

A Cost-efficient Approach to Building in Continuous Integration

Xianhao Jin, Francisco Servant

CIを構築するための費用対効果の高いアプローチ

Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, Pages 13–25

発表者：辻 裕真

背景

- ・ 継続的インテグレーション(CI)はソフトウェア開発において一般的な手法
- ・ CIの採用にはかなりの費用がかかる。
(Mozillaの試算では月額201,000ドル)
- ・ CIのコストはビルドにかかるコストがほとんど
- ・ ビルドの実行を少なくし、CIのコストを削減することが目的

仮説

H1 : CIのほとんどのビルドは成功する.

- ・ ビルドの結果を予測し成功すると予想されるものをスキップすることで大幅なコスト削減につながる.

H2 : CIで失敗するビルドの多くは連続して失敗する.

- ・ 失敗したビルドの後続のビルドは失敗すると予測するシステムを構築した場合, 失敗するビルドを正しく予測できる.

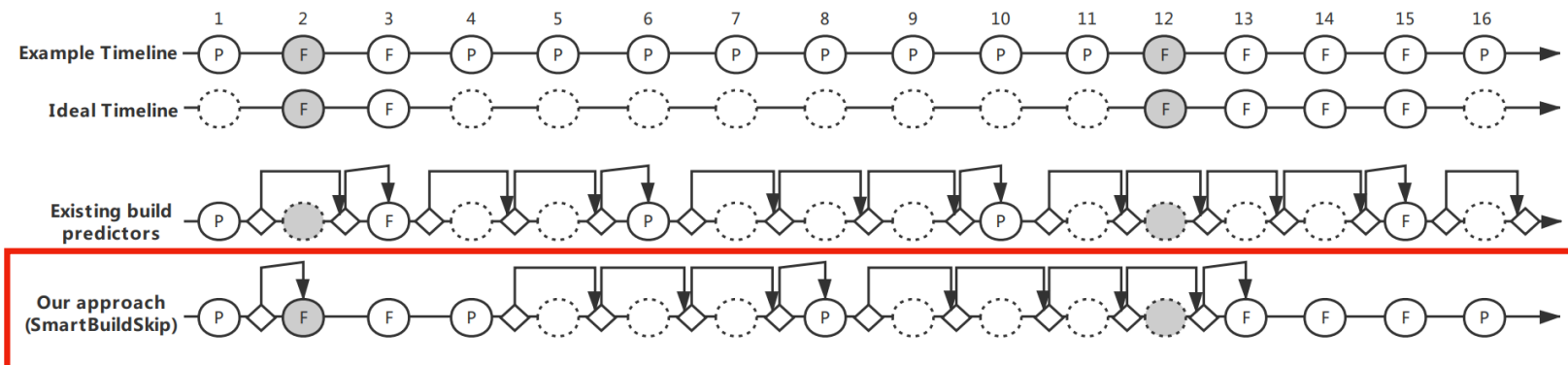
SmartBuildSkip

CIコストを削減する新しいアプローチ

1. 次に行われるビルドの結果を予測

- ・ 成功：ビルドを実行しない 失敗：ビルドを実行
- ・ ビルドを実行し失敗した際2のフェーズへ

2. 以降のビルドは失敗すると仮定し成功するまでビルドを実行



Research Questions

- ・ 仮説の評価

RQ1：失敗するビルドは成功するビルドより多いか.

RQ2：失敗したビルドが連続することは多いか.

- ・ 最初の失敗の特徴

RQ3：最初のビルド失敗の特徴は何か.
(連続して失敗しているうちの最初)

Research Questions

- SmartBuildSkipの評価

RQ4 : SmartBuildSkipは最初の失敗をどの程度予測できるか.

RQ5 : SmartBuildSkipは全ての失敗をどの程度予測できるか.

データセット

- TravisTorrent

1,359プロジェクトの2,640,825のビルドデータ

- 調査対象

1年以上経過, ビルドが200以上, ソースコードが1000行以上

359プロジェクトの274,742(失敗: 53,731)のビルドデータを
収集

RQ1-結果

RQ1：失敗するビルドは成功するビルドより多いか.

