Measuring Software Engineering

Peter Queally

14320108

CS3012 – Software Engineering

Measuring Software Development

When attempting to measure software engineering, there are many factors to consider, with none giving us a truly accurate picture of the whole situation. It is important to use as many metrics reasonably possible in order to get as accurate as possible a view of the performance. These metrics include performance of the software being produced, the security of the software, the productivity of the software development team and the quality of the source code.

Performance of software produced

The following metrics are used to measure the quality of the software. These will not explain specific features of the program but can provide good data to figure out failures going on with them.

Mean time between failures (MTBF)

MTBF is the average time elapsed between failures of a piece of software during normal operation. In other words, it is the time elapsed between the resolution of a system failure and the discovery of another failure. This metric is useful as it determines how stable the system is since it shows roughly how many failures occur.

Mean time between recovery (MTBR)

MTBR is defined as the average time required to troubleshoot and repair a software system and return it to its normal operations. This metric can be used to measure the maintainability of the system, as well as its reliability. MTTR is calculated by dividing the total maintenance time by the number of repairs over some specified time.

Application Crash Rate (ACR)

This metric is related to the last two and measures the rate at which a system fails. Calculated by dividing the number of times an application has crashed by the number of times it has been used. This metric provides a look at how the system is performing but is not very useful for predicting failures.

Security of software

Security of software is a crucial part of software development which mustn't be overlooked. Security metrics are tracked over time to show how software development teams develop the security responses.

Mean Time To Repair (MTTR)

This specifically refers to the average time elapsed between a security problem being discovered and the eventual resolution of the problem. This metric should be tracked over a specified period of time and should be compared to historic data if available. If MTTR is decreasing over time, the software development team is becoming more effective at handling the security issues. The opposite occurs if the MTTR is increasing.

Productivity of Software Development Team

Team productivity can be measured using multiple very useful metrics. It's important to make sure the productivity level is where desired or else application creation will be very inefficient. For a leader of a software development team, data is important to use in order to keep track of the team's performance and make decisions based on it. The following metrics may be used by the leader to evaluate the team's actions.

Velocity

Velocity is a metric for work done, which is often used in agile software development. Velocity always involves counting the number of units of work completed in a certain interval. The main idea behind velocity is to help teams estimate how much work they can complete in a given time based on how quickly similar work was previously completed. Velocity is a relative measure, which means the raw numbers matter very little. It's the trend that matters most.

One problem with velocity is that it conflates work done with planning accuracy. In other words, a team can inflate velocity by estimating tasks more conservatively. A second problem with velocity is that it does not take quality, alignment with user goals or priority into account. Velocity can be increased by neglecting good design, refactoring, coding standards and technical debt.

Cycle Time

This metric is simply the time it takes to bring a task or process from start to finish. Longer cycle times often mean the development process is inefficient, while shorter cycles usually mean an optimized process and a faster time shipping the product to the market. These cycles are commonly split into sub cycles to more accurately measure progress.

Code Churn

Code churn measures the changes made to a database of code over a period of time and quantifies the extent of these changes. It is important for a software development team as it can be used to predict defect density. It has been shown that certain code churn metrics correlate highly with flaws in the code. These metrics have a role in driving the development and test process, as highly churned code is more prone to defects and should be tested more thoroughly and more carefully.

Lead Time

Lead time is the length of time it takes from the team creating an idea to delivering the software. Lowering the lead time is a great way to improve the responsiveness to customers and clients of developers. This isn't the most useful metric during development process but is a great reflationary tool.

Quality of source code

Another way of measuring software development engineering is analysing the source code itself. This can be done by the follows:

Lines of Code

This is simply counting the number of lines of code the development team writes in a given period of time. You can compare this to historical data to see if progress is increasing or decreasing within the team. If the productivity of a software development team is measured by the number of lines they've written, this will lead to several problems, the most obvious being that engineers would be incentivized to write unnecessarily long, redundant code so they appear more productive than they actually are.

