

A Large-Scale Empirical Study of Just-in-Time Quality Assurance

Yasutaka Kamei, Emad Shihab, Bram Adams,
Ahmed E. Hassan, Audris Mockus, Anand Sinha,
Naoyasu Ubayashi

IEEE Transactions on Software Engineering 2013 pp. 757-773

バグ予測

- バグを含む（可能性がある）モジュールやファイルを特定する技術
- 蓄積したソフトウェアのデータを用いる
 - プロダクト
 - LOC, 複雑度, オブジェクト指向, etc...
 - プロセス
 - 変更行数, 開発者の数, 経験度, etc...

従来のバグ予測手法の問題点

- 予測する箇所の粒度が荒い（パッケージなど）
 - 予測した範囲からどこにバグが混入しているかを探すのに時間がかかる
- 誰が混入させたかわからない
 - バグを混入させた開発者を特定する必要がある

Just-in-Time defect prediction

- 粒度の細かい変更からバグを予測
- 混入させたバグの修正業務割り当て
- 変更追加時にバグを予測

目的

- JITバグ予測の手法構築
- JITバグ予測によるバグ修正のコスト削減

リサーチクエスチョン

- RQ1

- 構築した予測モデルの精度はどれ程か

- RQ2

- 予測されたリスクに基づく変更の優先付けは工数を削減するか

- RQ3

- バグを混入させる変更の主な特徴は何か

予測モデル構築

TABLE 1
Summary of Change Measures

Dim.	Name	Definition	Rationale	Related Work
Diffusion	NS	Number of modified subsystems	Changes modifying many subsystems are more likely to be defect-prone.	The defect probability of a change increases with the number of modified subsystems [41].
	ND	Number of modified directories	Changes that modify many directories are more likely to be defect-prone.	The higher the number of modified directories, the higher the chance that a change will induce a defect [41].
	NF	Number of modified files	Changes touching many files are more likely to be defect-prone.	The number of classes in a module is a good feature of post-release defects of a module [47]
	Entropy	Distribution of modified code across each file	Changes with high entropy are more likely to be defect-prone, because a developer will have to recall and track large numbers of scattered changes across each file.	Scattered changes are more likely to introduce defects [10], [19].
Size	LA	Lines of code added	The more lines of code added, the more likely a defect is introduced.	Relative code churn measures are good indicators of defect modules [43], [46].
	LD	Lines of code deleted	The more lines of code deleted, the higher the chance of a defect.	
	LT	Lines of code in a file before the change	The larger a file, the more likely a change might introduce a defect.	
Purpose	FIX	Whether or not the change is a defect fix	Fixing a defect means that an error was made in an earlier implementation, therefore it may indicate an area where errors are more likely.	Changes that fix defects are more likely to introduce defects than changes that implement new functionality [16][52].
History	NDEV	The number of developers that changed the modified files	The larger the NDEV, the more likely a defect is introduced, because files revised by many developers often contain different design thoughts and coding styles.	Files previously touched by more developers contain more defects [32].
	AGE	The average time interval between the last and the current change	The lower the AGE (i.e., the more recent the last change), the more likely a defect will be introduced.	More recent changes contribute more defects than older changes [15].
	NUC	The number of unique changes to the modified files	The larger the NUC, the more likely a defect is introduced, because a developer will have to recall and track many previous changes.	The larger the spread of modified files, the higher the complexity [10], [19].
Experience	EXP	Developer experience	More experienced developers are less likely to introduce a defect.	Programmer experience significantly decreases the defect probability [41].
	REXP	Recent developer experience	A developer that has often modified the files in recent months is less likely to introduce a defect, because she will be more familiar with the recent developments in the system.	
	SEXP	Developer experience on a subsystem	Developers that are familiar with the subsystems modified by a change are less likely to introduce a defect.	

