

# Measuring and assessing the software engineering process

By Nika Jashi

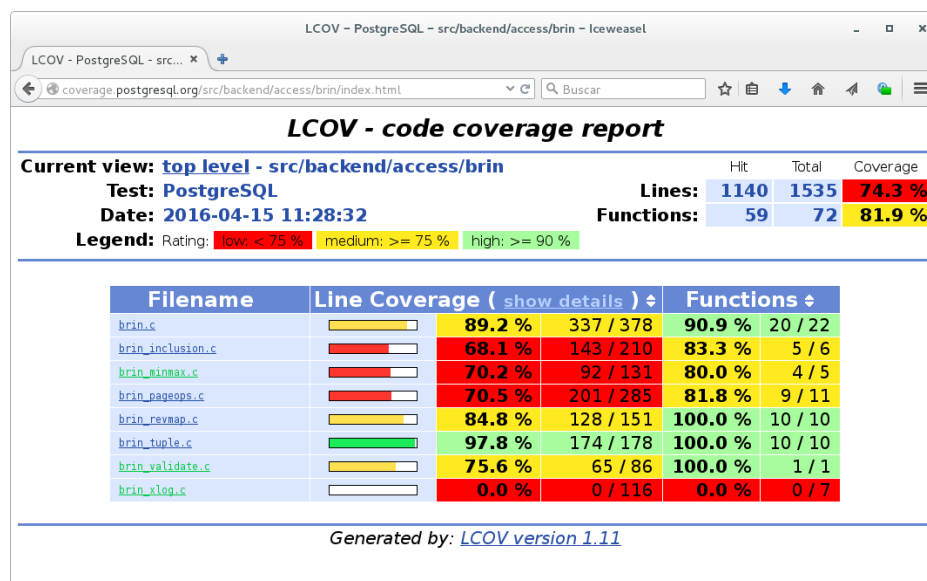
## Introduction

Measuring and assessing the software engineering process is a complex and nuanced task as there are number of methods and beliefs surrounding this topic. This report considers how the software engineering process can be assessed and examined in terms of measurable data. It delves into conventional methods as well as algorithmic methods for measurement and the platforms available in order to aid in this analysis. A number of ethical concerns are also explored during the process and recommendations are set out in order to mitigate these concerns.

## Conventional methods

**Lines of code (LOC):** perhaps the most primitive measurement of the software engineering process is counting the lines of code developed and written by an engineer. It is an easy metric to interpret as the lines of code can simply be noted and attributed to a specific engineer. However, it is not a very telling metric in terms of an impact the work of an individual has on an overall project. "Measuring software productivity by lines of code is like measuring progress on an airplane by how much it weighs."– Bill Gates. There exist far better methods for assessing the software engineering process.

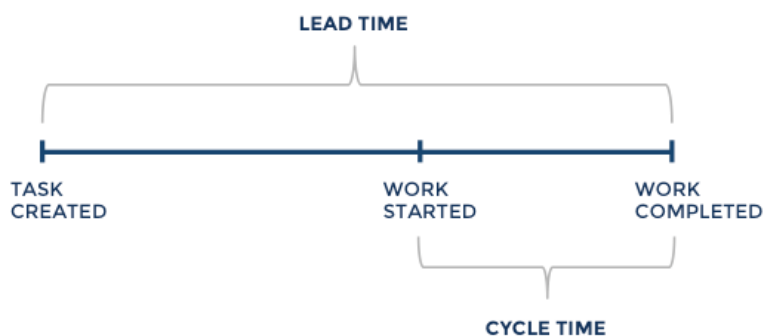
**Code Coverage:** a measurement of how many lines of code is successfully executed during testing. When compared with simply counting lines of code, it allows for a closer inspection of the productive capacity of an individual engineer. Code coverage plugins are available for most IDEs; it is a fairly efficient and easy to understand metric. The main criticism for code coverage relates back to simply measuring the lines of code, where the measurable data produced is an indicator of volume rather than the quality of the code written.



