Measuring Software Engineering

Introduction

Software engineering is the systematic process of applying engineering approaches when developing software. This process includes designing, maintaining and testing software. The software engineer must implement all the requirements of the client and work in teams with other developers.

Measuring software engineering is evaluating the size, quality and a particular attribute of the software. It is used to plan for the future of a project, improve the quality and budget for the project. The work of developers can also be measured. Metrics are measurements of this and there are many different types of metrics. Computation and algorithms are used on the collected data from the metrics. Many platforms and different infrastructures have emerged to do this collection and analysis.

The collection of such data raises many ethical questions. The data used to track the progress of a project or an engineer has many benefits and it is related to the work rather than the individual. Tracking the person breaches their privacy.

Metrics

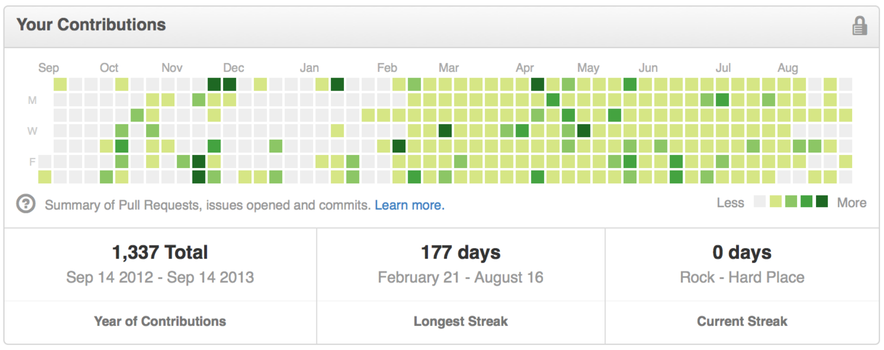
Lines of Code

Counting the number of lines of code (LOC) written by a software engineer has long been used as a method to measure software engineering. It is used to measure the productivity of the programmer and the quality of code, by counting the defects per thousand lines of code.

Although this approach is simple to implement and automate, it has major drawbacks. This measure does not take into account many important aspects of programmes like effort, functionality and complexity. It also is not comparable over different languages, for example the LOC of an assembly language program does not correspond to the LOC of a high-level language program like Python. This method does not encourage better and more efficient programming as the engineer may add extra unnecessary lines of code to improve their score.

Number of commits

The number of commits a programmer makes can also be used to measure the amount of work they do. It could be assumed that better developers make more commits. However the number of commits does not correlate with the amount of progress made in the program. Commits can consist of code deletion, adding comments, merging branches and many more operations and therefore cannot be compared between different programmers. Developers may be tempted to make unnecessary commits to increase their score.

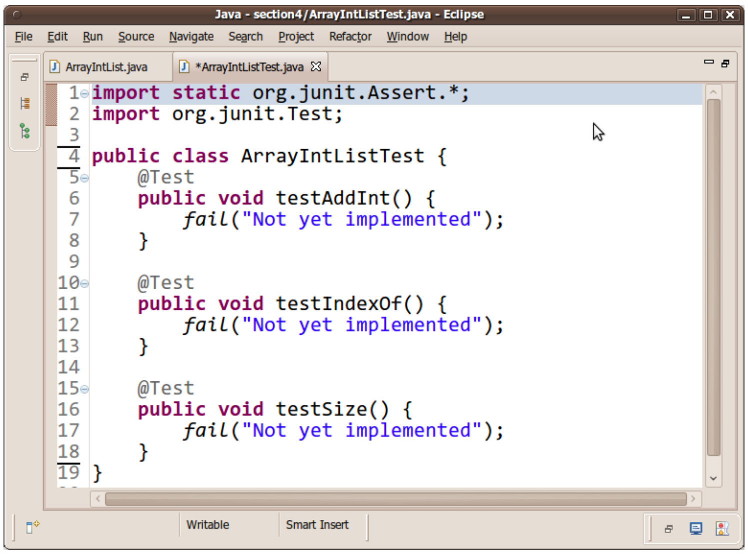


Summary of Pull Requests, issues opened and commits. Learn more.

