|  |  |
| --- | --- |
| Продукт | Метрики кода |
| Версия | 1.0 |
|  | Всего 17 страниц |
|  |

Анализ качества кода в репозиториях

|  |  |  |  |
| --- | --- | --- | --- |
| проект | SWE | дата | 2020-05-01 |
| проверка | PL | дата | 2020-06-01 |
| утверждение | PL | дата | 2020-07-01 |



Корпорация Huawei

все права защищены

（DVP05T04 V2.8 / для внутреннего использования）

История изменений

| Дата | Версия | CR | Часть изменений | Описание изменений | Автор |
| --- | --- | --- | --- | --- | --- |
| 2020-05-01 | 1.00 |  |  | Документ для презентации | z00534010 |
| 2020-05-08 | 1.01 |  |  | Расширен список используемой литературы, добавлено описание использованной литературы | z00534010 |
| 2020-05-15 | 1.02 |  |  | Улучшен стиль написания статьи: добавлены формулы, пояснения | z00534010 |
| 2020-05-22 | 1.03 |  |  |  | z00534010 |

Оглавление

1 Аннотация 5

2 Введение 5

3 Возможность интеграции представленной методики в существующие системы управления и сбора программного кода (Build pipelines) 6

4 Метрики используемые для анализа качества кода. 6

5 Предпосылки создания метрики 7

6 Количественная оценка качества исходного кода репозитория 8

7 Поиск взаимосвязей между метриками репозитория. 9

8 Распределение статистики качества кода репозитория в зависимости от языка программирования. 10

9 Гистограмма распределения статистики качества кода репозитория. 12

10 Поиск взаимосвязей между волатильностью репозитория и качеством кода. 12

11 Заключение 13

Литература 14

Анализ качества кода в репозиториях

Ключевые слова： метрики кода

Абстракт： Введение новой метрики анализа кода

Принятые сокращения：

| Аббревиатура | Полное имя | Описание |
| --- | --- | --- |
| CK метрик | Chidamber and Kemerer метрика | Метрики Объектно-ориентированного программирования, представленные авторами в работах 13, 14 |
|  |  |  |
|  |  |  |
|  |  |  |

# Аннотация

В настоящее время рещением одной из основных задач в обеспечении качества программных разработок как крупных аутсорисинговых компаний, так и фриланс программистов является обеспечение качества исходного программного кода. В статье представлена статистика по имеющимся публичным репозиторям, определены виды ошибок, снижающих качество кода, предложены пути улучшения.

Остро стоит проблема определения наиболее эффективной методики, которая может быть применена для анализа исходного кода независимо от используемого языка программирования. В качестве одной из методики была выбрана и рассмотрена методика, основанная на расчете показателя волатильности исходного кода репозитория, которая позволяет решить проблему объективной оценки качества исходного кода. Для проверки работоспособности выбранной методики получены качественные показатели, указывающие на снижение качества исходного программного кода путем анализа базы данных общедоступных репозиториев github.

На основе полученных исходных данных проанализирована эффективность применения методики, использующей показатель волатильности репозитория. Также показано, что данная методика применима для анализа качества кода произвольного github репозитория. Результаты применения методики показывают, что, основываясь на показателе волатильности репозитория можно обнаружить блоки программного кода, значительно снижающих качество программного кода репозитория, что позволяет своевременно проводить меры по его улучшению.

# Введение

Научная новизна данной работы состоит в том, что представленная модернизированная методика анализа исходного программного кода, основанная на показателях волатильности, представляет собой объективную оценку качества кода. Как показано далее, многие имеющиеся на рынке методики оценки качества программного кода не могут полностью решить задачу объективного анализа качества исходного кода, выполненного на произвольном языке прграммирования.

