが重要
  - 効率的に行うために**不具合予測手法**が研究されている
- 不具合予測手法
  - 不具合が混入しそうなモジュールやファイルを予測
    - » 過去のソフトウェア開発の履歴情報やソースコードを利用
  - 不具合があると判断された所に重点的にテストやレビュー

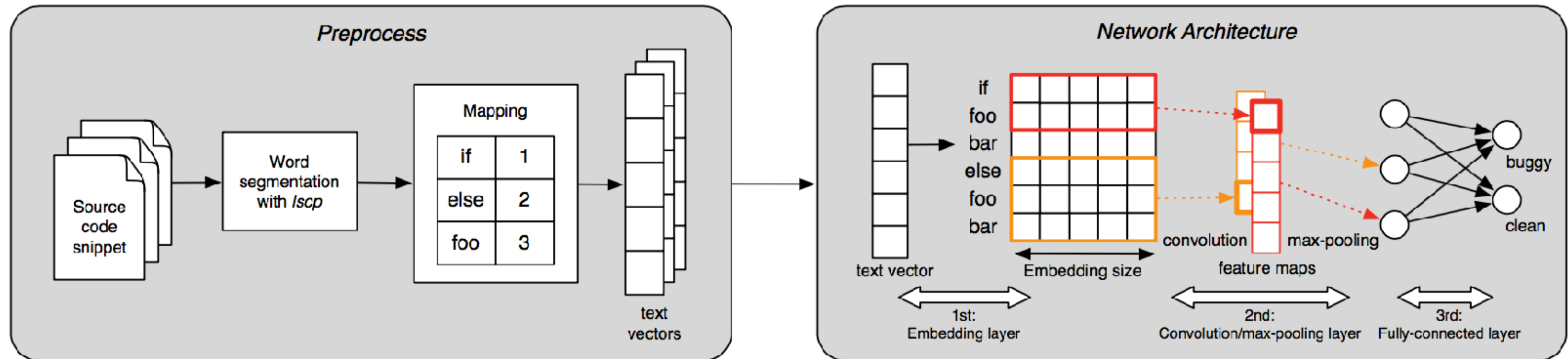
# 既存手法

---

- 機械学習を用いた方法が盛んに研究されてきた
- 最近では、深層学習を用いた方法が注目されている
  - 変更に対するソフトウェアメトリクスを用いる方法
  - 抽象構文木（ソースコード）の特徴を用いる方法
- **変更のソースコード片を用いる方法は未提案**
  - 変更コミット時に細かい粒度で不具合を予測できる
    - » 予測結果のフィードバックが早い