1,337 Total	177 days	0 days
Sep 14 2012 - Sep 14 2013	February 21 - August 16	Rock - Hard Place
Year of Contributions	Longest Streak	Current Streak

Code Coverage

The number of lines of code executed by test cases created for a program can also be used as a metric. Very high code coverage decreases the likelihood of having undetected bugs in the code. This metric can be used to track how rigorous a programmer is when writing code and if they account for all possibilities. A drawback of this metric is the time required to write thorough test cases for all the code.


Java - section4/ArrayIntListTest.java - Eclipse
File Edit Run Source Navigate Search Project Refactor Window Help
ArrayIntList.java
*ArrayIntListTest.java
Writable
Smart Insert

```java
import static org.junit.Assert.*;
import org.junit.Test;

public class ArrayIntListTest {
    @Test
    public void testAddInt() {
        fail("Not yet implemented");
    }

    @Test
    public void testIndexOf() {
        fail("Not yet implemented");
    }

    @Test
    public void testSize() {
        fail("Not yet implemented");
    }
}
```

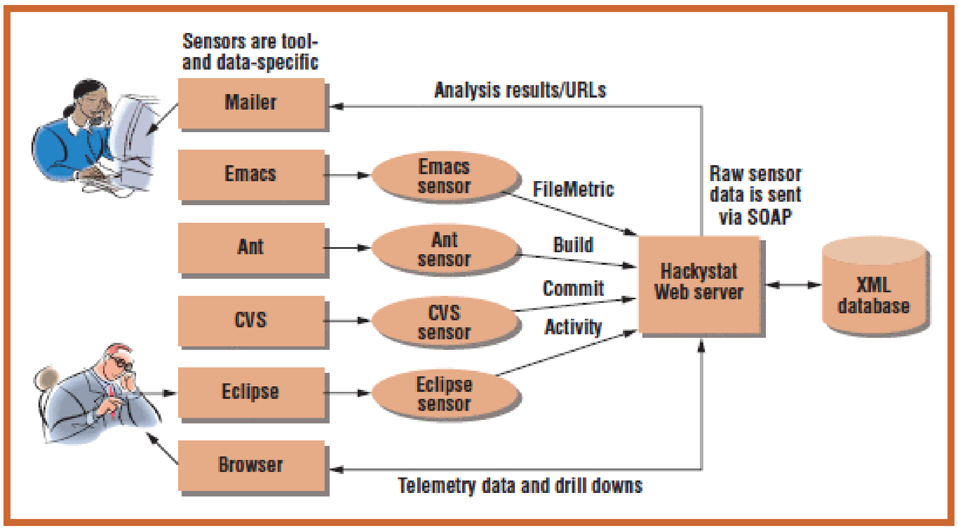
Infrastructure

Hackystat

Hackystat is a framework in which metrics are collected automatically by sensors installed in development tools, and sent to servers which analyse the collected data. It eliminates the need for manual input of information by the user. The sensors unobtrusively track the users activities and send data to the web server. They collect activity data such as the file being modified, the size and defect data, unit tests failing or passing. The server analyses the data over regular intervals. It stores the data in XML-based repositories.

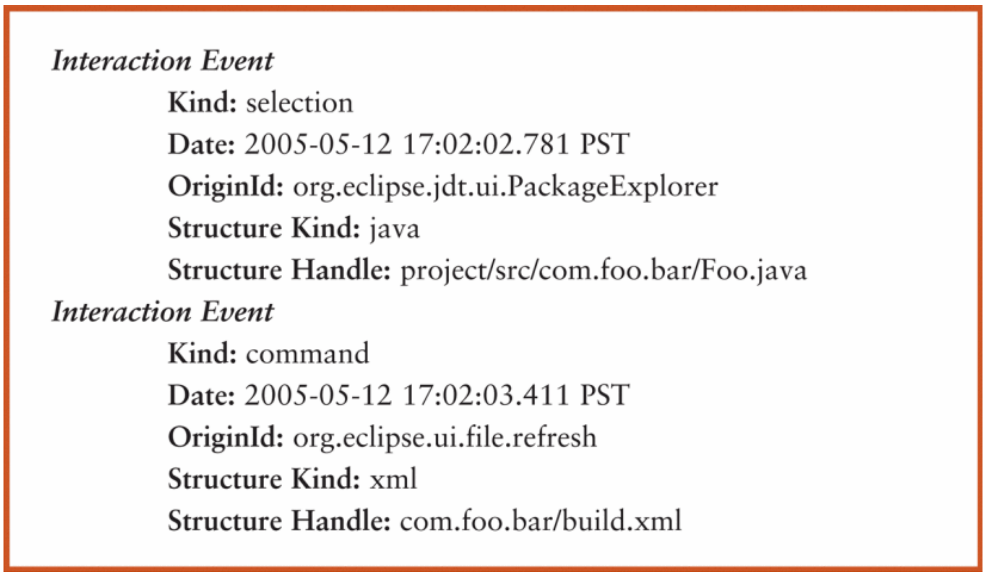
Hackystat:

- Tracks the time the user spends working on a specific file and the changes they make.
- It collects build data, the results of the code compiling, linking and testing the system.
- Tracks execution by observing the system as the code is executed.
- It collects usage data by looking at the user's interaction with the system, such as frequency, types and sequences of commands invoked.



<u>Mylar</u>

The Mylar Monitor is a standalone framework that gathers and reports on trace information about a user's activity in the popular Java IDE, Eclipse. It records events such as preference changes, window events, selections, periods of inactivity, and commands invoked These events are stored in a local XML file and uploaded by the developer to an HTTP server. Plug-ins can extend the Monitor for different kinds of user studies. The volume of data collected can be quite large so the Mylar Monitor compresses the data to reduce file size.


Interaction Event
- **Kind:** selection
- **Date:** 2005-05-12 17:02:02.781 PST
- **OriginId:** org.eclipse.jdt.ui.PackageExplorer
- **Structure Kind:** java
- **Structure Handle:** project/src/com.foo.bar/Foo.java

Interaction Event
- **Kind:** command
- **Date:** 2005-05-12 17:02:03.411 PST
- **OriginId:** org.eclipse.ui.file.refresh
- **Structure Kind:** xml
- **Structure Handle:** com.foo.bar/build.xml

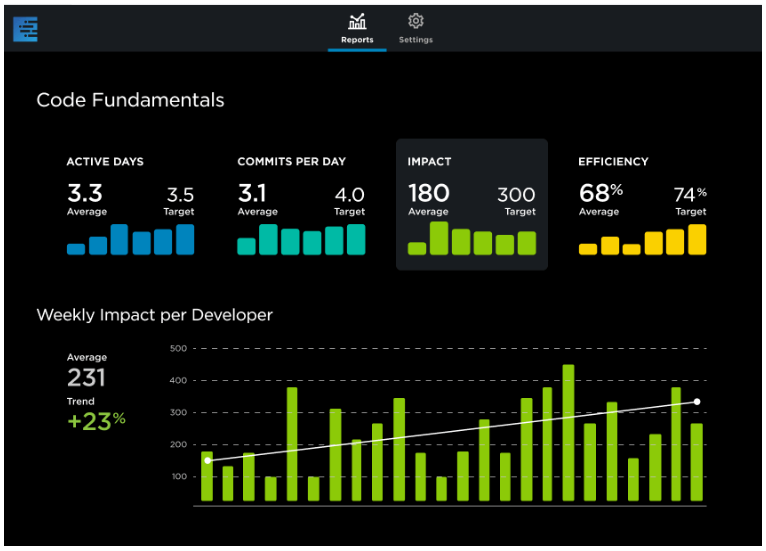
<u>Google Cloud</u>

Google cloud datalab is an interactive tool which allows the user to explore, visualise, analyse, transform data and build machine learning models in an easy way. Data is stored in notebooks which include code, documentation and the results of code execution. The notebooks work like text editors or IDEs, allowing the user to write and run code in them. The notebooks can be stored in git repositories with Google Cloud.

Cloud Datalab contains some of the most commonly used Python libraries used for data analysis, visualization, and machine learning. It has libraries for using important Google Cloud Platform services, such as BigQuery, Machine Learning Engine, Dataflow, and Cloud Storage.

Pluralsight Flow

Flow is an engineering analytics platform created by Pluralsight. It tracks data from commits, pull requests and tickets and creates reports and graphs with the data. It returns reports about the commit efficiency per month for each language, the skill level and the development of the skill level through the commits. ‘Work Log’ provides a single view of the team of developers' contributions. It shows the projects commits and pull requests, it gives an insight into what the average ‘day in the life’ of the developers in the team, and it identifies bottlenecks easily. This data allows the developer to improve their development method and progress faster.