# Возможность интеграции представленной методики в существующие системы управления и сбора программного кода (Build pipelines)

Имеющиеся общеиспользуемые системы управления версиями исходного кода, проверки и сборки готового решения поддерживают внедрение внешних процедур в качестве одного из исполняемых шагов финальной сборки. Исходя из этого необходимо определить механизмы интеграции для полной автоматизации процесса. Рассмотрим процесс интеграции методики расчета показателей волатильности в такие сложные по своему построению системы как Jenkins и Gitlab. При этом основой будет являться автоматизация расчета. В Gitlab используем механизм CI/CD Pipelines, где одним из шагов сборки будет проверка качества кода после добавления новой фиксации (commit). В случае превышения полученного значения над заданным сборка может быть отклонена.

# Метрики используемые для анализа качества кода.

После создания модели оценки качества исходного кода возникает вопрос тестирования, а именно сбор информации о качестве исходного кода имеющихся проектов. Для решения этой задачи было загружено более 1000 репозиториев с github и для каждого из них проанализированы вручную методом экспертной оценки следующие показатели:

* 1. Соответствие правилам, например:
     1. Синтаксические правила. К ним можно отнести стиль именования переменных (camelCase, через подчеркивание), констант (uppercase), методов, стиль написания фигурных скобок и нужны ли они если в блоке только одна строка кода. Когда программист пишет код, он его легко читает, потому что знает свой собственный стиль. Но стоит ему дать код где используется венгерская нотация и скобки с новой строки, ему придется тратить дополнительное внимание на восприятие нового стиля. Особенно трудна для понимания ситуация, когда несколько совсем разных стилей используются в одном проекте или даже модуле.
     2. правила поддержки кода — правила, которые должны сигнализировать что код слишком сложный и его будет трудно сопровождать. К примеру, индекс сложности метода или класса слишком большой или слишком много строк кода в методе, наличие дубликатов в коде или “magic numders”.
     3. очистка и оптимизация кода — самые простые правила в том смысле, что редко кто-то будет утверждать что выражения очень нужны, даже когда они нигде не используются. Сюда можно отнести лишние импорты, переменные и методы которые уже не используются, но по какой-то причине их оставили в наследство.
  2. Дубликаты: отображает насколько легко можно будет вносить изменения в код, рассчитывается в процентах как соотношение строк дубликатов ко всем строкам кода, чем меньше дубликатов тем лучше качество кода.
  3. Комментирование рассчитывается по двум важным метрикам:
     1. отношение комментариев ко всему коду — из этой метрики можно сделать вывод насколько детальные комментарии и насколько они могут быть полезными.
     2. комментирование публичных методов — отношение комментированных публичных методов к общему их количеству. Количество публичных методов без комментария должно стремится к нулю.
  4. Покрытие тестами: считывается как отношение количества покрытых тестами элементов кода к количеству всех существующих, выделяют следующие типы покрытия:
     1. покрытие файлов — файл покрыт если тест попал в файл и исполнил хотя бы одну строку кода из файла.
     2. покрытие классов — аналогично с покрытием файлов, только покрытие классов.
     3. покрытие методов — тот же способ исчисления метрики.
     4. покрытие строк — одна из наиболее используемых метрик по покрытию. Тот же способ исчисления.
     5. покрытие ветвлений — тот же способ, соответственно за элемент берется ветвление. Добиться хорошего показателя по этой метрики стоит наибольших усилий.
     6. суммарное покрытие — метрика покрытия при которой в расчетах принимается во внимание не один элемент, а несколько. Наиболее часто используют суммарное покрытие строк и ветвлений.

# Предпосылки создания метрики

Технической предпосылкой для создания метрики является возможность использования интефейса git: команда Git Log предоставляет информацию о каждом изменении, сделанном в программном обеспечении разработчиками в течение всего срока жизни проекта, таким образом появилась возможность анализировать историю изменений программного кода. Фундаментальной предпосылкой для создания метрики явилось наличие огромного количества исследований в области поиска взаимосвязей между метриками и качеством программного кода. Предлагается рассмотреть новую метрику, принадлежащую семейству метрик изменений исходного кода, анализирующую историю изменений кода в репозитории [6].

