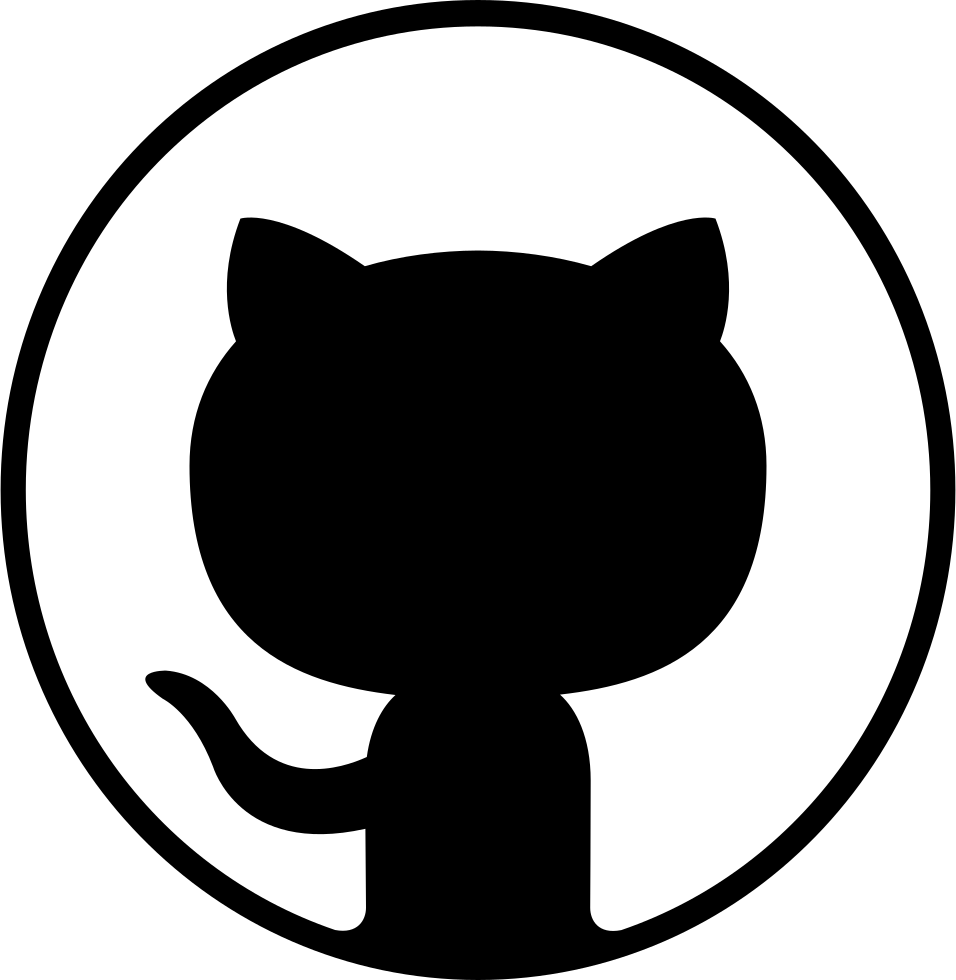
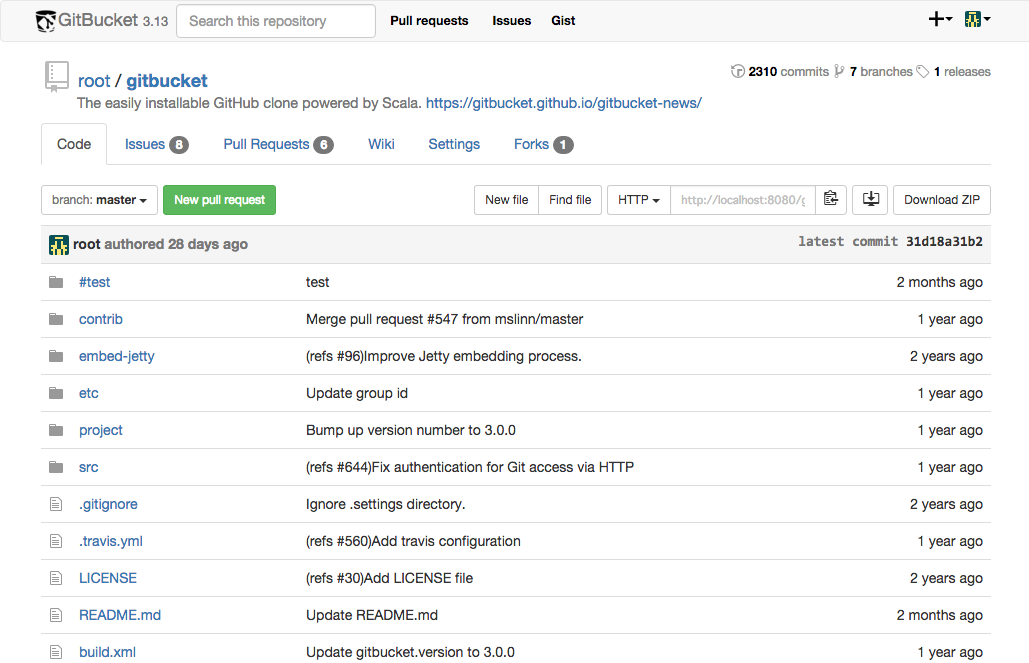
**Sessional :2**

