故障モジュール中の欠陥メソッド特定

畑 秀明[1,a)]　門田 暁人[1,b)]　松本 健一[1,c)]

概要：ソフトウェアの欠陥予測は，全モジュールのうち欠陥が含まれている可能性の高いモジュールを予測する．この予測によって，テストやレビューの工数を効率よく集中させることが期待されている．しかし実際のソフトウェア開発環境では担当モジュールが定められていることもあり，全てのモジュールから欠陥モジュールを予測することが好ましくないこともある．また，デバッグ時に，故障していると思われるモジュールをある程度特定できていることもある．そこで本稿では，故障モジュール（ディレクトリ）を与えられた際にその中から欠陥メソッドを特定する，局所的細粒度欠陥モジュール予測手法を提案する．オープンソースソフトウェアでの適用実験で，平均 0.62 から 0.81 の F 値を得た．

1. はじめに

ソフトウェア開発やメインテナンスにおいて，高い品質を保持することは重要な課題である．テストやレビューといった品質保証活動に対して，欠陥モジュール予測は工数を効率よく集中させると期待されている．例えば，BugCache という欠陥モジュール予測アルゴリズムは，Google 社内でのケーススタディでもその有用性が報告されている [1]．また，Monden らは，シミューレーションによって，欠陥モジュール予測の結果を活用してテスト工数を削減できることを示した [2]．Microsoft では，欠陥モジュール予測，変更のリスク分析，テスト優先付けを統合したシステムを開発し，実際の運用からその有用性を報告している [3]．

多くの欠陥モジュール予測研究では，テストまたはレビューへの適用が議論されてきた [4]．本稿では，これまで行われていない，デバッグに適用できる欠陥モジュール予測手法を提案する．Layman らは，実際のソフトウェア開発者らがどのようにデバッグしているかを Microsoft 社内で行ったインタビューを行った [5]．このインタビュー結果の分析から，デバッグにはソースコード以外の情報が必要であり，そのようなソースコードのコンテキスト情報の収集がデバッグプロセスの最初のステップであると報告されている．そのような，デバッグに役立つ欠陥モジュール予測を検討する．

デバッグ向けの欠陥モジュール予測として，局所的細粒度欠陥モジュール手法を提案する．これは，欠陥が含まれていると思われる故障モジュールが与えられた際に，その局所的なモジュール群から細粒度な欠陥モジュールを特定する．具体的には，あるディレクトリの中から欠陥メソッドを予測する．この局所的細粒度欠陥モジュール予測は，テストやレビュー向けの従来の欠陥モジュール予測と比べて主に 2 つの違いがある．

(1) 目的．従来の欠陥モジュール予測の目的は特定の欠陥に限定せず，全てのモジュールを対象に，優先的にテストやレビューすべきモジュールを順序付けることを目的としている．一方，提案するデバッグ向け欠陥モジュール予測は，ソフトウェア開発者が探索している局所的なモジュール内の特定の欠陥を特定することを目的としている．そのため，出力は局所的なモジュール（ディレクトリ）内の細粒度な欠陥モジュール（メソッド）のリストとなる．

(2) 予測範囲．Google で行われたケーススタディで指摘されているように，ソフトウェア開発者は特定のモジュールを担当することが多い [1]．そのため，従来の欠陥モジュール予測が出力する，ソフトウェア全体における欠陥モジュールリストは，ソフトウェア開発者にとって扱いにくい場合がある．それに対して提案手法は，特定の範囲に限定した，局所的な欠陥モジュール予測を行う．

局所的細粒度欠陥モジュール予測モデルの構築には，細粒度欠陥モジュール予測と同様にモジュールの開発履歴メトリクスを用いる [6]．問題となるのは，学習セットの収集方法である．あるスナップショットの全てのモジュールの開発履歴メトリクスを収集するのは既存の細粒度欠陥モ

[1] 奈良先端科学技術大学院大学
NAIST, Ikoma, Nara 630–0192, Japan
[a)] hata@is.naist.jp
[b)] akito-m@is.naist.jp
[c)] matumoto@is.naist.jp

ジュール予測と同様であり，これをベースラインとする．その他に，同じスナップショット内で，以前に欠陥があったディレクトリのみ，または，予測対象のディレクトリのみに限定した学習セットで予測モデルを構築する．加えて，複数のスナップショットを用いた場合の効果も議論する．

