

An Empirical Study on Maintainable Method Size in Java

Shaiful Alam Chowdhury, Gias Uddin, Reid Holmes

19th International Conference on
Mining Software Repositories (MSR2022)

発表者：松尾 和輝

背景

- ソフトウェアを安全に稼働させる為には、メンテナンスが必要不可欠である
- 将来のメンテナンス工数を見積もる為に、変更優先度やバグ優先度といった保守指標がよく用いられている
 - バグや変更が起こりやすいコンポーネントは保守コストが高い
- しかし、これらの指標を測定する為には、一度リリースされなければならない[78]

背景

- そこで、コードメトリクスと変更優先度、バク優先度の関連性を調査することで、間接的にメンテナンス工数を見積もる研究が行われている
- 本研究では、メソッドサイズに着目し、メソッドサイズとメンテナンス工数との関係を調査する

リサーチクエスチョン

- RQ1: SLOC はメソッドレベルの変更性, バグ優先度と関係があるか？
- RQ2：メンテナンス可能なメソッドサイズとは？
- RQ3：開発者は大規模なメソッドを分解する必要があるか？
- RQ4: サイズを制御することで他の品質メトリクスが制御されるか？

データセット

- 5つの異なる先行研究によって使用された, 49のGitHubプロジェクトを選択

Repository	# Methods	# Revisions (avg)	# Getters-Setters	Age $\geq$ 2 Years	# Used Methods	Snapshot
hadoop	70,524	1.8	11,219	63,378	53,729	4c5cd7
elasticsearch	63,253	3.7	8,875	39,085	34,641	92be38
flink	38,482	1.9	6,790	19,967	15,792	261e72
presto	37,639	1.9	8,520	26,785	20,766	bb20eb
lucene-solr	37,600	1.6	5,625	29,287	24,641	b457c2
...	...	...	...	...	...	...
mockito	1,526	4.2	144	1,221	1,113	077562
cucumber-jvm	1,222	2.7	170	435	374	b57b92
commons-io	1,149	3.1	85	952	884	11f0ab
vraptor4	934	1.7	132	933	801	593ce9
junit4	879	3.1	56	856	801	50a285
Total	785,606				520,874	