## SQL Code Coverage Report

**Team velocity:** suitable for larger projects, team velocity accounts for the quantity of software a team produces during a set time marker (a sprint), it can be measured in “story points” or hours; most commonly used in the agile development methodology. It is particularly useful for estimating the timeframe of a deliverable. Velocity is calculated by adding up the data from each story point at the end of a sprint. Worthy to note, however, that the metric cannot be used to compare different teams, as they can have differing interpretations of the story points, rather, it can help identify teams that have run into a development block, referred to as falling velocity.

**Cycle Time:** takes into account the total amount of time required for a specific task to be completed, for example, fixing a bug. This method can track both individuals and teams, much like team velocity, it is useful in estimating the speed at which a piece of software is developed. This metric is adopted from the lean manufacturing process, which places an emphasis on continual improvement through benchmarking.



*Source: Screenful*

**Metrics to measure software maintainability:** an important piece of measurable data is software maintainability. This refers to the degree to which an application is understood, repaired or enhanced. By tracking this data, organisations can identify areas for improvement and better understand the quality of the software they produce. Intertwined with the idea of software maintainability are the following metrics:

- Lead time: the length of time between the definition of a new feature and its deployment to the end user.
- Code coverage: as discussed above.
- Bug rates: considers the average number of bugs that arise from the deployment of a new feature. A key metric in determining the quality of the software produced.
- Mean time to repair (MTTR): the speed at which bugs, and issues are rectified and subsequently deployed again within a piece of software.

**Security:** an often-overlooked aspect in the discussion of evaluating the software engineering process is that of security. Put simply, this refers to the likeliness that an attacker can gain access to sensitive user information and or disrupt a software system in operation. Measurable data in this instance refers to the number of vulnerabilities (say per deployment), time to resolution of set vulnerabilities and the number of actual breaches in security that occur.

**User satisfaction:** involves gathering feedback from the end of user of a software product, this can take the form of surveys or gathering usage analytics. A popular measure of customer satisfaction is the Net Promoter Score (NPS). In essence, NPS measures the percentage of customers likely to recommend a product or service to a friend. This is particularly useful for gaining an insight the overall user experience and noting specific areas for improvement.



$$\text{😊 \%} - \text{😞 \%} = \text{NET PROMOTER SCORE}$$

### Overview of computational platforms available



**Jira:** a powerful work management tool launched by Atlassian in 2002, suitable for various different work environments, not limited to software development. It is popular with larger organisational teams. The platform provides support tools for the software development process, including requirements development and test case management. In addition, Jira includes a litany of other features to measure and assess software that is developed. These include issue tracking, progress reports, estimation and work logging. One of the many reasons the software has become so popular is due to its integration with other cloud-based software management platforms such as GitHub and Slack.

Jira enables project managers to set tasks and deadlines for individual engineers. Managers have the ability to create “boards” and assign tasks to engineers, the status of these tasks are updated in real time as work is completed, Jira gathers analytics and estimates whether a task will be completed within the timeframe required. Automation is an immensely useful feature in Jira. Users can set rules, which check that a certain criterion within development has been met, they can then establish triggers which execute the selected rules. Actions can be completed automatically such as sending an email when a bug has been fixed.



### *Jira Dashboard Overview*

**GitHub:** the most widely used software development and version control platform worldwide. Since its inception in 2008, it has garnered over 40 million users. Multiple users can work on a project, usually held within a “repository” and commit code in stages, the site contains basics metrics tracking, such as, number of commits per user to a project, code review turnaround time and a focus on key details with filters by team, repository, and date uploaded(committed). Developers can add functions to an existing product and work in a separate “branch” from the main branch, which is referred to as the “master branch”, the benefit of this is that they don’t disrupt the workflow of the main project and after successful and implementation testing this side function can be “merged” with the main branch. Comments can also be added by participants (either to help explain their code or to request changes) in a project. Due to the fairly basic analytic system, GitHub is best used in conjunction with other platforms such as Jira that extend its functionality within an organisational environment.