Изменение в системе управления версиями кода, такой как Git, представляет собой атомарную модификацию исходного кода в репозитории. Каждая система управления версиями кода предоставляет инструмент получения полной истории изменений исходного кода. В целях получения материала для данной работы использовался интерфейс git log. Было доказано ([4], и с более глубокой историей изменений в [16]), что метрики изменений исходного кода, полученные из истории репозиториев систем управления версиями, являются наиболее надежным индикатором дефектов в современных исследованиях программной инженерии.

Одной из простейших метрик семейства является метрика взбалтывания кода, использующая все измененные строки кода (добавленные, измененные и удаленные) за все время существования проекта [8]. В [5] показано как данная метрика в совокупности с метрикой сложности и других метрик активных разработок связана с уязвимостью программного обеспечения. Дальнейшая модификация этой метрики предложена [7], принимающая во внимание социально – технические аспекты: введены коэффициенты взаимосвязи между разработчиками в контексте совместной работы.

В [11] сделан сравнительный анализ из 17 метрик для понимания их эффективности в предсказании дефектов программного обеспечения.

В [12] представлен детальный анализ существующих метрик и предлагается новая, показывающая насколько близки друг к другу разработчики и насколько крепко они формируют пары между собой, вычисляемая через историю фиксаций кода Git репозиториев.

В отличие от метрик исходного кода, объектно-ориентированных метрик и CK метрик ([13], [14]), метрики изменения не касаются содержимого исходных файлов, а работают с историей изменений кода с течением времени. Одним из неиспоримых преимуществ использования метрик изменения по сравнению с метриками исходного кода в том, что они не зависят от конкретного языка программирования.

В [15] предпринята попытка построения инструмента сбора информации о изменениях файлов.

Предлагаемая в данной работе метрика показывает насколько велика разница между часто и редко обновляемыми файлами одного репозитория. Она подтверждает гипотезу, что большое количество старого кода (редко изменяемых файлов) является индикатором проблемы трансформации знаний о проекте.

# Количественная оценка качества исходного кода репозитория

Первый шаг – получение данных о количестве изменений каждого файла репозитория [2], [3]. Составим числовой ряд:

,

где f0, f1,. fn - арифметические суммы количества присутствий каждого файла во всех фиксациях (commits) исходного кода за время его существования. В случаях изменения пути и имени файла, все предылущие изменения относятся к новому файлу, таким образом не происходит ни единой потери информации о вносимых изменениях.

Далее считаем медиану (value) и дисперсию полученного числового ряда [1]. Медиана — число, характеризующее выборку, такое, что половина из элементов выборки больше него, а другая половина меньше. В более общем случае медиану можно найти, упорядочив элементы выборки по возрастанию или убыванию и взяв средний элемент, если в выборке чётное число элементов, используется полусумма двух соседних средних значений. Для расчета дисперсии рассчитаем среднее значение ряда F:

, а затем дисперсию:

Отношение дисперсии к медиане называется характеристикой волатильности репозитория **μ =** .

# Поиск взаимосвязей между метриками репозитория.

Автором проведена работа по сбору и анализу следующих общедоступных характеристик по более чем 1000 репозиториев наиболее популярных языков программирования:

* + 1. Go
    2. Java
    3. Clojure
    4. Scala
    5. Javascript
    6. Python
    7. Haskell
    8. C
    9. Cpp
    10. C#
    11. Erlang
    12. F#
    13. R
    14. Ruby
    15. Kotlin
    16. Typescript
    17. Elixir

Данное количество репозиториев позволяет с высокой степенью доверия рассчитать статистические взаимосвязи между собранными числовыми рядами. GitHub предоставляет следующие метрики по репозиториям:

1. forks – количество ответвлений проекта, используется разработчиками для работы в собственной ветке и последующих запросов на внесение изменений в исходный репозиторий. Чем больше подобных ответвлений, тем как правило больше разработчиков принимают участие в проекте и работа над репозиторием идет более активно;
2. issues – количество предлагаемых доработок, найденных багов на текущий момент в проекте. Чем больше данный показатель, тем, как правило, более активно идет работа над репозиторием;
3. size – размер репозитория в килобайтах. Чем больше данный показатель, тем, как правило, больше файлов он содержит и тем более зрелым является проект;
4. stars – количество звезд в репозитории. Чем больше данный показатель, тем, как правило, большим сообществом он уже используется и тем более зрелым является проект;
5. subscribers – подписчики проекта, как правило, люди, наиболее заинтересованные в получении новых версий и текущем статусе проекта. Чем больше данный показатель, тем, как правило, проект более используемый на практике и более зрелый.