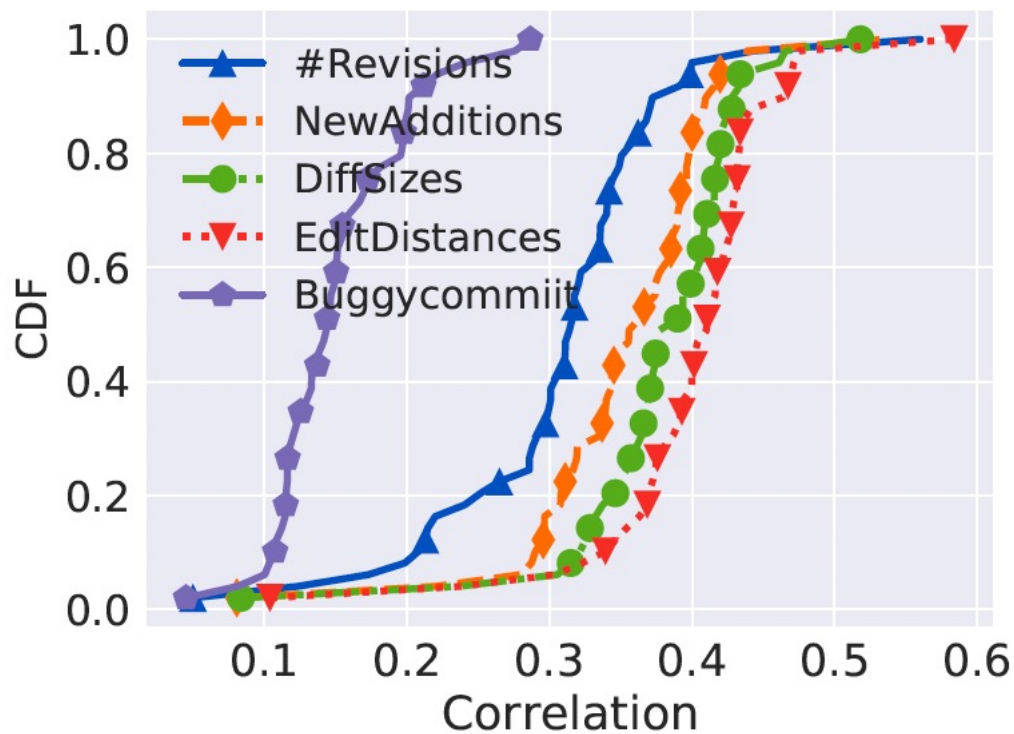
- Getter/Setter, 2年未満のメソッドは除外

RQ1：SLOC はメソッドレベルの変更性, バグ優先度と相関があるか？

- SLOCと5つの保守指標のKendall の τ 相関係数の分布を調査
 - 変更優先度
 - Revisions：メソッドが修正された回数
 - NewAdditions：追加行数
 - Diffsizes：変更行数
 - EditDistances：追加、削除、修正された文字数
 - バグ優先度
 - BuggyCommit：バグ修正コミット数

RQ1：SLOC はメソッドレベルの変更性， バグ優先度と相関があるか？

- SLOC は，メソッドレベルの粒度において，変更およびバグの発生しやすさと相関がある



RQ2：メンテナンス可能なメソッドサイズとは？

- Alves ら[3]による， 6stepの体系的アプローチを利用

- Step1：SLOCの測定

$$P_{M_{sloc}^1}^1 = 10, P_{M_{sloc}^2}^1 = 20, P_{M_{sloc}^3}^1 = 10$$

- Step2：重み付け

$$P_{M_{weight}^1}^1 = P_{M_{sloc}^1}^1 / (P_{M_{sloc}^1}^1 + P_{M_{sloc}^2}^1 + P_{M_{sloc}^3}^1) = 0.25$$

$$\text{Similarly, } P_{M_{weight}^2}^1 = 0.5, P_{M_{weight}^3}^1 = 0.25$$

RQ2：メンテナンス可能なメソッドサイズとは？

- Step3：グループ化

$$P_{M_{sloc}^i}^p = P_{M_{sloc}^j}^p = P_{M_{sloc}^k}^p .$$

The entity aggregation of this method group

$$= P_{M_{weight}^i}^p + P_{M_{weight}^j}^p + P_{M_{weight}^k}^p .$$

In our case with P^1 , the entity aggregation for the first method group with SLOC 10

$$= P_{M_{weight}^1}^1 + P_{M_{weight}^3}^1 = 0.5,$$

and for the second method group with SLOC 20 $= P_{M_{weight}^2}^1 = 0.5$

$slocs = [10, 20]$, and $aggr = [0.5, 0.5]$

RQ2：メンテナンス可能なメソッドサイズとは？

- Step4：正規化

$norms = [0.25, 0.25]$

- Step5：すべてのプロジェクトでStep1からStep4を行う
- Step6：アルゴリズムから x_axis , y_axis を求める

$x_axis = [10, 20], y_axis = [0.35, 0.65]$

- x_axis を昇順に（ y_axis もそれに合わせて）ソートして、累積折れ線グラフを作成

アルゴリズム

Algorithm 1: Generating x_axis and y_axis to produce the final cumulative line chart.