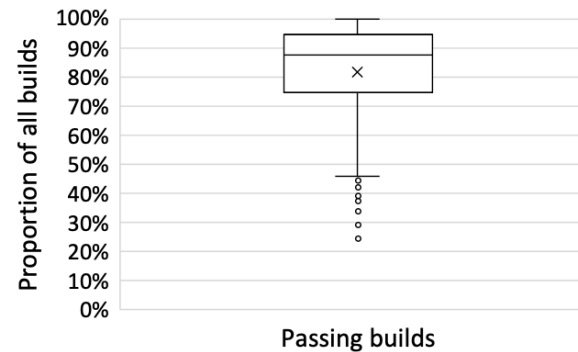


Figure 2: Ratio of passing builds to all builds. Passing builds represent a vast portion of all builds.

- 平均84%(中央値88%)が成功
- ほとんどのプロジェクトで成功したビルドが大きな割合を占めていた.
- 成功するビルドをスキップすることでコストを大幅に削減できる.

RQ2-結果

RQ2：失敗したビルドが連続することは多いか.

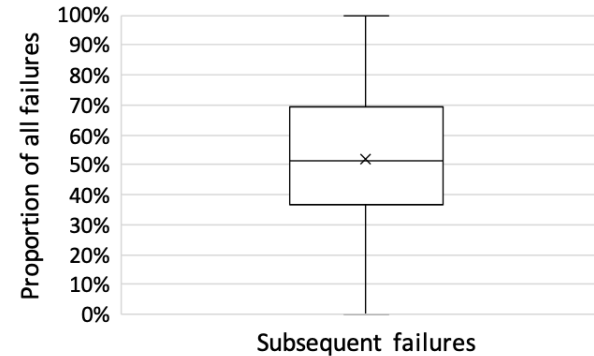


**Figure 3: Ratio of *subsequent failures* to *all build failures*.
More than half of *all build failures* are *subsequent failures*.**

- ・ プロジェクトの50%で52%以上が連続して失敗, 75%で38%以上が連続して失敗していた.
- ・ 失敗の後のビルドは失敗と予測することは有用.

RQ3-調査方法

RQ3：最初のビルド失敗の特徴は何か.

TravisTorrentに含まれるビルドの特徴, プロジェクトの特徴と最初に失敗したビルドとの関連を調査

Table 1: Features studied for correlation with *first failures*.

Build features	
Feature	Short Description
src_churn (SC)	The number of changed source lines since the last build.
file_churn (FC)	The number of changed source files since the last build.
test_churn (TC)	The number of changed test lines since the last build.
num_commit (NC)	The number of commits since the last build.
Project_performance_short (PS)	The proportion of passing builds in the recent five builds.
Project_performance_long (PL)	The proportion of passing builds in the whole previous builds.
Time_frequency (TF)	The time gap (hour) since the last build.
Failure_Distance (FD)	The number of builds since the last failing build.
Week_day (WD)	The weekday [0, 6] (0 being Monday) of the build.
Day_time (DT)	The time of day [0, 23] of the build.

Project features	
Feature	Short Description
Team_size (TS)	The median number of developers over the project's CI usage history.
Project_size (PS)	The median number of executable production source lines of code in the repository over the project's CI usage history.
Project_age (PA)	The time duration between the first build and the last build for that project.
Test_density (TD)	The median number of lines in test cases per 1000 executable production source lines over the project's CI usage history.

RQ3-結果

RQ3：最初のビルド失敗の特徴は何か.