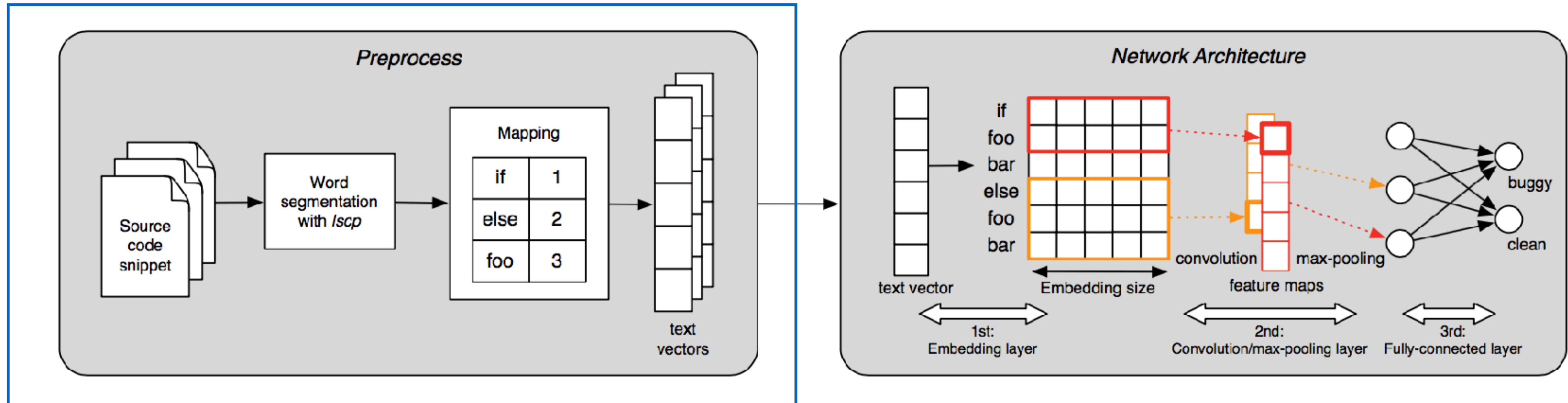
# 手法の全体像

- 前処理とネットワークの2つから成り立つ



# 手法の全体像

- 前処理とネットワークの2つから成り立つ



1. Gitのdiffコマンドで修正・追加工行+3行を取得
2. LSCP[1]で単語分割（構文解析ではなく，経験則的に単語分割）
3. 単語を頻出順に数字を1からマッピング（**非**頻出単語が重要）

[1] Thomas, W., S.: lscp: A lightweight source code preprocessor (2015).

# LSCPの例

入力

```

1  /* This multi-line comment
2  should not appear
3  in the final output.
4  */
5  class HelloWorldApp2 {
6      public static void main(String[] args) {
7          int ident1 = 0;
8          int ident2 = 1000;
9          if (0 == 1){
10             System.out.println("Hello World!");
11         }
12     }
13     // These comments should not be kept.
14     private static int otherFunction(int foo, int bar){
15         System.out.println("Another string, now with int inside!");
16         return foo+bar;
17     }
18 }

```



出力

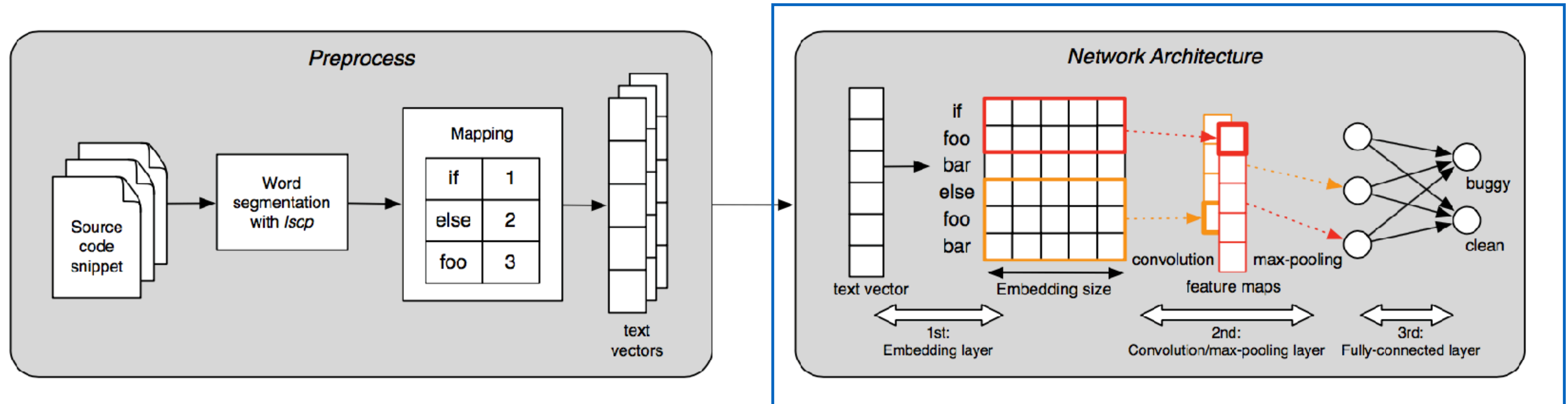
```

1  helloworldapp
2  args
3  ident
4  ident
5  system
6  out
7  println
8  hello
9  world
10 otherfunction
11 foo
12 bar
13 system
14 out
15 println
16 another
17 now
18 with
19 inside
20 foo
21 bar