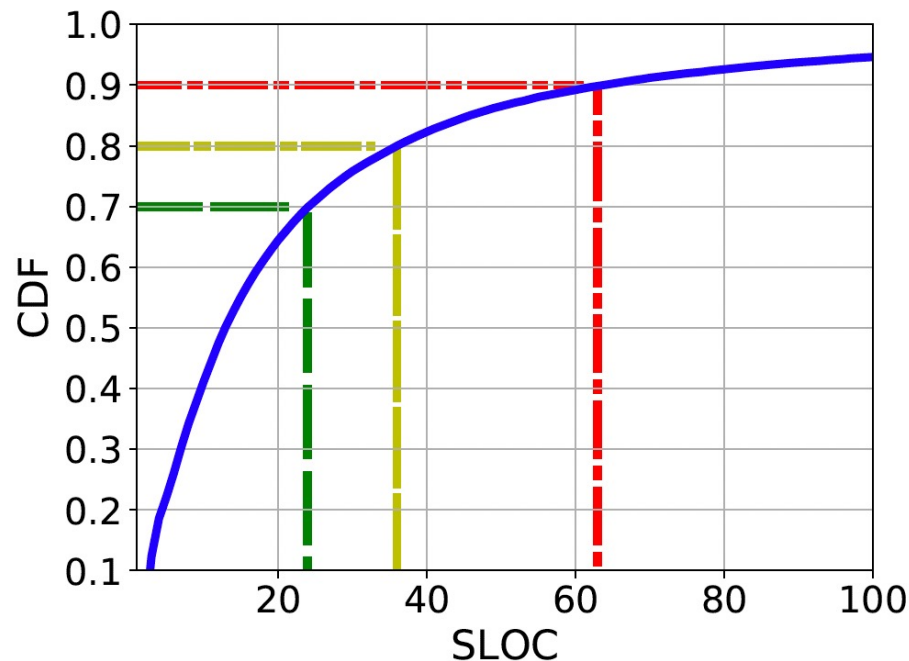
input : $list_slocs, list_norms$

output: x_axis, y_axis

```
1  $x\_axis \leftarrow []$ ;
2  $y\_axis \leftarrow []$ ;
3  $counter \leftarrow 0$  ;
4 for  $i \leftarrow 1$  to  $length(list\_slocs)$  do
5    $slocs \leftarrow list\_slocs[i]$ ;
6    $norms \leftarrow list\_norms[i]$ ;
7   for  $j \leftarrow 1$  to  $length(slocs)$  do
8     if  $slocs[j]$  not in  $x\_axis$  then
9        $x\_axis[counter] \leftarrow slocs[j]$ ;
10       $y\_axis[counter] \leftarrow norms[j]$ ;
11       $counter \leftarrow counter + 1$ ;
12    end
13    else
14       $index = locate(slocs[j], x\_axis)$ ;
15       $y\_axis[index] \leftarrow$ 
16         $y\_axis[index] + norms[j]$ ;
17    end
18 end
19 return  $x\_axis, y\_axis$ ;
```

RQ2：メンテナンス可能なメソッドサイズとは？

- 24SLOC以下で記述されたメソッドが7割であった
 - ほとんどのメソッドは24SLOC以下で記述できる



RQ3：開発者は大規模なメソッドを分解する必要があるか？

- SLOCが24以上のメソッドAと， SLOCが24以下のメソッドを， Aと同じSLOCにマージしたメソッドBを， 5つの保守指標に関してCliff's Deltaで比較

Individual Method	SLOC	#revisions	Selected
M1	40	3	Yes
M2	36	4	Yes
M3	28	1	Yes
M4	80	3	No
Merged Methods	Total SLOC	Total #revisions	Selected
M5, M6	40	2	Yes
M5, M7	28	1	Yes
M7, M9	16	0	No
M5, M7, M9	36	1	Yes

RQ3：開発者は大規模なメソッドを分解する必要があるか？

- ほとんどの場合、マージされたメソッドは、個別のメソッドよりもコストが低い
 - 大規模なメソッドを分解しても、ソフトウェアシステムの保守工数が増加することはない

Project	# Samples	# Revisions	# Additions	DiffSize	EditDistance	Bugs
hadoop	1304	N	N	N	N	N
weka	1274	N	N	N	N	N
hibernate-orm	841	N	N	N	N	N
hbase	742	N	N	N	N	N
docx4j	498	N	N	N	N	N
hazelcast	493	N	S	S	S	N
netty	445	N	N	N	N	N
jgit	413	S	S	S	S	S
wicket	412	N	N	N	N	N
voldemort	184	N	N	N	N	N
openmrs-core	170	N	N	N	N	N
wildfly	163	N	N	N	N	N
mongo-java-driver	144	S	S	S	S	N
checkstyle	137	S	S	S	S	N
javaparser	105	N	N	N	N	N
hibernate-search	78	N	N	N	N	N
spring-boot	72	N	N	N	N	N
facebook-android-sdk	72	N	N	N	N	N
lombok	50	N	N	N	N	N
jna	45	S	S	S	S	N
hector	45	S	S	S	S	N
okhttp	41	S	S	S	S	S
commons-io	20	N	N	S	N	N