**Git-Hub**

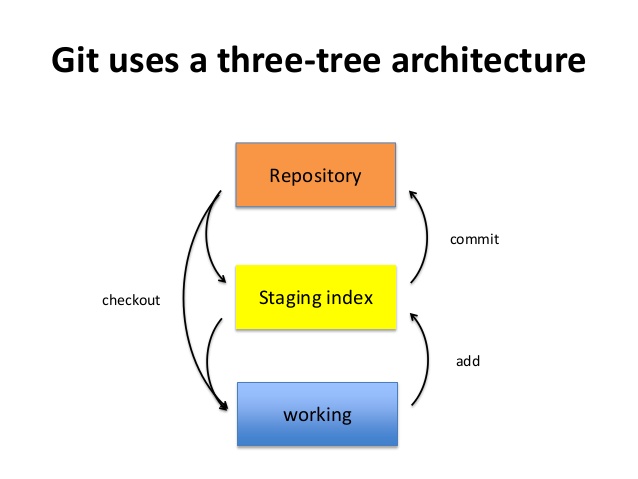
* **Logo:**

****

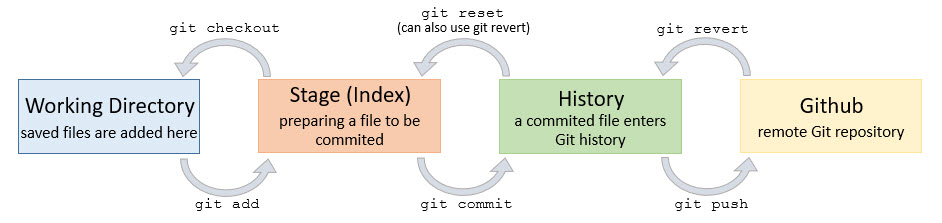
* **Git Hub interface**

****

**Git architecture:**

****

**Git Commands:**

****

**Features:**

* https://lh3.googleusercontent.com/l7A22iQfAWoDWP1Jyewd7HiiQyfAkVSIbbzJqHOx8hVl7s8TzrUPlqlxy1iM8BmbQjj0eQ=s50**Collaborative Coding**
* Contribute to projects quickly with automatic environment setup.
* Make sure you see the changes you care about.
* Build community around your code.
* https://lh3.googleusercontent.com/5iKcvbomiyC6UgavZ4dtWJiBljmhjhgUXmO1u-7D61rc-946plVH7H50StRvKGDU-2rtiQ=s50**Automation and CI/CD**
* Automate everything: CI/CD, testing, planning, project management, issue labeling, approvals, on boarding, and more.
* Standardize and scale  
  best practices, security, and compliance across your organization.
* Get started quickly with thousands of actions from partners and the community.
* https://lh3.googleusercontent.com/kNuqj-pf2Mb16MGKx9wRi_o1iGPexDKK7A9Br3z-IKILl8HpMP9VM6oFoJB4YOzB9QKSiQ=s50**Security**
* Secure code as you write it. Automatically review every change to your codebase and identify vulnerabilities before they reach production.
* Understand and address any vulnerabilities in your open source dependencies.
* Automatically detect and deactivate secrets committed to your repos.
* https://lh3.googleusercontent.com/PIJBVwQnW9krq_9LsVzpHodPkLiRgfc738HvdAHPTTd9tL8c_rP6vGOQ-XxSWfjyNPrDvg=s50**Client Apps**
* Take GitHub with you on any connected device.
* Skip the visual UI with a quick, powerful text interface.
* Access GitHub from your OS X or Windows desktop.

https://lh3.googleusercontent.com/qJOYsbgSmtKEpydiKPQWmknNEml0nJaDmzcznLmnl4xLz7Q40-TRA6PcZKSox7Zd5PrBmQ=s51**Project Management**

Keep your team in step and your code at the center of your projects.