```

# 手法の全体像

- 前処理とネットワーク (CNN) の2つから成り立つ

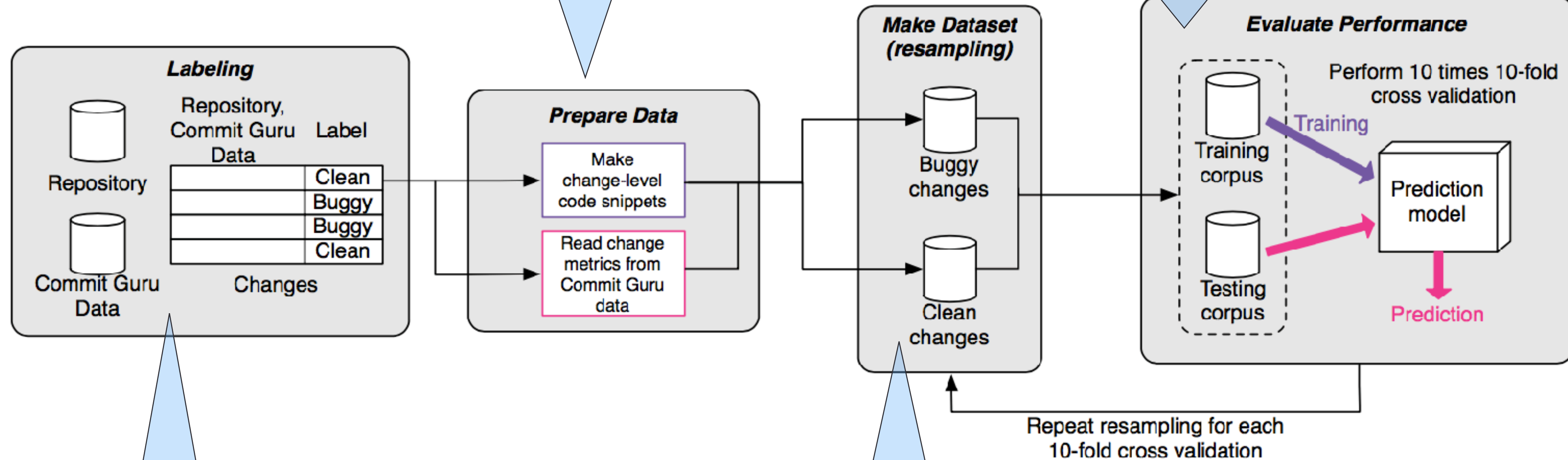


1. 128次元に変換
2. 新しい特徴を抽出するためのフィルターを学習
3. ソフトマックス関数で不具合が含まれる確率を求める

# 評価実験

- ・ 変更メトリクスを抽出  
(既存手法用で提案手法では不要)

- ・ 予測精度の評価



- ・ 各コミットがバグを含むかを調査
- ・ コード片を抽出

- ・ バグを含むコミットと  
含まないコミットの数揃える

# メトリクスとプロジェクト

表1 変更メトリクスの説明.

| 分類         | 名称      | 定義                  |
|------------|---------|---------------------|
| Diffusion  | NS      | 修正されたサブシステムの数       |
|            | ND      | 修正されたディレクトリの数       |
|            | NF      | 修正されたファイルの数         |
|            | Entropy | 修正されたコードの各ファイルごとの分布 |
| Size       | LA      | 追加されたコード行数          |
|            | LD      | 削除されたコード行数          |
|            | LT      | 変更される前のファイルのコード行数   |
| Purpose    | FIX     | バグ修正の変更か否か          |
| History    | NDEV    | 修正に関わった開発者の人数       |
|            | AGE     | 最後の変更から最新の変更までの平均時間 |
|            | NUC     | ユニークな変更の数           |
| Experience | EXP     | 開発者の経験              |
|            | REXP    | 最近の開発者の経験           |
|            | SEXP    | サブシステムについての開発者の経験   |

