CSU33012-SOFTWARE ENGINEERING

MEASURING SOFTWARE ENGINEERING REPORT

Introduction

This is a report on measuring software engineering activity divided into 4 parts where I have outlined and examined the ways and metrics through which one can assess the activity by different methods, platforms and algorithms**.** To measure engineering activity, one can use software metrics and KPIs(Key Performance Index).

Part 1: How can one measure engineering activity

STRUCTURE

When feasible, use a deeper package hierarchy, fewer classes, and simple methods to organise the code: By reducing module dependencies, creating smaller code items, and breaking down large and complicated packages and procedures into structured Java packages with simpler classes and methods, code metrics were much improved. This approach also eliminated type conflict during code blending, but also the separating and separation of co-dependent code into independent components for synchronization with other modules with which it shared a higher proclivity. Splitting dependencies made re-allocating components across modules easier, which encouraged module structural uniformity.[1]

COMPLEXITY

Thomas J. McCabe, Sr. created Cyclomatic complexity which is a software metric that indicates the complexity of a program. It is a quantitative measure of the number of linearly independent paths through a program's source code. They are a set of file integrity metrics that deliver a quantitative basis for estimating code complexity based on a program's decision structure. The theory behind McCabe metrics is that the more structurally complex a code becomes, the more complicated it is to test and sustain it, and thus the likelihood of defects increases. Lines of code metrics are straightforward measurements that can be extracted from the code. Total lines of code, blank lines of code, lines of commented code, lines of code and statement, and lines of executable code are examples of metrics.

[2]

McCabe metrics	
Cyclomatic complexity ($v(G)$)	Number of linearly independent paths
Cyclomatic density (vd(G))	The ratio of the file's cyclomatic complexity to its length
Decision density (dd(G))	Condition/decision
Essential complexity (ev(G))	The degree to which a file contains unstructured constructs
Essential density (ev(G))	(ev(G) − 1)/($v(G)$ − 1)
Maintenance severity	ev(G)/$v(G)$
Lines of code metrics	
Total lines of code	Total number of lines in source code
Blank lines of code	Total number of blank lines in source code
Lines of commented code	Total number of lines consisting of code comments
Lines of code and comment	Total number of source code lines that include both executable statements and comments
Lines of executable code	Total number of the actual code statements that are executable
Halstead metrics	
n1	Unique operands count
n2	Unique operators count
N1	Total operands count
N2	Total operators count
Level (L)	(2/n1)/(n2/N2)
Difficulty (D)	1/L
Length (N)	N1 + N2
Volume (V)	$N \times \log(n)$
Programming effort (E)	DV
Programming time (T)	E/18

Halstead metrics are used to directly assess the difficulty of a program module from its source code, with an emphasis on computational load. These metrics were created to determine module complexity solely from the operators and operands. The Halstead metrics are based on the premise that the more difficult the code is to understand, the more genetic flaw the modules have.[3]

Calculation [edit]

For a given problem, let:

- η_1 = the number of distinct operators
- η_2 = the number of distinct operands
- N_1 = the total number of operators
- N_2 = the total number of operands

From these numbers, several measures can be calculated:

- Program vocabulary: $\eta = \eta_1 + \eta_2$
- Program length: $N = N_1 + N_2$
- Calculated estimated program length: $\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$
- Volume: $V = N \times \log_2 \eta$
- Difficulty : $D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2}$
- Effort: $E = D \times V$

The difficulty measure is related to the difficulty of the program to write or understand, e.g. when doing code review.

The effort measure translates into actual coding time using the following relation,

- Time required to program: $T = \frac{E}{18}$ seconds

Halstead's delivered bugs (B) is an estimate for the number of errors in the implementation.

- Number of delivered bugs : $B = \frac{E^{\frac{2}{3}}}{3000}$ or, more recently, $B = \frac{V}{3000}$ is accepted[citation needed].

Frequency of commits:

Code churn (also interchangeably known as rework) is a metric that indicates how often a given piece of code—e.g., a file, a class, a function—gets edited. As you'll soon see, if a given piece of code receives changes too often, that's usually a bad sign.[4]

Code churn is a measure of the amount of code change taking place within a software unit over time. Churn often propagates across dependencies. Commit frequency is typically used to reward teams with a high frequency and improve teams with A lower one. At face value, commit frequency may seem like an okay metric. Of course, it's not perfect but it might be useful and easy to count. You can encourage small, frequent commits and - if used right - could support greater transparency, collaboration and continuous delivery. You can also identify a team with lower number of commits - which could be a problem - and encourage smaller commits which make sense too. The problem with the number of commits is that doesn't tell you the actual value delivered. Just like lines of code, it's very easy to game. Just create more commits. [5]

Mean Time To Repair (MTTR)

The mean time to repair (MTTR) is a fundamental metric for determining the maintainability of remanufactured objects. It denotes the average amount of time necessary to repair a faulty part or equipment. [1] It is the overall corrective maintenance time for failures divided by the total number of compensatory maintenance activities for failures performed within a certain period. [2] It often does not include lead time for items that are not promptly available, as well as any administrative or logistical downtime (ALDT).MTTR is commonly used in fault-tolerant design to incorporate the time the issue remains latent (the time from when the failure occurs until it is detected). The system may be unable to recover if a latent defect is unnoticed until an impartial failure occurs.