オープンソースソフトウェア Ant，ECF，PDE UI への適用実験から，最大で平均 0.62 から 0.81 の F 値を得た．これは，ベースラインと比べて 0.25 から 0.58 も高い．この高い予測精度によって，デバッグ時に欠陥メソッドを特定することに役立つと考えられる．

本稿の貢献は次の 2 つである．

- 初めてデバッグ向けの欠陥モジュール予測手法を提案した．従来のテストやレビュー向けの欠陥モジュール予測手法との違いを議論し，局所的細粒度欠陥モジュール予測手法を提案した．
- 提案手法をオープンソースソフトウェアプロジェクトで評価した．特に複数の学習セットの効果を従来手法の学習セットと比較した．

2. 局所的細粒度欠陥モジュール予測

2.1 アプローチ

従来の欠陥モジュール予測が，ある時点のスナップショットに存在する全てのモジュールを入力とし，全モジュール群から欠陥モジュールを予測するのに対し，局所的欠陥モジュール予測では，特定のディレクトリ内のモジュールを入力とし，その特定のモジュール群から欠陥モジュールを予測する．また，典型的な欠陥予測モデルがファイルレベルで欠陥モジュールを予測するのに対して，本稿ではメソッドレベルの欠陥モジュール予測を行う．細粒度で欠陥モジュールを予測することは，デバッグで特定の欠陥を発見する上で必要だと考えている．

予測モデルの構築において，従来の欠陥モジュール予測では，過去のある時点のスナップショットに存在する全てのモジュールを学習セットとしてきた．本稿では，局所的欠陥モジュール予測に対して次の 3 種類の学習セットを用意した．

All 過去のスナップショットに存在する全てのモジュール．従来の欠陥モジュール予測と同じであるため，これをベースラインとする．

Buggy 過去のスナップショットに存在する，欠陥メソッドを含むディレクトリのモジュール．**All** から欠陥を含まないディレクトリのモジュールを除外することによって，特に欠陥を含むメソッドの特徴を学習させることを意図している．

Local 過去のスナップショットに存在する，予測対象のディレクトリのモジュール．予測を行うディレクトリに存在するメソッドのみの学習セット．

また，複数のスナップショットを用いて学習セットを増やした場合の効果を調べるため，上述の 3 種の学習セットを，1 つのスナップショット，または，2 つのスナップショットで用意する．

表 2 対象プロジェクトのデータ
Table 2 Studied Projects.

プロジェクト	開始日	最新日	開発者数
ant	2000-01-13	2014-03-04	48
ecf	2004-12-03	2014-02-21	25
eclipse.pde.ui	2001-05-24	2014-02-20	27

2.2 メトリクス

本稿では，Java 言語で開発されたソフトウェアを解析対象とする．開発履歴メトリクス収集のためのメソッドの履歴分析には，細粒度履歴管理リポジトリ [9,10] を用いる．本リポジトリにより，細粒度な Java のメソッドの履歴の追跡が可能となり，ファイルレベルと同様の開発履歴メトリクスが，メソッドレベルでも計測できる．**表 1** に示す開発履歴メトリクスを計測する．これらは近年有用性が認められているものであり [11]，また，メソッドレベルの欠陥モジュール予測でもその有用性が示されている [6]．

2.3 予測モデル

予測モデルの構築には，欠陥モジュール予測において精度が高いといわれている Random Forest [12] を採用し，R の *randomForest* パッケージを用いて実現した．Random Forest は，モデル構築にランダム性が含まれるので，モデル構築と予測は 1000 回繰り返す．F 値は，1000 回の試行結果の中央値を求める．

3. 評価実験

3.1 対象プロジェクト

表 2 に本稿で対象とするオープンソースソフトウェアのプロジェクトを示す．Apache ソフトウェア財団で開発されている Apache Ant と，Eclipse 財団で開発されている ECF（Eclipse Communication Framework），PDE UI（Plug-in Development Environment UI）を選んだ．これらのプロジェクトは，長期間活発に開発されており，関連する障害リポジトリと版管理リポジトリが公開されている．これらの版管理リポジトリから作成した細粒度版管理リポジトリは，Kataribe プロジェクトとして公開している [13][*1]．

3.2 欠陥情報の取得

オープンソースソフトウェアプロジェクトのリポジトリにおいて，いつどのモジュールに欠陥が混入したかを特定するアルゴリズムとして，SZZ アルゴリズム [14] が広く使

[*1] http://kataribe.naist.jp/public

表 1 計測するメトリクス
Table 1 Metrics collection.