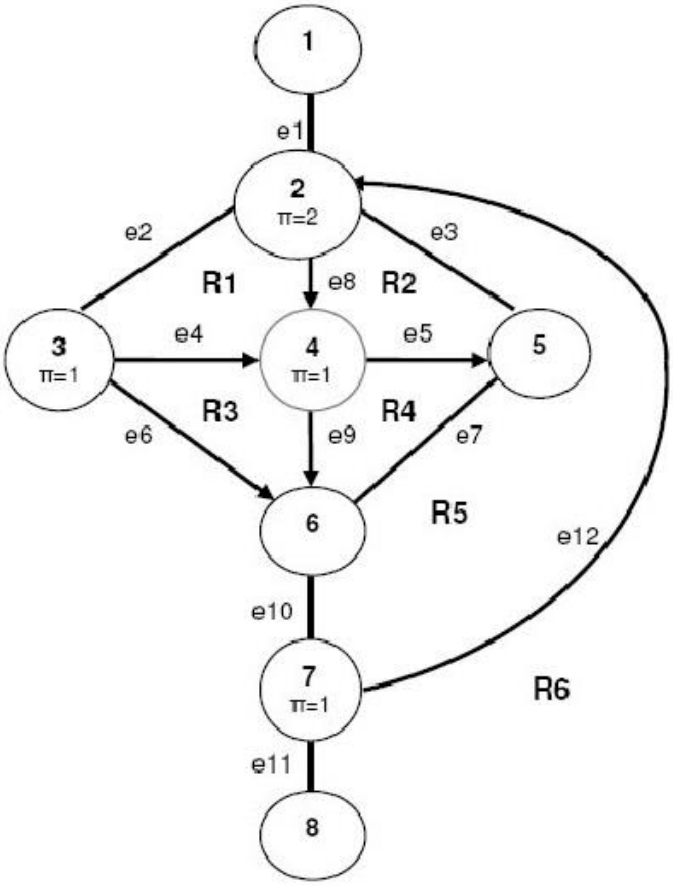
Cyclomatic Complexity

This is used to indicate the complexity of a program. It is a quantitative measure of the number of linearly independent paths through a program's source code. One testing strategy of this is to test each linearly independent path through the program; in this case, the number of test cases would equal the cyclomatic complexity of the program.

Edges and Nodes Method:
$v = e - n + 2$
$e = 12, n = 8$
$v = 12 - 8 + 2 = 6$

Predicate Method:
$v = \Sigma\pi + 1$
$\Sigma\pi = 5$, sum of predicates
$v = 5 + 1 = 6$

Region (Topological) Method:
$v = \Sigma R$, sum of regions R
$\Sigma R = 6$
$v = 6$



Computational Platforms Available

Developers have a multitude of tools at their disposal to measure the quality of their software development. One of the most popular approaches is to use one of the available open source automated code review tools such as Codacy or Sider.