表2 対象プロジェクトの詳細.

| Project | Language | The Total Number<br>of Changes | Buggy Rate |
|---------|----------|--------------------------------|------------|
| Hadoop  | Java     | 13,920                         | 24.8%      |
| Camel   | Java     | 24,740                         | 23.2%      |
| Gerrit  | Java     | 18,794                         | 20.1%      |
| Osmand  | Java     | 31,366                         | 14.0%      |
| CMake   | C++      | 28,400                         | 10.1%      |
| Bitcoin | C++      | 11,093                         | 14.4%      |
| Gimp    | C++      | 37,116                         | 22.5%      |

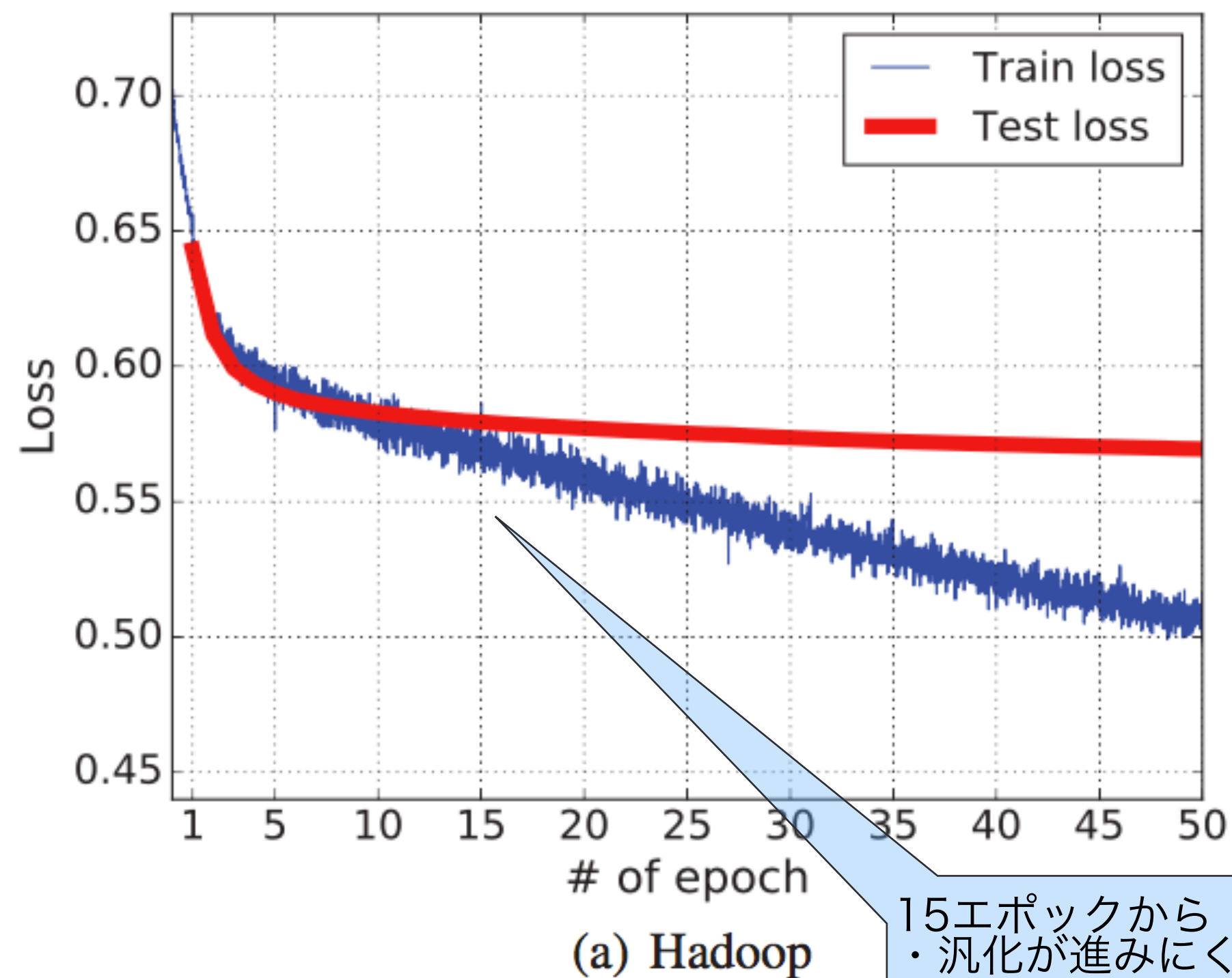
# Research Question

- RQ1 : 不具合予測に適用可能か
  - 訓練誤差とテスト誤差から以下を確認
    - » 不具合を学習可能か
    - » 過適合を起こさないか
- RQ2 : 既存手法より予測精度が良いか
  - 既存手法Deeper[1]とAUCで比較
- RQ3 : 学習時間はどの程度必要か
  - 既存手法と学習時間で比較

# RQ1：不具合予測に適用可能か

11

- 学習が進むごとに訓練誤差・テスト誤差が減少
- 学習が進むごとに平均AUCが向上



15エポックから  
・汎化が進みにくくなる  
・訓練誤差とテスト誤差が広がる

表4 学習過程における W-CNN の AUC のプロジェクト間での平均値.