計測情報	メトリクス	説明
コード	LOC	対象版の行数
	AddLOC	最初の版からの追加行数
	DelLOC	最初の版からの削除行数
プロセス	ComNum	変更回数
	FixComNum	欠陥修正の変更回数
	HCM	プロセス複雑度メトリクス．History Complexity Metric HCM^{3s} [7]
	Period	存在期間（日単位）
	AvgPeriod	Period / ComNum
	MaxInterval	変更間の間隔の最大期間（日単位）
	MinInterval	変更間の間隔の最小期間（日単位）
	BugCouping	他のモジュールへの欠陥混入と同時に変更された回数
	NearBugs	同じファイルに存在する欠陥メソッド数
開発者	Ownership	開発者ごとの変更回数の割合をオーナシップとし，オーナシップの最大値 [8]
	DevTotal	そのモジュールを変更した開発者数
	DevMinor	オーナシップが 20%未満の開発者数
	DevMajor	オーナシップが 20%以上の開発者数

表 3 テストセットデータ
Table 3 Test set data.

(1) Ant

	ディレクトリ	欠陥なし	欠陥あり	メソッド総数
1	src/main/org/apache/tools/ant/taskdefs	1,911	33	1,944
2	src/main/org/apache/tools/ant	683	13	696
3	src/main/org/apache/tools/ant/util	458	9	467
4	src/main/org/apache/tools/ant/taskdefs/optional	255	9	264
5	src/main/org/apache/tools/ant/types/optional/image	42	8	50
	全て	7,705	124	7,829

(2) ECF

	ディレクトリ	欠陥なし	欠陥あり	メソッド総数
1	protocols/bundles/org.jivesoftware.smack/src/org/jivesoftware/smack	258	50	308
2	protocols/bundles/org.jivesoftware.smack/src/org/jivesoftware/smackx/packet	211	29	240
3	protocols/bundles/ch.ethz.iks.r_osgi.remote/src/main/java/ch/ethz/iks/r_osgi/impl	128	25	153
4	protocols/bundles/org.jivesoftware.smack/src/org/jivesoftware/smackx/filetransfer	98	17	115
5	protocols/bundles/org.jivesoftware.smack/src/org/jivesoftware/smackx	168	16	184
	全て	11,285	325	11,610

(3) PDE UI

	ディレクトリ	欠陥なし	欠陥あり	メソッド総数
1	ui/org.eclipse.pde.core/src/org/eclipse/pde/internal/core	524	144	668
2	ui/org.eclipse.pde.core/src/org/eclipse/pde/internal/ui/editor/plugin	716	57	773
3	ui/org.eclipse.pde.core/src/org/eclipse/pde/internal/ui/wizards/imports	166	52	218
4	ui/org.eclipse.pde.core/src/org/eclipse/pde/internal/core/builders	304	51	355
5	ui/org.eclipse.pde.core/src/org/eclipse/pde/internal/core/schema	260	41	301
	全て	16,204	1,242	17,446

われている．版管理リポジトリと障害リポジトリの情報を用いて，障害リポジトリに登録された障害の原因となった欠陥情報を取得する．

3.3 対象スナップショット

故障している（欠陥を含む）モジュールから，提案する局所的細粒度欠陥モジュール予測が欠陥を見つけることができるかを評価する．実際に欠陥メソッドを含むディレクトリから欠陥メソッドを予測する．各プロジェクトで，欠

表 4 学習セットデータ
Table 4 Training set data.

(1) Ant

	2009 年最初			2010 年最初		
ディレクトリ	欠陥なし	欠陥あり	メソッド総数	欠陥なし	欠陥あり	メソッド総数
1	1,748	86	1,834	1,856	59	1,915
2	645	25	670	668	21	689
3	416	15	431	426	14	440
4	252	10	262	258	9	267
5	42	8	50	42	8	50
全て	7,625	211	7,836	8,007	169	8,176

(2) ECF

	2009 年最初			2010 年最初		
ディレクトリ	欠陥なし	欠陥あり	メソッド総数	欠陥なし	欠陥あり	メソッド総数
1	201	38	239	258	50	308
2	181	28	209	211	29	240
3	111	23	134	127	26	153
4	81	17	98	98	17	115
5	150	15	165	168	16	184
全て	7,741	382	8,123	10,099	363	10,462