予測モデル構築

- メトリクスの定義

- 拡散

- 変更が拡散するほどバグが混入しやすくなる

- サイズ

- プログラムのサイズが大きいほどバグが混入しやすくなる

- 目的

- バグ修正の際にバグが混入しやすい

- 歴史

- 最近変更が加えられたファイルはバグが混入しやすい

- 経験

- 経験の浅い開発者はバグを混入させやすい

予測モデル構築

- モデル構築

- ロジスティック回帰を使用

- 説明変数は定義したメトリクス
 - 目的変数はバグが混入したかしていないか
 - 予測値 $\geq 0.5 \rightarrow 1$, 予測値 $< 0.5 \rightarrow 0$

- 前処理

- アンダーサンプリング

- 評価

- 10回交差検証

データセット

TABLE 2
Statistics of the Studied Projects

	Period	The total number of changes		Average LOC		# of modified files per change	# of changes per day	# dev. per file Max Avg	
				File	Change				
Bugzilla	08/1998 - 12/2006	4,620	(36%)	389.8	37.5	2.3	1.5	37	8.4
Columba	11/2002 - 07/2006	4,455	(31%)	125.0	149.4	6.2	3.3	10	1.6
Eclipse JDT	05/2001 - 12/2007	35,386	(14%)	260.1	71.4	4.3	14.7	19	4.0
Eclipse Platform	05/2001 - 12/2007	64,250	(14%)	231.6	72.2	4.3	26.7	28	2.8
Mozilla	01/2000 - 12/2006	98,275	(5%)	360.2	106.5	5.3	38.9	155	6.4
PostgreSQL	07/1996 - 05/2010	20,431	(25%)	563.0	101.3	4.5	4.0	20	4.0
OSS-Median	-	27,909	(20%)	310.1	86.7	4.4	9.4	24	4.0
C-1	10/2000 - 12/2009	4,096		-	16.4	2.0	1.2	-	-
C-2	10/2000 - 12/2009	9,277		-	19.2	2.4	2.8	-	-
C-3	07/2002 - 12/2009	3,586		-	16.6	2.0	1.3	-	-
C-4	12/2003 - 12/2009	5,182		-	12.9	1.8	2.4	-	-
C-5	10/1982 - 12/1995	10,961		303.0	39.0	4.8	2.3	-	-
COM-Median	-	5,182		-	16.6	2.0	2.3	-	-

†The percentage in brackets shows the percentage of defect-inducing changes to all changes.

RQ1:構築した予測モデルの精度はどれ程か

TABLE 4
Summary of Prediction Performance (RQ1)

	Acc.	Prec. threshold: 0.5	Recall	F1	AUC	% improved Prec. AUC	
BUZ	67%	54%	69%	60%	74%	48.9%	48.0%
COL	70%	51%	67%	58%	75%	64.3%	51.0%
JDT	69%	26%	65%	37%	74%	87.6%	47.0%
PLA	67%	27%	70%	38%	75%	89.3%	50.3%
MOZ	77%	13%	63%	22%	77%	168.4%	54.9%
POS	74%	49%	65%	56%	79%	97.1%	57.9%
--- Avg	71%	37%	67%	45%	76%	92.6%	51.5%
Med	69%	38%	66%	47%	75%	88.4%	50.6%
C-1	66%	35%	58%	44%	68%	> 20.0	35.4%
C-2	63%	31%	61%	41%	68%	> 20.0	35.1%
C-3	63%	47%	62%	54%	69%	> 20.0	37.8%
C-4	66%	41%	61%	49%	69%	> 20.0	37.4%
C-5	70%	5%	66%	10%	73%	> 20.0	46.9%
--- Avg	66%	32%	62%	39%	69%	> 20.0	38.5%
Med	66%	35%	61%	44%	69%	> 20.0	37.4%
All-Avg	68%	34%	64%	43%	73%	> 20.0	45.6%
All-Med	67%	35%	65%	44%	74%	> 20.0	47.0%

"> 20.0" means that the improvement is greater than 20.0%.