Computational Analysis

Halstead's Metrics

Halstead's Metrics is based on counting the number of lines of code. It counts tokens, which are either operators or operands, and collects the following values :
n1 = number of unique operators
n2 = number of unique operands
N1 = number of occurrences of all operators
N2 = number of occurrences of all operands

Halstead's metrics include:

- Program Length: N = N1+ N2.
- Vocabulary: The total number of distinct operator and operands appearances, n = n1 + n2.
- Program Volume: Proportional to the size of the program, gives the size in bits of space required to store the program. It depends on the algorithm being implemented in the program. V = Size * (log2 vocabulary) = N * log2 (n).
- Program Level:The value of L ranges between 0 and 1. It represents the level of abstraction of the programming language used. The higher the level of the language, the less effort is required in programming with it. L = V * / V
- Program Difficulty: This shows how difficult the program is to handle. D = (n1 / 2) * (N2 / n2), D = 1 / L. As the volume of the code increases, the level of the program decreases and the difficulty increases. This means

that repetitive use of operands and the lack of use of higher level constructs results in an increase to volume and difficulty.

- Programming Effort: The amount of mental effort required by the programmer to implement an algorithm in a specific language. E = V / L = D * V = Difficulty * Volume.

There are a number of rules for counting in each language. For example in C, all the variables and constants are considered operands and function calls are considered as operators.

Halstead metrics are useful measurements because they are easy to calculate and they quantify the overall quality of the code. It can be used for any programming language and can be used in reporting and scheduling projects. The drawbacks of Halstead metrics are that it requires finished code and cannot be used as a predictive estimating model.

Cyclomatic Complexity

Cyclomatic complexity is a metric used to specify the complexity of the program. It is computed using a directed graph, the nodes in the graph represent the smallest group of commands in the program and the directed edges connect two nodes. The path between two nodes is the order the commands are executed. For example, if the code does not contain any flow control statements, there is only one path through the program. The cyclomatic complexity of this program would be 1. If the program contains an if statement, there would be two paths through the program, true or false, and the cyclomatic complexity would be 2.

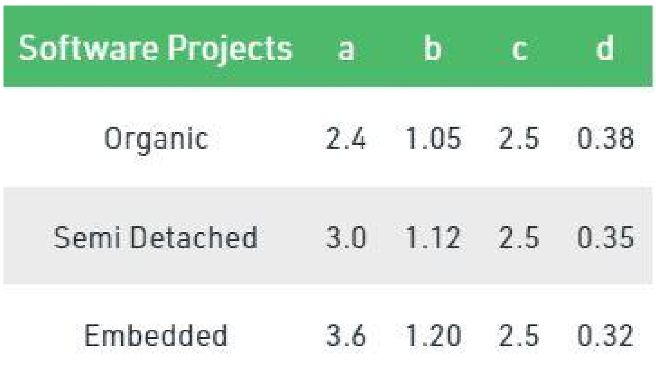
The formula for cyclomatic complexity is $M = E - N + 2P$, where E is the number of edges in the graph, N is the number of nodes and P is the number of connected elements.

This is a useful metric as it can be used as a quality metric as it shows the relative complexity of the program. It can be computed quicker than Halstead's metrics. It can be used to design test cases. Disadvantages of this metric are that nested conditionals can be difficult to understand and it may give misleading results when examining simple comparisons.

Constructive Cost Model

Cocomo (Constructive Cost Model) is a regression model constructed on LOC. It can be used to estimate the size, effort, cost, time and quality of a project. It is a procedural cost estimation technique. The outcomes of the Cocomo are effort, the amount of labour the project needs, and schedule, the time needed to finish the project. These are the key parameters for the quality of the program.

There are various Cocomo models which approximate a range of cost estimation at distinct levels depending on the level of accuracy. The basic Cocomo is E = a(KLOC)^b, time = c(Effort)^d and PersonRequired = Effort / time. The values for a, b and c are taken from a table depending on the category of the system being used.