(3) PDE UI

	2009 年最初			2010 年最初		
ディレクトリ	欠陥なし	欠陥あり	メソッド総数	欠陥なし	欠陥あり	メソッド総数
1	482	161	643	503	155	658
2	706	64	768	713	59	772
3	114	53	167	145	49	194
4	281	58	339	288	62	350
5	258	43	301	259	42	301
全て	14,518	1,412	15,930	15,565	1,324	16,889

陥メソッド数が多い上位 5 つのディレクトリを対象とする．**表 3** に，予測を行うテストセットを示す．予測対象のモジュールは，2011 年の最初のスナップショットから用意する．**表 4** には学習セットを示す．学習セットは，2009 年と 2010 年の最初のスナップショットから用意する．スナップショットによって，全てのディレクトリでのメソッド総数は大きく変動しているが，各ディレクトリでのメソッド総数の変動は小さいことが分かる．

3.4 評価指標

予測モデルは欠陥の有無を予測する．評価指標には，Precision と Recall の調和平均である，F 値を用いる．Precision と Recall はトレードオフの関係にあるので，一方を高くすると他方が低くなる．F 値が高いことは，両方が高いことを意味する．

正例 (Positive) を正しく予測できたものを True Positive (TP)，負例を正と予測したものを False Positive (FP)，負例 (Negative) を正しく予測できたものを True Negative (TN)，正例を負と予測したものを False Negative (FN) と呼ぶ．

Precision は，正と予測されたもののうち実験に正であるものの割合である．

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall は，全ての正例のうち正しく正と予測されたものの割合である．

$$\text{Recall} = \frac{TP}{TP + FN}$$

F 値は，Precision と Recall の調和平均である．

$$\text{F} - \text{measure} = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

F 値は，0 から 1 の範囲を取り，値が大きいほどよい予測モデルといえる．

4. 結果

2.1 節で説明したように，従来手法と同様である，全てのモジュールを用いる学習セット（All）をベースラインとし，欠陥を含むディレクトリのモジュールを用いる学習セット（Buggy）と，予測を行うディレクトリのモジュールを用いる学習セット（Local）の予測結果を比較する．

表 5 2010 年の学習セットでの予測結果
Table 5 Prediction Results with Training sets of 2010.

(1) Ant

	% 欠陥あり	All（ベースライン）	Buggy	Local
1	1.7	0.48	**0.50**	**0.59**
2	1.9	0.38	**0.47**	**0.56**
3	1.9	0.36	0.36	0.36
4	3.4	0.50	0.50	**0.94**
5	16.0	-	-	**1.00**
平均		0.34	**0.37**	**0.69**

(2) ECF

	% 欠陥あり	All（ベースライン）	Buggy	Local
1	16.2	0.08	**0.52**	**0.76**
2	12.1	-	-	**0.73**
3	16.3	-	**0.48**	**0.57**
4	14.8	-	**0.35**	**0.45**
5	8.7	0.12	**0.22**	0.12
平均		0.04	**0.31**	**0.53**

(3) PDE UI

	% 欠陥あり	All（ベースライン）	Buggy	Local
1	21.6	0.44	0.44	**0.79**
2	7.4	0.69	0.69	**0.77**
3	23.9	0.61	0.61	**0.77**
4	14.4	0.45	0.45	**0.75**
5	13.6	0.59	0.59	**0.73**
平均		0.56	0.56	**0.76**

表 5 は，テストセットの 1 年前の 2010 年のデータを学習セットとした予測結果である．いずれの学習セットにおいても 1000 回の試行で得られる F 値の値はほぼ等しく，分散は小さかった．ベースラインでは，PDE UI プロジェクトでは，平均 0.56 の F 値であるが，ECF プロジェクトや Ant プロジェクトでは，正しく予測できなかった（Recall が 0 であった）．ベースラインより高い値の F 値を太字で示す．学習セットをフィルタリングした Buggy と Local は，ベースライン以上の F 値を得た．特に Local のデータセットは，平均の F 値が 0.53 から 0.76 と高い予測精度であった．欠陥ありのモジュールの割合が 2%から 20%程度と小さいデータセットにかかわらず，高い予測精度といえる．このことから，局所的欠陥モジュール予測には全てのモジュールではなく，予測対象モジュールに近いモジュールのみを用いた学習セットの方が高い予測精度となることが分かる．学習セットのデータ量は小さくなるが，予測対象モジュール内にある特徴をよりよくモデル化できていると考えられる．