Give your team continuously updated information on your progress, priorities and roadmap.

Track what you deliver down to the commit.

**Team Administration**

Simplify access and permissions management across your projects and teams.

Update permissions, add new users as you grow, and give everyone the exact permissions they need.

Sync with Okta and Azure Active Directory.

Title:Analysis of Software Developer Activity on

a Distributed Version Control System

Author: Shu Li\*, Hayato Tsukiji\*, and Kosuke Takano\*\*

Year:2016

* In this paper, for the analysis of various characteristics of software developers, such as their skills and project roles, a method of feature extraction based on those developers’ history of collaborative

development using a distributed version control system has been presented.

* Real Git projects on GitHub were experimentally analyzed, and the

experimental results demonstrate the feasibility of our proposed method**.**

* **Figure :Data available using Git-Hub API**

**Project Name**

**Overview**

**Name of Developer**

**Commit Message**

**URL**

**Name of project Owner**

**HTML 5**

**PHP**

**Programming language and amount in which they are used**

**JS**

**Node.js**

**C**

* Our proposed method allows us to analyze each developer’s characteristics in terms of “strong development areas (M/V/C)”, “contribution”, “initiative”, “support”, and “leadership” in progressing software development on a Git project.

* **Figure: Items that can be analyzed using proposed model**

**(E)Leadership**

**(D)Support**

**(C)Initiative**

**(B)Contribution**

**(A)String development area(M/V/C)**

**Analysis of Developer**

**Analysis of MVC Framework**

**Controller**

**View**

**Model**

* **Example of Analysis result for a Software Development project using Laravel**
* Based on these data, our method extracts developer characteristics with regard to

(A) development areas

(B)contribution

(C) initiative

(D) support

(E) leadership.

Model:

**Analyzer**

**Git Projects on Git-Hub**

Laravel provider

(1)Analysis of MVC Architecture

Ruby On Rail provider

Java Spring 4 provider

Xxx Provider

**Providers for MVC Framework analysis**

(4)Count of number of function calls in each MVC layer

(5)Count of number of configuration files/tools committed

**Functions of analysis of development status in MVC framework**

(3)Count of no. of source codes modified by other developers in each MVC layer

(2)Count of no. of source codes newly committed in each MVC layer

(D)Extraction of support

(E)Extraction of leadership

**Functions for extraction of Development features**

(C)Extraction of initiative

(B)Extraction of contribution

(A)Extraction of development area

Extraction of development areas]

**Filename** **Development area** **Function** **Role**

**Database access**

**Model**

**Database**

**engineer**

**xxx.model**

**xxx.html**

**xxx.control**

**UIOutput**

**Coder**

**Front end**

**engineer**

**Event Handler**

**Controller**

**View**

Conclusion:

By means of experiments using real Git projects on GitHub, method

Has been developed that allows us to extract the characteristics of software developers with regard to “strong development areas (M/V/C)”,“contribution”, “initiative”, “support”, and “leadership”

based on an MVC framework analysis of a Git project.

Paper:2

Title:No Single Metric Captures Productivity

Author: Ciera Jaspan, Caitlin Sadowski

Year:2019

* Earlier attempts have been made to measure productivity based on lines of code (LOC).
* **Why Do People Want to Measure Developer Productivity?**

1. one possible motivation for measuring developer productivity is identifying high/low-performing individuals and teams
2. other motivations include surfacing global trends across a company, rating the effectiveness of different tools or practices, running comparisons for an intervention meant to improve productivity, and highlighting inefficiencies where productivity can be improved.

* **What’s Inherently Wrong with a Single Productivity Metric?**

1. **Productivity Is Broad**

* When we create a metric, we are examining a thin slice of a developer’s overall time and output. Developers engage in a variety of other development tasks beyond just writing code, including providing guidance and reviewing code for other developers, designing systems and features, and managing releases and configuration of software systems
* Even for the narrow case of measuring productivity of developers in terms of code contributions, quantifying the size of such contributions misses critical aspects of code such as quality, or maintainability.