  gibson042 approved these changes 23 days ago [View changes](#)

The concept seems reasonable. 🍌

```

src/ajax/script.js
48 48         var script, callback;
49 49         return {
50 50             send: function( _, complete ) {
51 51                 script = jQuery( "<script>" ).prop( {
52 52                     charset: s.scriptCharset,
53 53                     src: s.url
54 54                 } ).on(
54 54                 } ).attr( s.scriptAttrs || {} ).on(

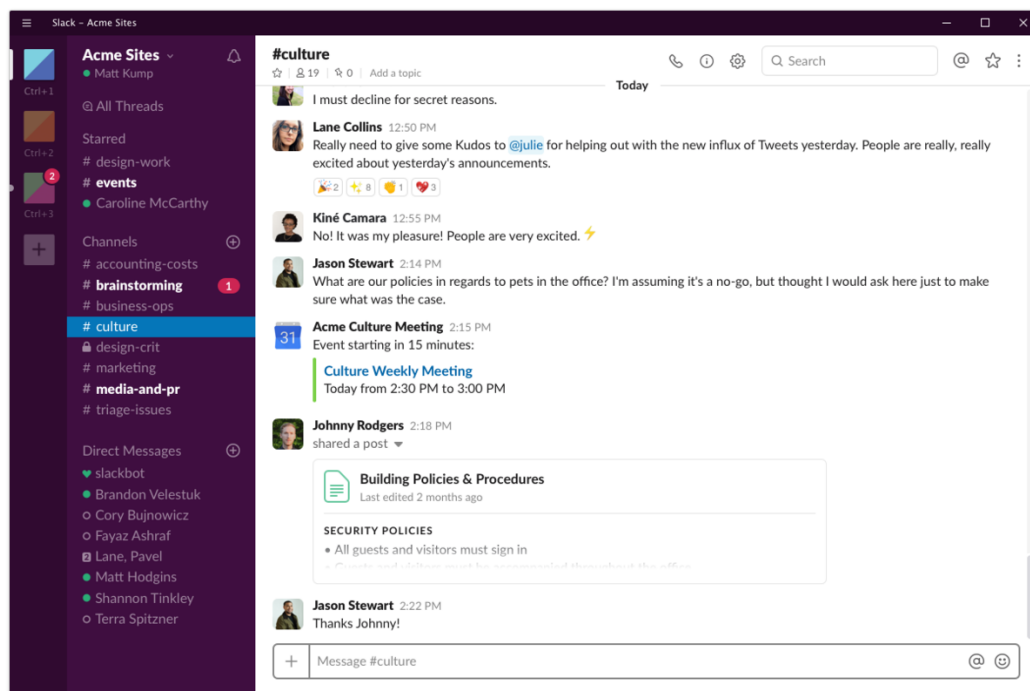
```

 gibson042 23 days ago Member

There should probably be a line break here.

### *GitHub code snippet with comments*

**Slack:** slack is a project management tool similar to Jira. However, is it directed more towards collaborators rather than managers. The platform allows users to maintain and prioritise tasks and modules. The live messaging feature is used by developers to communicate tasks, problems and updates. It has a modest user interface, in comparison to Jira. Slack is also compatible with GitHub. Developers create “channels” with in a specific area of a project, for example, a security section, relevant collaborators then join this channel and can see any changes made to the software.



Slack - Acme Sites

Ctrl+1  
Ctrl+2  
Ctrl+3

Acme Sites  
Matt Kump

All Threads

Starred

# design-work

# events  
Caroline McCarthy

Channels

# accounting-costs

# brainstorming  
1

# business-ops

# culture

design-crit

# marketing

# media-and-pr

# triage-issues

Direct Messages

slackbot

Brandon Velestuk

Cory Bujnowicz

Fayaz Ashraf

Lane, Pavel

Matt Hodgins

Shannon Tinkley

Terra Spitzner

#culture

19 0 Add a topic

Today

I must decline for secret reasons.