N：無視できる

S：小さい

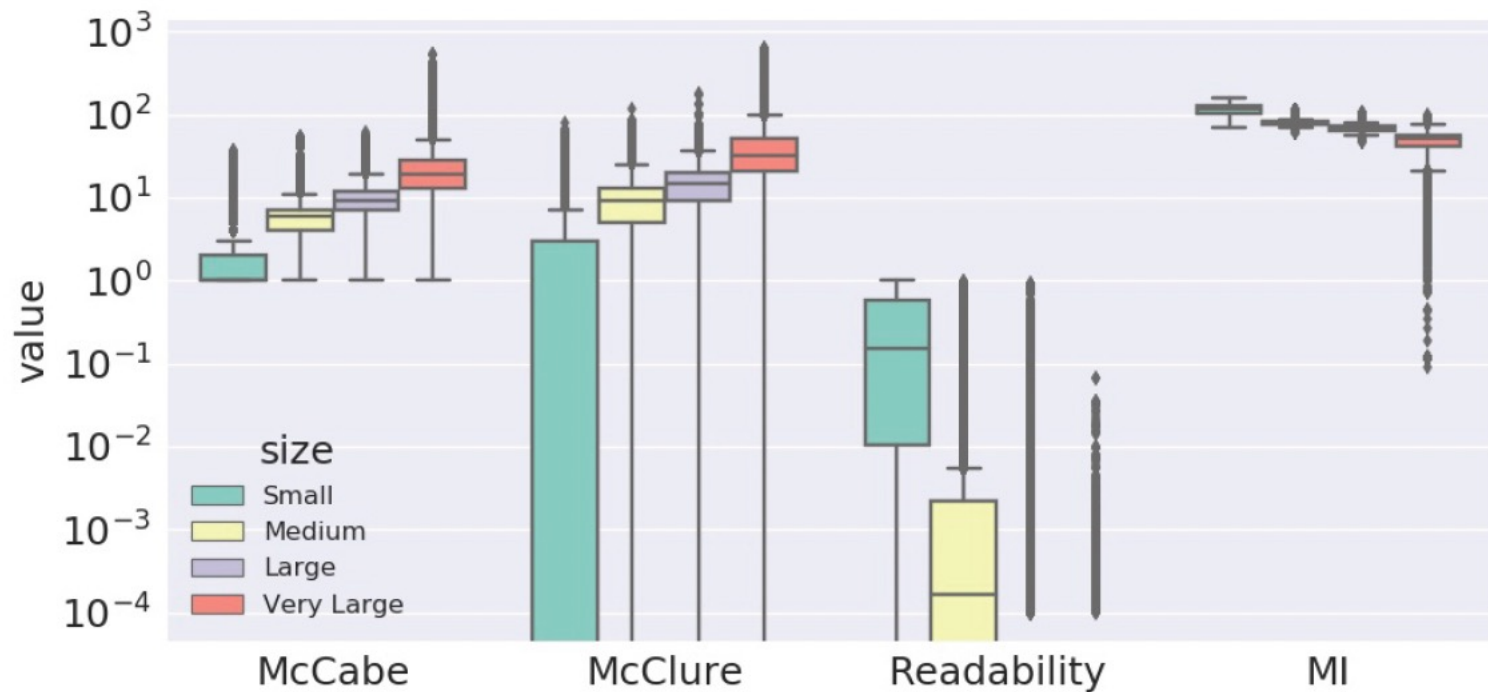
青色は、マージされたメソッドの方がその指標に対する値が低いことを示す

RQ4: サイズを制御することで他の品質メトリクスが制御されるか？

- メソッドサイズを制御することで、以下の3つの指標も制御できるか調査
 - Testability
 - McCabe, McClureメトリクス
 - Readability
 - Buse ら[15]が実装した一般に公開されている可読性ツールを利用
 - Maintainability Index
$$171 - 5.2 * \ln(\text{Halstead Volume}) - 0.23 * (\text{Cyclomatic Complexity}) - 16.2 * \ln(\text{Lines of Code})$$

RQ4: サイズを制御することで他の品質メトリクスが制御されるか？

- メソッドサイズを制御することで、3つの指標も制御できる



まとめ

- 目的
 - メソッドサイズとメンテナンス工数との関係を調査
- 結果
 - SLOC はメソッドレベルの変更性，バグ優先度と相関がある
 - メソッドのサイズは，分解を行い24SLOC以下に抑えることで，保守コストを抑えることができる
 - メソッドサイズを制御することで， Testability， Readability， Maintainability Indexも制御することができる

所感

- 所々説明がなくて、あまりよくわからなかった
- どうしても24SLOC以上必要な実装もあるのでは
- メソッドサイズを抑えることで、保守コストを抑えられるのは興味深い