1. **Flattening/Combining Components of a Single Aspect Is Challenging**

* flattening all of these into a single measure along with quantity has limited applicability and risks, reducing the action ability of a metric. Is a developer with few code contributions of very high quality more or less productive than a developer with many contributions but some quality issues?
* flattened metrics may not make intuitive sense and so may be distrusted or misinterpreted. For example, if a variety of factors (e.g., cyclomatic complexity, time to complete, test coverage, size) are compressed into one number representing the productivity impact of a patch.

1. **Confounding Factors**

* Even if we are able to tease out a single metric that holistically covers some aspect of productivity, confounding factors can make the metric meaningless. Take the case of comparing programming languages. It is difficult to measure the productivity of languages in particular because of the number of confounding factors. There is the language itself, the tools, the libraries, the culture, the types of projects, and the types of developers who are attracted to that language.
* There can even be *externalities* that are not captured within a metric. For example, one team might appear to be submitting fewer lines of code than another team. There are many possible causes for such a difference that do not mean the team has lower productivity; perhaps the team is taking more steps to improve quality and therefore has fewer bugs down the road,

**Conclusion:**

* There is no single productivity metric for software engineers.
* Instead, focus on a set of custom metrics targeted to a specific question.

Paper:3

Title:Software Developer Performance Measurement Based on Code Smells in Distributed Version Control System

Authors: **Natach Jongprasit and Twittie Senivongse**

Year:

* This paper proposes a method and a supporting tool for measuring the performance of individual software developers in a Git project based on code smells.
* The number of bad smells in the code can be considered as one of the code quality aspects of the developer performance.

**Table 1** Implementation smells

|  |  |
| --- | --- |
| Implementation smell | Description |
| Complex conditional | A complex conditional statement |
| Complex method | A method with high cyclomatic complexity |
| Duplicate code | A code clone within a method |
| Empty catch block | A catch block of an exception is empty |
| Long identifier | An identifier with excessive length |
| Long method | A method is excessively long |
| Long parameter list | A method has long parameter list |
| Long statement | An excessively long statement |
| Magic number | An unexplained number is used in an  expression |
| Missing default | A switch statement does not contain a default  case |

* **Code Smells**

Code smells are symptoms of poor design and implementation choices in the code which may indicate deeper problems and affect program maintainability.

Code smells are not bugs as code still functions but they may lead to bugs and failure in the future. Here, the focus is on two categories of code smells:

(1) implementation smells which are code structures that indicate potential problems in the implementation,

(2) design smells which are code structures that violate fundamental design principles.

**Method to Measure Performance of Individual Software Developer**

****

* Figure depicts a typical collaboration scenario in a Git project. Developers may clone a project, branch off the master branch to modify the files, and make commits. Later the branch can be pulled to merge with the master when the modification is done. Measuring the performance of a developer considers only the commits made by adding or modifying source code.
* In Figure, all six code snapshots (i.e. initial code, commits C1-C4, and code at the pull request) are scanned for code smells.
* The initial codebase and the code snapshot created by the pull request are scanned, so that the detected code smells will be used further to calculate developer performance.

**Code Smell Density Calculation**

* Code smell density is calculated by

SDj = Number of smells in code j / KLOCj

where *j* = a version of code snapshot,

*SDj* = smell density of a code snapshot *j* ,

*Number of smells in code j* = number of code smells in code snapshot *j* ,

*KLOCj* = size of code snapshot *j* (in the unit of a thousand line of code).

**Performance Score Calculation Based on Previous Code**

**Snapshot**

* In a collaboration setting, all developers who previously have committed changes to the source code may have their share of the smells found in a particular code snapshot.
* Therefore, the performance of the developer who commits that code snapshot is based on whether he/she has improved the quality of the code, compared with the previous version of the code snapshot
* That is, the developer would be considered as having good performance if he/she can reduce the smell density by fixing some smells in the previous version of the code and not adding many new smells to the added and modified code in the version that he/she has committed.
* The performance score of a developer based on previous code snapshot is therefore calculated by

PPi j = SDi − SDj

Where

j = a version of code snapshot committed by this developer,

i = a previous version of code snapshot j ,

PPi j = performance score of this developer based on quality improvement made to the code snapshot j in comparison with the previous version i ,

SDj = smell density of a code snapshot j ,

SDi = smell density of a previous code snapshot i ..

Conclusion:

* proposes a method and a set of metrics, based on Bayesian average rating, to compute individual performance of developers in a software project in which the collaboration is via a distributed version control system like Git.