Lane Collins 12:50 PM  
Really need to give some Kudos to @julie for helping out with the new influx of Tweets yesterday. People are really, really excited about yesterday's announcements.

Kiné Camara 12:55 PM  
No! It was my pleasure! People are very excited. ⚡

Jason Stewart 2:14 PM  
What are our policies in regards to pets in the office? I'm assuming it's a no-go, but thought I would ask here just to make sure what was the case.

Acme Culture Meeting 2:15 PM  
Event starting in 15 minutes:  
Culture Weekly Meeting  
Today from 2:30 PM to 3:00 PM

Johnny Rodgers 2:18 PM  
shared a post

Building Policies & Procedures  
Last edited 2 months ago

SECURITY POLICIES

- All guests and visitors must sign in

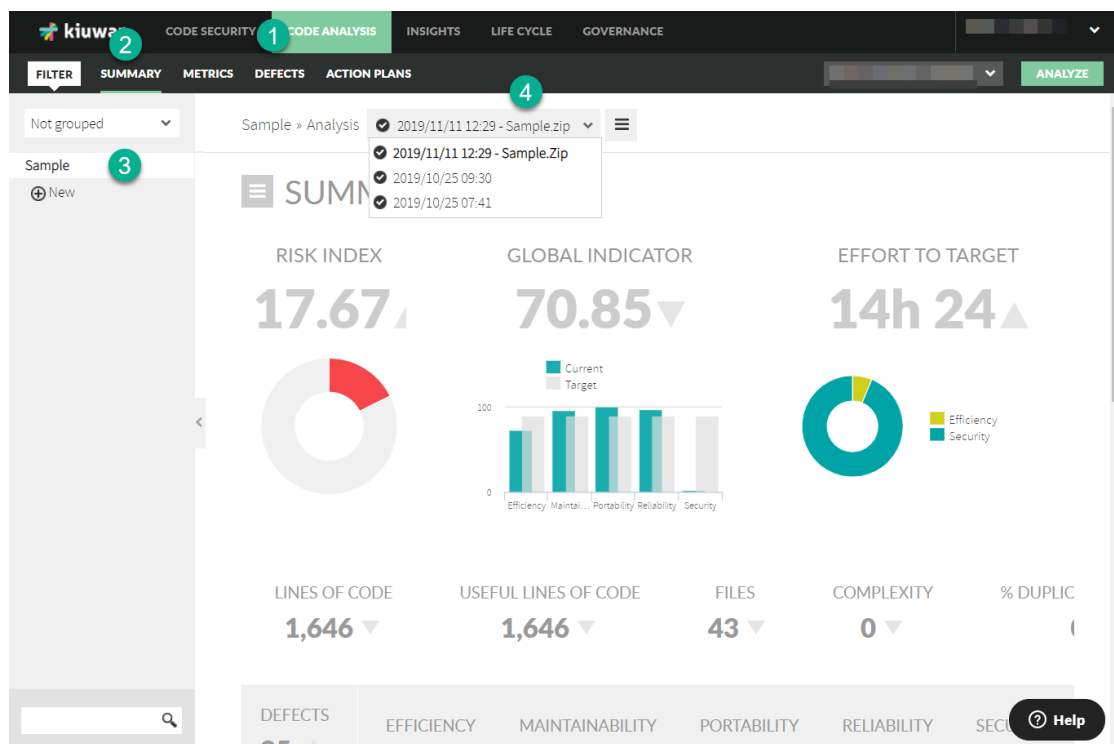
Jason Stewart 2:22 PM  
Thanks Johnny!