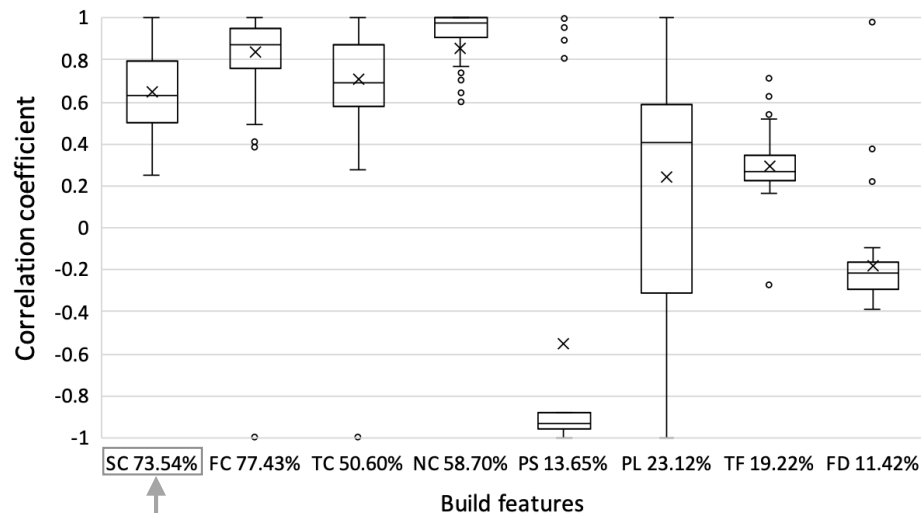


Figure 4: Correlation between build-features and ratio of first failures. Four build features (SC, FC, TC, NC) had a statistically significant correlation for more than 50% of projects.

有意であったプロジェクトの割合

- 有意であった特徴

SC, FC, TC, NC

変更が大きいほどビルドが失敗しやすくなる.

RQ3-結果

RQ3：最初のビルド失敗の特徴は何か.

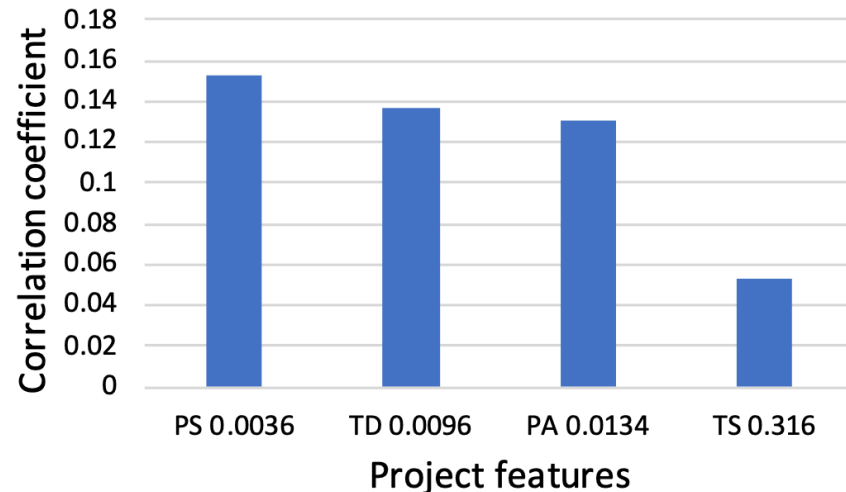


Figure 5: Correlation between project features and ratio of *first failures*. The correlation was statistically significant for three project features: PS, TD, PA.

- ・ 有意であった特徴

PS, TD, PA

プロジェクトサイズ, テストケース, CI使用期間が大きくなるとビルドが失敗しやすい.

RQ4&5-調査方法

RQ4 : SmartBuildSkipは最初の失敗をどの程度予測できるか.

RQ5 : SmartBuildSkipは全ての失敗をどの程度予測できるか.

- ・ 最先端手法との比較

SmartBuildSkipと最先端のビルド予測手法であるHW17で予測効果を比較

- ・ HW17 : 現在のビルドの16の特徴, 過去のビルドの4の特徴, ビルドログの分析により生成された8つの特徴を用いたランダムフォレストアルゴリズムで予測
- ・ SmartBuildSkip : RQ3で有意であった特徴を用いたランダムフォレストアルゴリズムで予測

RQ4&5-調査方法

RQ4 : SmartBuildSkipは最初の失敗をどの程度予測できるか.

RQ5 : SmartBuildSkipは全ての失敗をどの程度予測できるか.

4つの異なるアプローチを評価

- SmartBuildSkip-Whithin

同一プロジェクトで学習し, ビルドの特徴を用いる.

- SmartBuildSkip-Cross

複数のプロジェクトで学習し, ビルドとプロジェクトの特徴を用いる.