Software Projects	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semi Detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

A project is of type organic if the team is small and experienced and the problem to solve is relatively easy. A semi-detached project lies in between organic and embedded in terms of team size and experience. The problems being solved in semi-detached projects are more difficult and require more ingenuity than organic projects. Embedded projects are the most complex and need the highest level of experience. A bigger team is required for this type of project.

The intermediate and detailed Cocomo models require many more parameters and produce more accurate results than the basic model. Overall the Cocomo model is easy to implement and understand. It is an easy way to estimate the total cost of the project and using drivers as parameters helps with understanding how different factors affect the project. Its drawbacks are that it does not take into account all the documentation and requirements of the project and it depends heavily on time factors. It also does not take into account the cooperation between the developers and the customer.

Ethics

There is much debate whether it is ethical to collect data about developers and to track their work. While there are some obvious benefits, there are also major concerns about this.

Analysing developers' work patterns can help improve the way they work and make them more efficient. Having a system tracking their progress is also a motivating factor and encourages the developers to adopt these improvements. Having a system monitor progress can also provide more recognition as developers achievements can be acknowledged, which again increases motivation. It also encourages developers to follow the proper engineering practises and their work is monitored.

However, this monitoring can go a step too far. For example, the active badge location system. This is a device that tracks an employee's location in the office. The badges transmit information about the location of the person to a network of sensors around the office. It is no longer just tracking the work of the employee but their movements. This is a major privacy concern and compromises the employees freedom of movement in their workplace.

Collecting data about the developers work is key to quickly improving how they work and highlighting areas that need improvement. Collecting data about them as a person, like their location or health, is an invasion of privacy and is not necessary for them to do their work.

Conclusion

In conclusion, there are many different ways to collect and analyse data from a developer or on a project. Different metrics can be used to measure the software engineering activity and there are a multitude of algorithms that can be used to analyse it. Different tools and platforms are continuously emerging that make tracking and quantifying the data simple. There are ethical concerns as this tracking becomes more invasive and less focused on the work itself.

Sources

Fenton, N. E., and Martin, N. (1999) "Software metrics: successes, failures and new directions." Journal of Systems and Software 47.2 pp. 149-157.

P.M. Johnson et al., “Beyond the Personal Software Process: Metrics Collection and Analysis for the Differently Disciplined,” Proc. 25th Int’l Conf. Software Eng. (ICSE 03), IEEE CS, 2003, pp. 641–646.

Johnson, Philip M., et al. "Improving software development management through software project telemetry." IEEE software 22.4 (2005): 76-85.

W. Snipes, V. Augustine, A. R. Nair and E. Murphy-Hill, "Towards recognizing and rewarding efficient developer work patterns," 2013 35th International Conference on Software Engineering (ICSE), San Francisco, CA, 2013, pp. 1277-1280.

A. Sillitti, A. Janes, G. Succi, and T. Vernazza, “Collecting, integrating and analyzing software metrics and personal software process data,” in Proceedings of the 29th Euromicro Conference. IEEE, 2003, pp. 336– 342.

G. C. Murphy, M. Kersten, and L. Findlater, “How are Java software developers using the Elipse IDE?” IEEE Software, vol. 23, no. 4, pp. 76–83, Jul. 2006.

W. Snipes, V. Augustine, A. R. Nair and E. Murphy-Hill, "Towards recognizing and rewarding efficient developer work patterns," 2013 35th International Conference on Software Engineering (ICSE), San Francisco, CA, 2013, pp. 1277-1280.

Pentland, A. (2014). Social physics: how good ideas spread-the lessons from a new science. Penguin.

R. Want, A. Hopper, a. Veronica Falc and J. Gibbons. The active badge location system. ACM Trans. Inf. Syst., 10(1):91–102, 1992.

L. Singer and K. Schneider, "It was a bit of a race: Gamification of version control," Games and Software Engineering (GAS), 2012 2nd International Workshop on, Zurich, 2012, pp. 5-8.

https://www.pluralsight.com/product/flow
https://www.geeksforgeeks.org/software-measurement-and-metrics/
https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/?ref=rp
https://en.wikipedia.org/wiki/Halstead_complexity_measures
https://www.geeksforgeeks.org/cyclomatic-complexity/
https://www.geeksforgeeks.org/software-engineering-cocomo-model/?ref=lbp