+ Message #culture

### *Slack Channels*

**Kiuwan:** offers a program for software analytics, quality, security measurement and management during the software development process. The measurement tool supports 32 languages including Java, Python and C sharp. An IDE plugin is available; the plugin can detect and fix security vulnerabilities when the developer complies their project. It can also automate the process of complying with security and coding standards. The plugin can be used either in viewer mode or analyzer mode. Viewer mode supports downloading reports, this makes it easier to create and assign tasks. Analyzer mode lets developers look at a specific file within a program in order to pinpoint the source of the bug. An impressive product offered by Kiuwan is the Code Analysis (QA) feature, allowing users to detect important defects in the software.

Managers can see pivotal data such as risk index of the code under review, defects by characteristic, language and priority. This reduces production incidents and errors detected in early tests by performing validations of source code, results in a subsequent decrease in development time. Additional features include technical debt (simply put time needed to rework software due to time cost choices) reduction. The platform can also create automatic action plans and make sure that they are being implemented during the development process. Using the Kiuwan SDK, a set of coding rules can be implemented for engineers to follow.



*Kiuwan Code Analysis*

## Algorithmic approaches available

### 1. Team Productivity(P)

$$P = \text{Kilo Lines of Code/Person months of effort}$$

Tausworthe (1982) defined overall team productivity in terms of P. W denotes the work effort (in terms of one person-month) and L denotes the production of L lines of code (in kilos) over this duration. Derived from a supporting formula he takes into account the time spent on communication overheads; defining a variable  $r$ , he came to the conclusion that the larger this value is the more effort that is required to shorten a schedule (inefficient productivity). While this equation is fairly simple, his research showed that it was possible for a project to have an optimal level of staffing and that software managers should aim to reach this ideal level to generate optimum productivity.

### 2. Simple Model of productivity:

*Physical productivity = Number of lines of code/Man hours (in days/hours or months)*  
*Functional productivity = Number of function Points/Man hours (in days/hours or months)*  
*Economic productivity = Value/Cost, where value = f (Price, Time, Quality, Functionality)*

In his publication Card explains three factors that contribute to the level of productivity in the software engineering process: physical, functional and economic productivity. Physical refers to the amount of effort exerted, usually measured in lines of code, but can refer to classes, screen or any other unit of measurement. While functional productivity refers to the ratio of functionality delivered to resources consumed, this can be measured in terms of use cases, requirements or function points. Economic productivity takes into account the cost of resources as a ratio to the value of the product produced (software).

### 3. Software Productivity (including software reuse)

$$\text{Productivity} = \frac{\sum_{i=1}^n (r_i + f_i + L_i + c_i)}{\sum_i}$$

where,  $R$  = reuse,  $F$  = functionality,  $L$  = length,  $\Sigma$  = effort

Nwelih and Amandin published a report building on the software reuse model created in 1989. They proposed a software productivity model that included software reuse, as using reusable software generally results in higher overall development productivity. Their model includes four main

data points:  $R$ , which takes into consideration that 1% of objects will be used from previous projects,  $L$ , the length of the program (lines of code + comments),  $F$ , the unadjusted function point count (UFC) and  $E$  which is the human input (effort) into the project (can be measured in LOC for example).

#### 4. Function Point Analysis

$$\text{Unadjusted Function Point Count (FPC)} = \text{External inputs (EI)} + \text{External outputs (EO)} + \text{External Inquiries (EQ)} + \text{Internal Logical Files (ILF)} + \text{External Interface files (ELF)}$$

Initially defined by Allan J. Albrecht in 1979 who worked at IBM. Function Point Analysis is a method for measuring software requirements, split into what are referred to as “Function Points”. Based on the interaction of the software system components with both internal and external users it is divided into five parts: External inputs, refer to the capturing and storing of input by the end-user. External outputs send data to outside users of the system. External Inquiries are a combination of both input and output components, relevant information is then generated from input/output database files. Internal Logic Files contain the data that is stored within the software system. External Interface Files refer to the data taken from outside the software system.