RQ4&5-調査方法

RQ4 : SmartBuildSkipは最初の失敗をどの程度予測できるか.

RQ5 : SmartBuildSkipは全ての失敗をどの程度予測できるか.

4つの異なるアプローチを評価

- HW17-Within

同一プロジェクトで学習

- HW17-Cross

複数のプロジェクトで学習

RQ4-結果

RQ4 : SmartBuildSkipは最初の失敗をどの程度予測できるか.

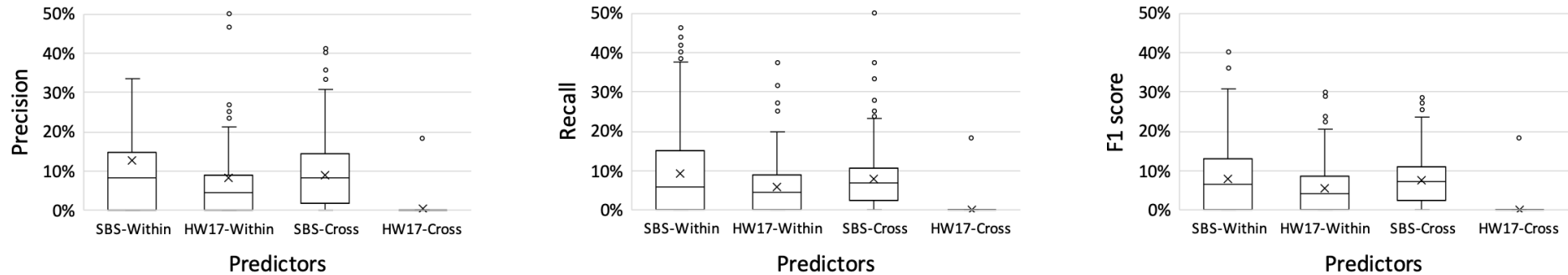


Figure 6: Performance comparison on predicting *first failures*

- Precision

Withinで3%, Crossで9%改善

- Recall

Withinで4% , Crossで7%改善

RQ5-結果

RQ5 : SmartBuildSkipは全ての失敗をどの程度予測できるか.

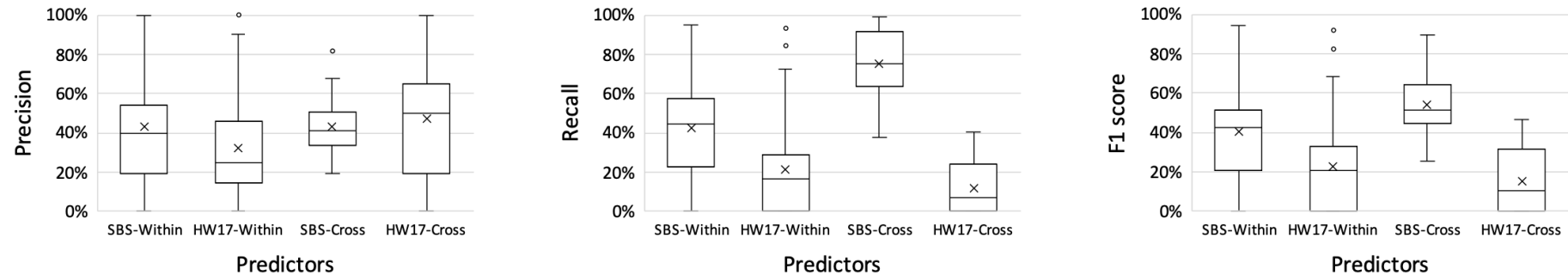


Figure 7: Performance comparison on predicting *all failures*

- Precision

Withinで16%改善, Crossで9%悪化

- Recall

Withinで28%, Crossで68%改善

結論

- ・ 成功するビルドの割合は大きい.
- ・ 失敗したビルドの多くが連続で失敗する.
- ・ CIコスト節約のための新しい手法であるSmartBuildSkipの提案と評価

最先端のビルド予測手法のPrecisionとRecallを改善できた

所感

- ・ 成功すると予測された際にビルドをスキップするのはコスト削減につながると思うが、失敗する可能性や精度の低さを考えると実際の現場で活用することはできるのか.