RQ2: 予測されたリスクに基づく変更の優先付けは工数を削減するか

- 工数を考慮した評価

- 全体の**20%**を見ることがどれだけのバグを予測できるかを評価

- RQ1で構築したモデルと工数を考慮したモデルとで P_{opt} を比較

- $P_{opt} = 1 - \Delta_{opt} \ [0,1]$

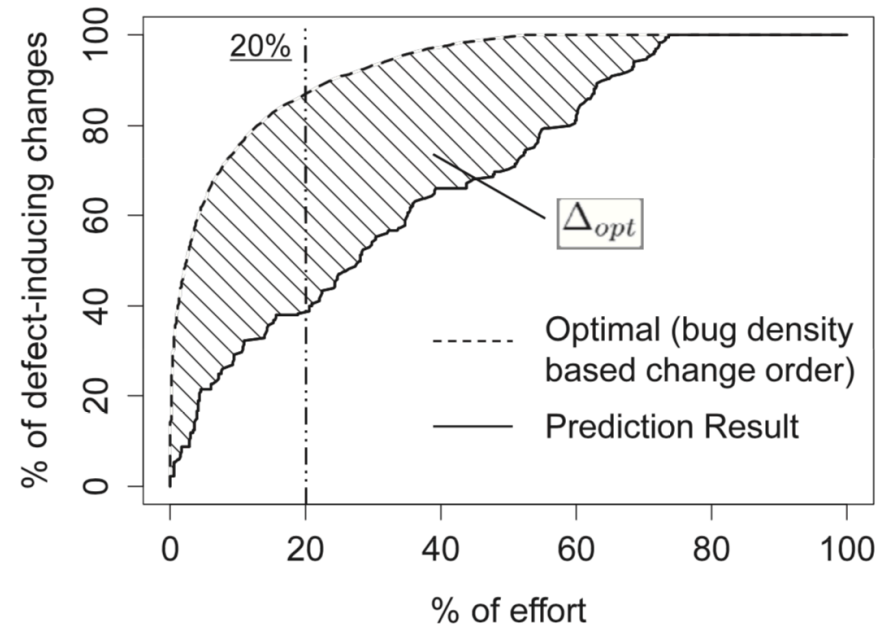


Fig. 2. Example of an effort-based cumulative lift chart for Bugzilla.

RQ2: 予測されたリスクに基づく変更の優先付けは工数を削減するか

- 工数を考慮した予測モデル（線形回帰）
 - 目的変数を以下の式で得られる $R_d(x)$ に変更
 - $R_d(x) = \frac{Y(x)}{Effort(x)}$ （工数が小さければ上位）
 - $Y(x)$ ：バグが混入したかどうか 0 or 1
 - $Effort(x)$ ：変更の規模
 - 予測結果を降順にソートし，上位20%を見るだけでどれだけバグを特定できるかを評価

RQ2: 予測されたリスクに基づく変更の優先付けは工数を削減するか

- Accuracy, P_{opt} とともに工数を考慮したモデルの方が大きい

TABLE 5
Summary of Prediction Performance (RQ2)

	LR		EALR	
	Acc.	P_{opt}	Acc.	P_{opt}
Bugzilla	11%	45%	38%	72%
Columba	2%	19%	39%	57%
JDT	1%	27%	20%	48%
Platform	4%	34%	30%	55%
Mozilla	6%	38%	15%	45%
PostgreSQL	3%	27%	26%	51%
----- Avg	5%	32%	28%	55%
Median	4%	31%	28%	53%
C-1	9%	36%	29%	60%
C-2	5%	32%	47%	72%
C-3	6%	27%	50%	75%
C-4	8%	33%	38%	66%
C-5	14%	34%	51%	71%
----- Avg	8%	32%	43%	69%
Median	8%	33%	47%	71%
All-Avg	6%	32%	35%	61%
All-Median	6%	33%	38%	60%