次に，複数のスナップショットを用いる効果を調査するため，2 つのスナップショットを学習セットとして予測モデルを構築する．ベースラインの学習セット，2009 年と 2010 年の All 学習セット，2009 年と 2010 年の Local 学習セットの予測結果を比べる．2 年分の学習セットによって予測精度は向上している．特に Local 学習セットでは，ベースラインと比べると，0.25 から 0.58 も F 値が向上した．

表 6 複数のスナップショット（2009 年と 2010 年）の予測結果
Table 6 Prediction Results with Training sets of 2009 and 2010.

(1) Ant

	ベースライン	All（2 年）	Local（2 年）
1	0.48	**0.57**	**0.59**
2	0.38	**0.44**	**0.73**
3	0.36	0.36	**0.46**
4	0.50	**0.54**	**1.00**
5	-	-	**1.00**
平均	0.34	**0.38**	**0.76**

(2) ECF

	ベースライン	All（2 年）	Local（2 年）
1	0.08	**0.18**	**0.76**
2	-	-	**0.85**
3	-	-	**0.86**
4	-	**0.21**	**0.52**
5	0.12	0.12	0.12
平均	0.04	**0.10**	**0.62**

(3) PDE UI

	ベースライン	All（2 年）	Local（2 年）
1	0.44	**0.51**	**0.80**
2	0.69	**0.73**	**0.82**
3	0.61	**0.72**	**0.82**
4	0.45	**0.48**	**0.83**
5	0.59	**0.72**	**0.80**
平均	0.56	**0.63**	**0.81**

5. 議論

5.1 妥当性への脅威

本稿では，欠陥が含まれていると思われる故障モジュールを特定のディレクトリとして評価したが，これはデバッグを想定した初期モジュール集合としては適切でない可能性がある．また，実験ではディレクトリ中のサブディレクトリ内のファイルを無視したが，ディレクトリの親子関係についてはさらに考慮する必要がある．

評価実験では，3 つのオープンソースソフトウェアプロジェクトへの適用結果を報告した．そのため，本結果の一般性には妥当性への脅威がある．今後は，商業ソフトウェアを含む多様なソフトウェアプロジェクトへの適用が必要である．

欠陥情報の収集は，版管理リポジトリと障害リポジトリの情報を結びつけることで実現している．自動的にデータ収集しているため，欠陥の誤特定や見逃しが起こることが考えられる．これが予測結果に影響する可能性がある．

5.2 今後の課題

本稿の結果から，局所的細粒度欠陥モジュール予測には，予測対象モジュールの複数スナップショットデータを用いることで，高い予測精度が得られることが分かった．本稿の実験のように，2 年分のスナップショットデータを用意するには，プロジェクトが 2 年より長い期間開発履歴を蓄積していることが必要である．また，予測対象のディレクトリが過去のスナップショットに無い場合，Local 学習セットを用意できない．本提案は，ある程度長い期間開発が進み，開発が安定して十分な学習データが用意できる場合には有効と考えられる．しかし，そのような開発履歴が十分蓄積できていない場合，どう学習セットを用意するかは今後の課題である．

また，提案手法がデバッグにどう貢献するかを明らかにするため，実際の開発に適用するケーススタディを行う必要がある．その際，開発者がどの程度の粒度で故障モジュールを推定できるかや，予測された欠陥モジュールから欠陥を効率よく発見できるかなどを分析することが課題である．

評価実験の結果得られた，Local 学習セットによる予測モデルが All 学習セットによる予測モデルより精度がよいという知見は，従来の欠陥予測における，プロジェクト外データを使うよりプロジェクト内データで予測モデルを構築する方が精度がよいという知見とも適合する，プロジェクト外のデータを有効活用するためのクロスプロジェクト欠陥予測にはいくつかの研究がある [15, 16]．これらの成果を活用することで，局所的細粒度欠陥モジュール予測の精度を高めることも今後の課題となる．