Получена следующая матрица корреляций между характеристикой волатильности репозитория (μ) и другими характирестиками Git репозитория:



Как следует из приведенных данных статистики по более чем 1000 репозиториев с открытым исходным кодом, ни одна из общедоступных характеристик репозитория не коррелирует с характеристикой волатильности репозитория. Следовательно, μ является уникальной характеристикой.

# Распределение статистики качества кода репозитория в зависимости от языка программирования.

Общеизвестно, что качество кода в таких функциональных языках как Elixir(Erlang), Scala, Kotlin заметно выше, чем в императивных языках с низким порогом входа для новичков как Ruby, Javascript, что и показывает простой математический расчет:



JavaScript, Ruby имеют большую долю репозиториев, однако их показатель волатильности выше среднего

Из диаграммы видно, что Python является одним из самых распространенных языков, имеющим статистически наилучшее качество среди аналогов.

# Гистограмма распределения статистики качества кода репозитория.



Данная гистограмма помогает с выбором критического значения, до которого качество кода может быть приемлемо. При значениях характеритики качества кода выше критического значения может быть рекомендовано пересмотреть историю изменений отдельных файлов и вносить изменения в репозиторий включая большее количество измененных файлов, либо проводить более частый пересмотр актуальности имеющегося кода.

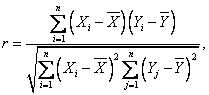
# Поиск взаимосвязей между волатильностью репозитория и качеством кода.

Для каждой из характеристик качества кода репозитория заданы следующие диапазоны оценки:

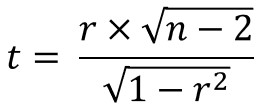
* Правила написания кода: -20 до +20, где -20 значительно лучше среднего и +20 значительно хуже среднего
* Дубликаты: от -10 до +10, аналогично -10 отсутствие дубликатов и +10 большое количество дубликатов
* Комментарии: от -5 до +5, где -5 отвечает наиболее адекватному количеству и качеству комментариев в исходном коде, +5 – количество комментариев избыточно, либо комментарии наоборот отсутствуют, из комментариев тяжело понять смысл комментируемого программного блока кода.
* Покрытие тестами: от -15 до +15, где -15 отвечает максимально возможному покрытию тестами функционала, +15 – минимальное покрытие тестами.

Общее качество программного кода репозитория рассчитывается как арифметическая сумма приведенных четырех показателей.

Корреляция [1] показателя волатильности с обобщенным показателем качества кода репозитория оказывается выше **88%**. Расчет производится следующим образом: обозначим хi – показатели характеристики волатильности (μ) репозитория, yi – соответствующие показатели качества программного кода, полученные суммированием соответствующих четырех показателей. Тогда искомый коэффициент корреляции:

, где

T – статистика для n = 1025, r = 88.78% вычисляется по формуле:

и равна 61.70.

Наконец чтобы убедиться в значимости найденной корреляции и отклонить гипотезу об ее отсутствии вычисляем p value по формуле: TDIST(x, deg\_freedom, tails), где х – найденная Т – статистика, степени свободы (deg\_freedom) = 1025 - 2 = 1023, для одностороннего анализа tails = 1 и является бесконечно малым числом. Таким образом, гипотеза об отсутствии корреляции отклоняется с уровнем уверенности более 99.99%.