Goodhart's Law and the Effect of Measuring

The lines of code metric failed to be a good signal as soon as it was used to gauge a developer's productivity and influenced the developer's wage, with developers manipulating the strategy to increase salary.

[6]

Lead Time

A lead time is the amount of time that elapses between the start and finish of a process. For example, the duration between placing an order and receiving new automobiles from a certain automaker might range from a fortnight and 6 months, depending on a range of parameters. The entire time order to manufacture an item, according to one business lexicon, includes order preparation time, queue time, setup time, run time, move time, inspection time, and put-away time. In the case of made-to-order items, the time between the release of an order and the manufacture and shipment that satisfy that order is referred to as the order fulfilment period.

[7]

PART 2: Platforms on which one can gather and perform calculations over these data sets

Basecamp

Basecamp is a real interaction platform that may be used by development teams for resource planning and long-term scheduling. Basecamp includes a calendar, a to-do list, and a file-sharing system that teams may use to keep track of priorities. Basecamp is not industry-specific; rather, it may be used by any organization that seeks to manage a workgroup, particularly non-profits, start-ups, and freelancers.

GitHub and Git

Git is a distributed version control system that is open source. GitHub hosts this version control system in addition to offering additional services such as bug tracking, project status, task management, and continuous integration. With over 40 million users, it is amongst the most renowned git hosting services. Its users provide a plethora of information that may be accessed using the GitHub API. It allows programs to collect public data about individuals, teams, and projects in addition to creating their git releases and other tasks via the API.: [8]https://api.github.com

This includes:

Project Commits - Data about a project's commits can be fetched.

- GET /repos/:owner/:repo/git/commits/:commit_sha

· **User Commits/Pull Requests** – Data about a user's events can be fetched.

- GET /users/:username/events

· **Release Data** – Data about a project's releases which could be used to work out the cycle time.

- GET /repos/:owner/:repo/releases

· **Project Cards** – Data about a project's tasks including the task's state.

- GET /projects/columns/:column_id/cards

Some of this information may also be examined graphically using the project insights tab on GitHub. This displays a user's commit frequency as the number of commits each calendar day. This insights tab indicates the number of project contributors, traffic, forks, branches. This data, which covers the plurality of the indicators listed above, may be leveraged to get insights into the work of an individual developer or team. As previously said, GitHub gives its insights into the majority of the data; nevertheless, data collected via the API may be analyzed using other tools and technologies.

JIRA

Atlassian's JIRA is a proprietary issue tracking solution that offers bug tracking and agile project management. Users may monitor releases, processes, and issues using JIRA's product development component. Because the JIRA API is incorporated inside your product, the endpoints are hosted on your own private JIRA server. The API allows for the collecting of agile metrics data.

- **View tasks and issues assigned to an individual** – Can view all the issues assigned and completed by an individual, useful for finding a developers velocity.
  - GET /rest/api/2/search/?jql=assignee=:name
- **View tasks finished in a timespan** – Can view all tasks finished in a period which can be used to work out team velocity.
  - GET / rest/api/2/issue/search?jql=duedate=:”xxxx-xx-xx”

This data, like GitHub's API, may be utilized to calculate some of the agile metrics described above. This software, on the other hand, is not open source and requires money and effort to incorporate.

Personal Software Process (PSP)

Engineers can use the Personal Software Process (PSP) to create a disciplined personal framework for completing software development. It includes a toolset of methodologies, forms, and scripts that enable and showcase how developers may prepare, manage, and monitor their work. Watts Humphrey created the PSP while working at Carnegie Mellon University. It is meant for developers who want to optimize their particular software development process.

PSP is split into four levels:

1. **PSP 0 –** Personal measurement, basic size measures and coding standards.
2. **PSP 1 –** Planning of time and schedule.
3. **PSP 2 –** Personal quality management, design and code reviews.
4. **PSP 3 –** Personal process evolution.

As forms and scripts are an integral part of this framework, there are a variety of tools available. **The Software Process Dashboard** is an open-source initiative to create a PSP support tool. Originally developed by the United States Air Force, it has now evolved under the open-source model. The dashboard supports:

- **Data Collection –** Time, defects, size and plan vs actual data.
- **Planning –** Integrated scripts, templates, forms, summaries and earned value.
- **Tracking –** Earned value support.
- **Data Analysis –** Charts and reports aiding the analysis of historical data trends.
- **Data Export –** To allow the use of other tools.

The open-source model allows the dashboard to be modified and distributed at no cost.

The data collected and analysed by the PSP and supporting tools allows developers to evaluate their work and can suggest improvements.

Algorithms

Algorithms are major elements of both data generation and analysis. While algorithms are required to evaluate patterns in data, they may also be used to improve the quality of data obtained by selecting just the most significant indicators to report. These algorithms can be used to analyse current high-quality software projects can reveal which metrics we should truly be tracking.