| エポック数 | 5     | 10    | 15    | 20    | 25    | 30    | 35    | 40    | 45    | 50    |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| AUC   | 0.788 | 0.797 | 0.803 | 0.807 | 0.811 | 0.813 | 0.815 | 0.816 | 0.816 | 0.817 |

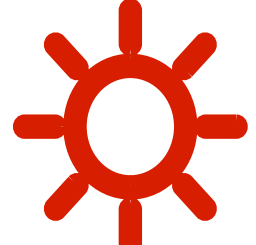
- AUCで比較
  - 全プロジェクトにおいて予測精度が向上

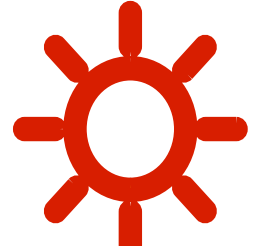
表5 W-CNN 15, W-CNN 50, 及び, Deeper の AUC.

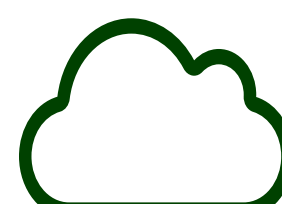
| プロジェクト  | 提案手法     |          |   | 従来手法   |
|---------|----------|----------|---|--------|
|         | W-CNN 15 | W-CNN 50 |   | Deeper |
| Hadoop  | 0.788    | 0.802    | > | 0.684  |
| Camel   | 0.783    | 0.798    | > | 0.630  |
| Gerrit  | 0.817    | 0.832    | > | 0.752  |
| Osmand  | 0.810    | 0.825    | > | 0.698  |
| CMake   | 0.803    | 0.821    | > | 0.622  |
| Bitcoin | 0.793    | 0.811    | > | 0.675  |
| Gimp    | 0.825    | 0.830    | > | 0.624  |
| AVG     | 0.803    | 0.817    |   | 0.669  |

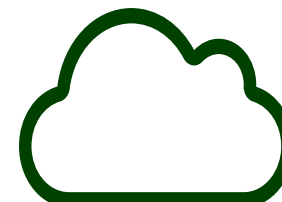
# 所感

---

 流行りのディープラーニング

 予測精度が約22%向上

 誰かが既にやってそうな気がするけど . . .

 RQ1の結論が多少強引な気がする

訓練誤差とテスト誤差が減少 → 不具合予測に適用可能