# Заключение

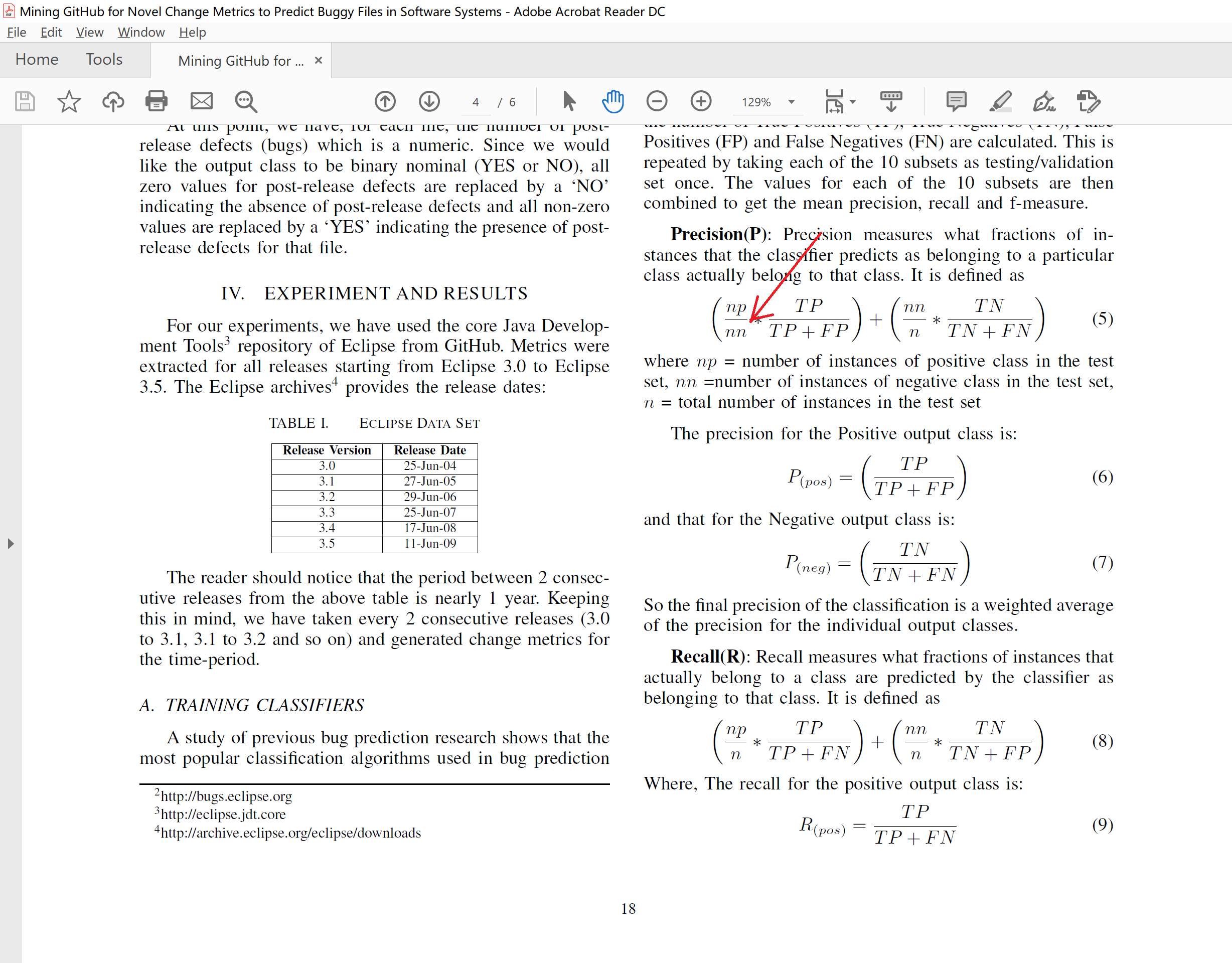
В статье изложены результаты анализа качества исходного программного кода имеющихся публичных репозиториев. Представлена статистика показателей волатильности. Подробно рассмотрен процесс интеграции методики с имеющимися в открытом доступе и наиболее часто используемых как крупными аутсорсинговыми компаниями, так и независимыми фрилансерами системами управления версиями и сборки программного кода. Результаты применения данной модели показали, что зная показатели волатильности можно объективно оценвать качество работы над исходным кодом разработчиками. Показана эффективность данной методики в сравнении с другими имеющимися на рынке и рекомендации по применению. Результаты исследования показывают, что предложенная методика может быть применена в качестве объективной оценки качества исходного программного кода независмо от языка программирования.