6. おわりに

本稿では，デバッグ向けの欠陥モジュール予測として，特定のディレクトリから欠陥メソッドを予測する，局所的細粒度欠陥モジュール予測手法を提案した．過去のスナップショット中の全てのモジュールを学習セットに用いる，従来手法と同様の予測モデルと比べて，予測対象ディレクトリ中のモジュール 2 年分を学習セットとした予測モデルは，F 値で 0.25 から 0.58 向上し，0.62 から 0.81 という高い予測精度となった．この高い予測精度によって，デバッグ時に欠陥を効率よく発見できると期待できる．

謝辞 本研究は JSPS 科研費 26540029 の助成を受けた．また，独立行政法人情報処理推進機構 技術本部 ソフトウェア高信頼化センター（SEC: Software Reliability Enhancement Center）が実施した「2013 年度ソフトウェア工学分野の先導的研究支援事業」の支援を受けた．

参考文献

[1] Lewis, C., Lin, Z., Sadowski, C., Zhu, X., Ou, R. and Whitehead Jr., E. J.: Does bug prediction support human developers? findings from a google case study, *Proc.* ICSE '13, pp. 372–381 (2013).

[2] Monden, A., Hayashi, T., Shinoda, S., Shirai, K., Yoshida, J., Barker, M. and Matsumoto, K.: Assessing the Cost Effectiveness of Fault Prediction in Acceptance Testing, *IEEE Trans. Softw. Eng.*, Vol. 39, No. 10, pp. 1345–1357 (2013).

[3] Czerwonka, J., Das, R., Nagappan, N., Tarvo, A. and Teterev, A.: CRANE: Failure Prediction, Change Analysis and Test Prioritization in Practice – Experiences from Windows, *Proc.* ICST '11, pp. 357–366 (2011).

[4] Hall, T., Beecham, S., Bowes, D., Gray, D. and Counsell, S.: A Systematic Literature Review on Fault Prediction Performance in Software Engineering, *IEEE Trans. Softw. Eng.*, Vol. 38, No. 6, pp. 1276–1304 (2012).

[5] Layman, L., Diep, M., Nagappan, M., Singer, J., Deline, R. and Venolia, G.: Debugging Revisited: Toward Understanding the Debugging Needs of Contemporary Software Developers, *Proc.* ESEM '13, pp. 383–392 (2013).

[6] Hata, H., Mizuno, O. and Kikuno, T.: Bug prediction based on fine-grained module histories, *Proc.* ICSE '12, pp. 200–210 (2012).

[7] Hassan, A. E.: Predicting faults using the complexity of code changes, *Proc.*, ICSE '09, pp. 78–88 (2009).

[8] Bird, C., Nagappan, N., Murphy, B., Gall, H. and Devanbu, P.: Don't touch my code!: examining the effects of ownership on software quality, *Proc.* ESEC/FSE '11, pp. 4–14 (2011).

[9] 畑　秀明，水野　修，菊野　亨：リポジトリ再構築によるメソッドトレーサビリティの実現，ソフトウェアエンジニアリングシンポジウム 2010 (SES2010)，pp. 57–62 (2010).

[10] Hata, H., Mizuno, O. and Kikuno, T.: Historage: Fine-Grained Version Control System for Java, *Proc.* IWPSE-EVOL '11, pp. 96–100 (2011).

[11] 畑　秀明，水野　修，菊野　亨：不具合予測に関するメトリクスについての研究論文の系統的レビュー，コンピュータソフトウェア，Vol. 29, No. 1, pp. 106–117 (2012).

[12] Lessmann, S., Baesens, B., Mues, C. and Pietsch, S.: Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings, *IEEE Trans. Softw. Eng.*, Vol. 34, pp. 485–496 (2008).

[13] Fujiwara, K., Hata, H., Makihara, E., Fujihara, Y., Nakayama, N., Iida, H. and Matsumoto, K.: Kataribe: A Hosting Service of Historage Repositories, *Proc.* MSR '14, pp. 380–383 (2014).

[14] Śliwerski, J., Zimmermann, T. and Zeller, A.: When do changes induce fixes?, *Proc.* MSR '05, pp. 1–5 (2005).

[15] Zimmermann, T., Nagappan, N., Gall, H., Giger, E. and Murphy, B.: Cross-project defect prediction: a large scale experiment on data vs. domain vs. process, *Proc.* ESEC/FSE '09, pp. 91–100 (2009).

[16] Zhang, F., Mockus, A., Keivanloo, I. and Zou, Y.: Towards Building a Universal Defect Prediction Model, *Proc.* MSR 2014, pp. 182–191 (2014).