#### Ethics concerns and conclusive recommendations

The very nature of assessing and measuring the software engineering process requires a solid understanding of ethical principles, which the users of the data should be aware of. Two papers in particular deal with the issue of ethics along this process. In their report, “Ethics in Computer Software Design and Development”, Thomson and Schmoldt consider a number of aspects that can lead to ethical concerns within the measurement and assessment of the software development process.

Privacy is key area of concern. Analysing employee performance necessitates a certain level of access to personal information such as location, address, contact details etc. Each stage of the development cycle can raise concerns for confidentiality and can thus create problems in the deployment of the product. Privacy concerns are not limited to just developers but can affect the end users of the software.

Usually, unauthorised access to information can have unwelcome consequences. However, in some instances even authorised access can lead to ethical issues. A combination of information, taken from different data points can cause an invasion of privacy, referred to as data fusion. Thomson and Scholmdt provide a telling example, in which a different branch within a company can combine



pieces of employee information which results in an inaccurate and invasive product that outside parties have access to.

A general perception exists that the work of an individual in a company is the property of the organization. This can blur lines of intellectual property rights of an individual employee, it is therefore important that measures are taken to attribute work to those involved, particularly during analysis and measurement of the development cycle.

As stated, ethical issues, in relation to metrics tracking, concern not just engineers and employees but can affect the end-user of the software. According to Thomson and Scholmdt, software systems have a fairly poor ability to predict "error states", this can lead to several issues surrounding the spread misinformation and involuntary disclosure of private information. They explain a possible solution from, and an engineering standpoint is to have a clear documentation of assumptions and appropriate testing conditions during development.

The data provided by analytical methods, undertaken by companies have an impact on worker remuneration, advancement and in severe instances whether they are made redundant. It is thus imperative that accurate systems of measurement and analysis are employed by the company to prevent unfair evaluation and subsequent action. Fenton and Neil assert that many of the assessment techniques used by large organisations are inaccurate and outdated. For example, the lines of code measurement were used by some companies, as discussed, this is not an accurate indicator of the overall value that a developer brings to their role but rather the volume of the work they produce. They explain that in order to develop accurate metrics the objective should be "to handle the key factors largely missing from the usual metrics models: uncertainty and combining different (often subjective) evidence.", as the relatively simple metrics are inadequate for this purpose.

## Bibliography

- Behreins, Charles A. "Measuring the Productivity of Computer Systems Development Activities with Function Points." *IEEE Transactions on Software Engineering* 9.6 (1983).
- Card, David N. "The Challenge of Productivity Measurement ." 2006.
- Fenton, Norman E. and Martin Neil. "Software metrics: successes, failures and new directions." *The Journal of Systems and Software* 47 (1999): 149-157.
- Frakes, William and Carol Terry. "Software reuse: metrics and models." *ACM Computing Surveys* 28.2 (1996).
- Grist Project Management. *Albrecht function point analysis*. April 2017. November 2020. <<https://www.gristprojectmanagement.us/software-2/albrecht-function-point-analysis.html>>.
- Infopulse. *Top 10 Software Development Metrics to Measure Productivity*. 19 October 2018. November 2020. <<https://www.infopulse.com/blog/top-10-software-development-metrics-to-measure-productivity/>>.
- Nwelih, E. and I.F. Amadin. "Modeling Software Reuse in Traditional Productivity Model." *Asian Journal of Information Technology* 11.7 (2008): 484-488.
- Peterson, Kai. "Measuring and predicting software productivity: A systematic map and review." *Information and Software Technology* 53.4 (2011): 317-342.
- Sudhakar, Goparaju Purna, Ayesha Farooq and Sanghamitra Patnaik. "Measuring Productivity of Software Development Teams." *Serbian Journal of Management* 1.7 (2012): 65-75.
- Tausworthe, R.C. "Staffing Implications of Software Productivity Models." TDA Progress Report. 1982.
- Thomson, Alan J. and Daniel L. Schmoldt. "Ethics in computer software design and development." *Computers and Electronics in Agriculture* 30 (2001): 85-102.