# Литература

1. ТЕОРИЯ СТАТИСТИКИ. УЧЕБНИК. Под ред. Громыко Г.Л. М.: ИНФРА-М, 2010
2. Git Pocket Guide: A Working Introduction, Richard E. Silverman, O'Reilly Media; 1st edition 2013
3. Pro Git, Scott Chacon, Apress; 1st edition, 2009
4. Mining GitHub for novel change metrics to predict buggy files in software systems, Muthukumaran K and Choudhary Abhinav and Murthy NL Bhanu, 2015 International Conference on Computational Intelligence and Networks, pages 15-20
5. Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities, Shin, Yonghee and Meneely, Andrew and Williams, Laurie and Osborne, Jason A, IEEE transactions on software engineering, 2010, p.772-787
6. Software Metrics, Fenton, NE and Pfleeger, SL, Boston, MA: International Thompson Publishing, 1997
7. Interactive churn metrics: socio-technical variants of code churn, Meneely, Andrew and Williams, Oluyinka, ACM SIGSOFT Software Engineering Notes, 2012, p.1-6
8. Code Churn: A measure for estimating the impact of code change, Munson, John C and Elbaum, Sebastian G, 1998
9. A practical model for measuring maintainability, Heitlager, Ilja and Kuipers, Tobias and Visser, Joost, 2007
10. Finding refactorings via change metrics, Demeyer Serge, Stéphane Ducasse, Oscar Nierstrasz, ACM SIGPLAN Notices, volume 35 number 10, pages 166-177, 2000
11. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction, Moser, Raimund and Pedrycz, Witold and Succi, Giancarlo, Proceedings of the 30th international conference on Software engineering, 2008
12. Tie strength metrics to rank pairs of developers from github, Natércia A. Batista and Sousa, Guilherme A and Brandão, Michele A and da Silva, Ana Paula C and Moro, Mirella Moura, Journal of Information and Data Management, 2018
13. A Metrics Suite for Object Oriented Design, Shyam R. Chidamber and Chris F. Kemerer, Transactions on software engineering, vol. 20, no. 6, June 1994
14. Towards a metrics suite for object oriented design, Shyam R. Chidamber, Chris F. Kemerer, Conference proceedings on Object-oriented programming systems, languages, and applications, November 1991 pages 197–211
15. An automated tool for collection of code attributes for cross project defect prediction Ruchika Malhotra; Bhavyaa Bansal; Chitranshi Jain; Ekta Punia, 2017 2nd International Conference on Man and Machine Interfacing (MAMI) 21-23 Dec. 2017
16. Are Change Metrics Good Predictors for an Evolving Software Product Line? Sandeep Krishnan, Chris Strasburg, Robyn R. Lutz, Katerina Goševa Popstojanova. Proceedings of the 7th International Conference on Predictive Models in Software Engineering, September 2011

Поскриптум.

Качество «научных» статей по-прежнему оставляет желать лучшего:



Примеры репо:

1. Python. Jittor/jittor: отличный формат кода, отсутствие дубликатов, наличие комментариев (<https://github.com/Jittor/jittor/blob/master/python/jittor/optim.py>), полное покрытие кода тестами (<https://github.com/Jittor/jittor/tree/master/python/jittor/test>)
2. Пример низкого качества кода на Python, JohnSnowLabs/spark-nlp: flake8 выдает ошибки, есть дубликаты, неплохие комментарии, есть неполное покрытие тестами (<https://github.com/JohnSnowLabs/spark-nlp/tree/master/python/test>)
3. Отличный код Javascript: Kyusung4698/PoE-Overlay (<https://github.com/Kyusung4698/PoE-Overlay/tree/5ced41b3b9cd8fed250ca8253aea3f950770f5cc/src/app/modules/evaluate/component/evaluate-search-chart>)
4. Менее качественный код Javascript: MoePlayer/DPlayer
5. Хорошее качество кода на Go Lang: OpenDiablo2/OpenDiablo2