{Acc., P_{opt} } for the random predictor is {20, 50}

RQ3: バグを混入させる変更の主な特徴は何か

- RQ1とRQ2のモデルでどのメトリクスがモデルに影響を与えるかを調査
- RQ1のモデルではオッズ比を利用
- RQ2のモデルでは正と負の係数を数え、どちらが多いかで評価

RQ3:バグを混入させる変更の主な特徴は何か

TABLE 6
The Impact of Change Factors on Defect-Inducing Changes in the Six Open Source Projects and Five Industry Projects

Metrics name	All		OSS								COM						
	+	-	+	-	BUZ	COL	JDT	PLA	MOZ	POS	+	-	C-1	C-2	C-3	C-4	C-5
NS	4	0	3	0	2.09			1.62		1.36	1	0	2.62				
NF	11	0	6	0	2.95	3.00	2.62	3.07	4.29	5.61	5	0	1.62	1.33	4.26	2.10	1.87
Entropy	1	5	0	4		0.44		0.60	0.49	0.62	1	1			0.26		2.28
LA/LT	10	0	6	0	5.62	4.13	8.72	2.74	2.85	20.43	4	0	1.23	1.24	1.35	1.36	
LD/LT	1	0	1	0				1.25			0	0					
LT/NF	9	0	5	0		1.54	1.19	1.38	1.15	1.24	4	0	1.37	1.36	1.34	1.67	
FIX	4	0	4	0	4.54		2.20	3.37	2.07		0	0					
NDEV	3	2	0	2				0.81		0.74	3	0		1.23	2.33	1.73	
AGE	2	6	0	6	0.72	0.84	0.88	0.90	0.85	0.76	2	0	1.13	1.11			
NUC/NF	3	0	3	0				4.50	3.20	7.40	0	0					
EXP	3	3	2	3	0.89	0.79	1.33	1.14	0.92		1	0		1.85			
SEXP	3	3	1	3		1.24	0.72	0.94	0.94		2	0			1.15	1.11	

上の表はRQ1のモデルののオッズ比, 下の表はRQ2のモデルの係数を示している

TABLE 7
The Regression Coefficients of Change Factors in the Effort-Aware Models

Metrics name	All		+	-	OSS						+	-	COM				
	+	-			BUZ	COL	JDT	PLA	MOZ	POS			C-1	C-2	C-3	C-4	C-5
NS	0	0	0	0							0	0					
NF	6	4	6	0	0.03	0.07	0.04	0.05	0.05	0.04	0	4	-0.07	-0.06	-0.07	-0.06	
Entropy	0	6	0	6	-0.10	-0.33	-0.17	-0.24	-0.21	-0.12	0	0					
LA/LT	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
LD/LT	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
LT/NF	3	4	3	3	-0.01	0.03	-0.01	0.01	0.01	-0.01	0	1					-0.01
FIX	4	0	4	0	0.07		0.04	0.06	0.01		0	0					
NDEV	2	7	0	6	-0.04	-0.15	-0.06	-0.06	-0.04	-0.09	2	1		-0.07	0.06	0.07	
AGE	1	6	0	6	-0.02	-0.04	-0.03	-0.03	-0.02	-0.02	1	0					0.00
NUC/NF	0	5	0	5		-0.53	-0.26	-0.32	-0.22	-0.17	0	0					
EXP	3	2	2	1		-0.03	0.03	0.01			1	1		0.05			-0.01
SEXP	1	4	1	3		0.04	-0.02	-0.01	-0.01		0	1	-0.02				

まとめ

- JITバグ予測手法を構築
- 工数を考慮することで効果的にバグの予測が可能
- 修正されたことがあるファイルで変更の規模が大きいバグに欠陥が混入しやすい

所感

- メトリクスを選択理由が参考になった
- モデルの評価の際に訓練データとテストデータの
時系列を考慮しているのか気になった