Genetic Algorithms

A genetic algorithm is a strategy for optimising a given scenario that uses a natural selection process similar to biological evolution. This technique could be used with many of the metrics discussed in the preceding section to determine the most significant metrics seen in high-quality software projects. One study found that after 20 generations of using metrics such as SLOC, number of comments, and number of methods, among others, the main metrics correlating to a quality software project were: number of comments and comment length white space and method name length; and apparent complexity of methods.

Affinity Propagation

A clustering algorithm is Affinity propagation. The algorithm communicates between data points by sending 'messages.' Each data point sends out its relative attractiveness to each other data point, allowing the receiver to express its ability to associate with the transmitter, leading to the formation of a cluster of points. This may be applied to the software metrics covered so far to identify some of the most significant metrics seen in high-quality and successful software projects. This method may be used with a genetic algorithm to provide another technique for analysing software metrics and forecasting software quality.

PART 3: Various kinds of computation that could be done over software engineering data, to profile the performance of software engineers

Functional Point Analysis

Analysis of functional points Functional point analysis is the technique for measuring software requirements based on the many functions into which the demand might be divided. Allan J. Albrecht created it in 1979 and defined it as "a dimensionless number specified in function points that we have discovered to be a useful relative measure of function value offered to our clients." The FPA is a way of measuring functional size. It evaluates the functionality provided to its users based on the external perspective of the functional needs provided by the user. It is an instrument for evaluating the units of a software product to facilitate quality and productivity analysis. It can be used as a normalisation measure when comparing software.

Machine learning

Semmle.Inc. measured software development productivity using a machine learning method. Semmle examined over 10 million contributions made by 300,000 people in over 56, 000 open source projects. The programming language QL was used to assess the code's quality. Their effort was to divide the amount of code generated by the length of time required to develop the code. For every minute a given developer spent between two separate commits, they taught the neural network to estimate the chance that the person was coding during that period. They aim to quantify the amount of code created not by the sheer volume of code committed, but by the amount of 'labour time' required to develop the code. To accomplish this, researchers train a computational model to estimate the chance that a given developer was writing during the minute between two different code changes (a 'commit interval'). They train the neural network using recent data from the previously specified dataset for that developer. By combining these models, they may provide an in-depth quantitative analysis of the software development process.

PART 4:Ethics and legal or moral issues surrounding the processing of this kind of personal data

Instead of developers occupying a conference room for a day completing a code review while no new work on product feature is being generated and management fretting about a timetable even though code reviews are integrated into the plan, the sessions are brief. The above-average developer immediately zeroes in on the pattern in use and can analyze the code fast, providing valuable comments to a review. After using this style for a time, it becomes able to gather a few developers, project the code on the wall, and receive positive comments since the general design of the shop has become uniform and the size of any block of code has been reduced to an amount that can be digested rapidly. Because metrics and automated code scanning have eliminated minor difficulties, pair programming is achievable on a different scale than a controller and viewer method. Instead of identifying nits like a misplaced semicolon, ideas like multiple cores and code styles are being debated.[9]

Data Security Data privacy is another ethical dilemma associated with software engineering measurement. The sorts of data gathered and analyzed by teams may, in certain situations, violate a developer's privacy. When this pertains to this procedure, administrators must be absolutely upfront and explain to developers what measures are being deployed and what data is being monitored. GDPR and data privacy standards As previously noted, monitoring the success of software development teams using metrics necessitates a large amount of data. The sorts of data collected and processed by teams may, in some situations, violate the creator's confidentiality.

MY RECOMMENDATIONS

To determine the composition of the information which will be gathered to create a flaw forecasting model, I suggest a simple road map: - Improve defect prediction by leveraging churn and/or social network indicators. Churn data may be extracted automatically from version management system logs. Mining issue management systems, version management systems, and developers' e-mails, which they send to one other to discuss software issues and new features, may also be used to collect social interaction metrics automatically. After extracting static code, churn, and social communication metrics, fault diagnosis models can be built using any combination of different types of indicators (i.e., static code metrics; churn metrics; social interaction metrics; static code and churn metrics; static code and social interaction metrics; churn and social interaction metrics; and static code, churn, and social interaction metrics). The performance comparison findings among these simulations may be used to determine which models will be deployed to detect defect-prone regions of the system.

REFERENCES-

https://git-scm.com/

https://developer.github.com/v3/

https://www.atlassian.com/software/jira

https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=5283

https://www.processdash.com/

https://ieeexplore.ieee.org/document/1226139

https://sss.bnu.edu.cn/~pguo/pdf/2008/y.pdf

[1]https://www.sciencedirect.com/topics/computer-science/code-metrics

[2]https://www.sciencedirect.com/science/article/pii/B9780124115194000161

[3]https://www.sciencedirect.com/science/article/pii/B9780124115194000161

[4]https://linearb.io/blog/what-is-code-churn/

[5]https://www.usehaystack.io/blog/software-development-metrics-top-5-commonly-misused-metrics

[6]https://www.sciencedirect.com/science/article/pii/B9780128042069000131

[7]https://en.wikipedia.org/wiki/Lead_time

[8]https://api.github.com

[9]https://www.sciencedirect.com/science/article/pii/B9780123815200000084