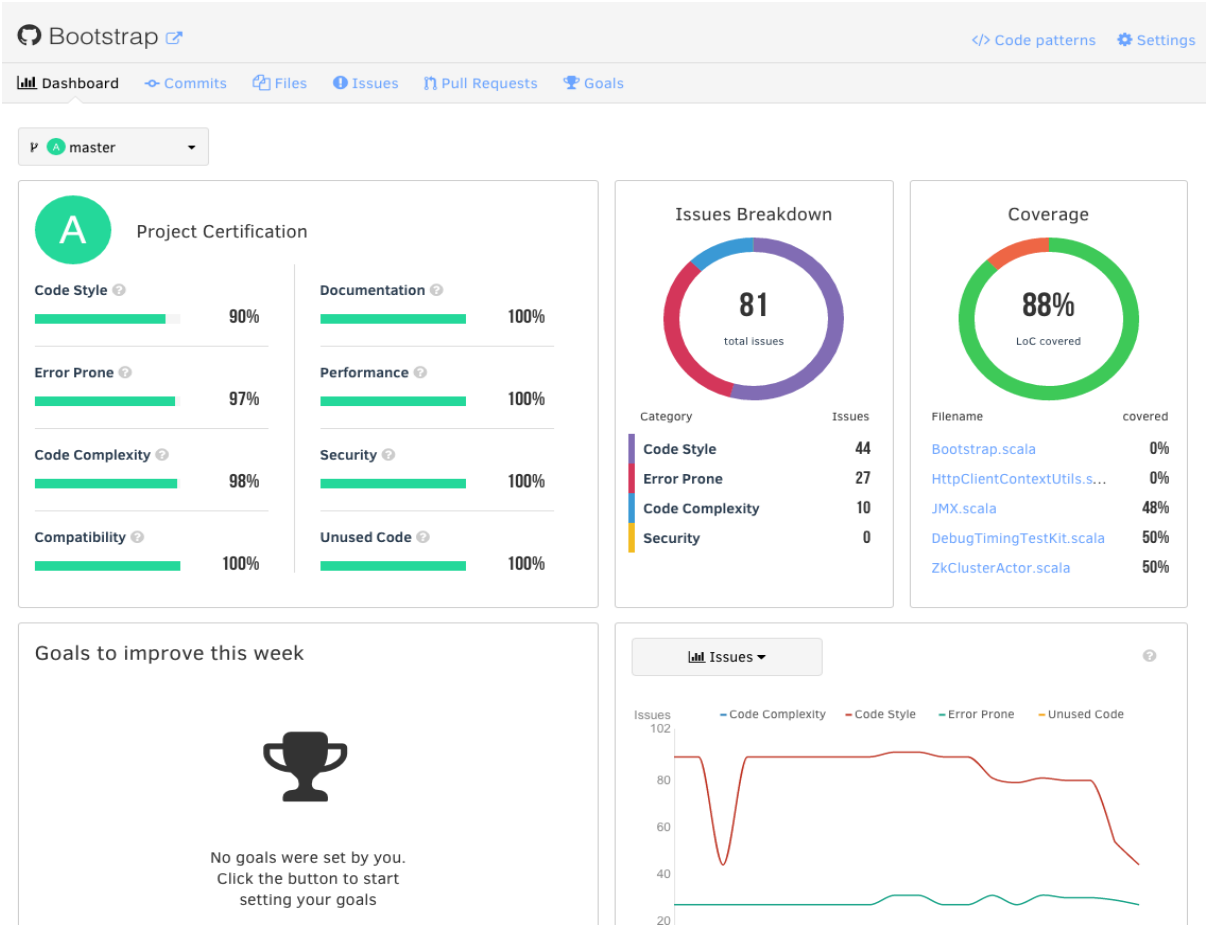


Figure 1 Codacy Dashboard

Each have their own benefits and drawbacks, leaving it up to developer preference.

Code Review Tools

Codacy is an automated code analysis/quality control tool that is integrated into your online version control tool. Among the metrics it provides are cyclomatic code duplication and code coverage changes from unit testing with each commit.

Companies use tools like Codacy to enforce their own respective coding standards onto their employees' code. It also comes with built in docker analysis and extensive security checks, and allows users to define their own patterns which are checked automatically.

Version Control Tools

Git and Bitbucket can be very useful in measuring software development, as companies can track and record various metrics through the use of the repository system. For example, the commit frequency can be measured alongside the number of lines of code written.

GitHub and other similar tools measure code-related metrics themselves. For example, one metric is the counting of lines of code either added or removed in a commit, as well as the exact lines edited by the developer. It is possible to visualize this data into graphs also using GitHub's API and 3rd party graph services. These graphs can be used by development team leaders to show their sub-ordinates their progress and criticize them accordingly.

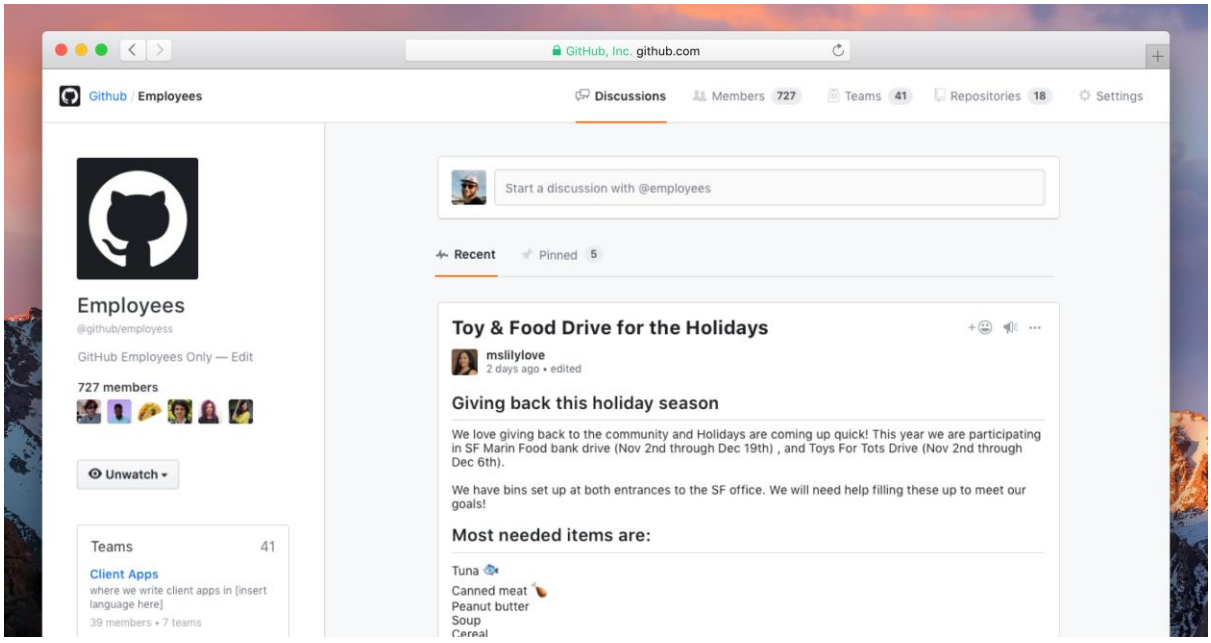


Figure 2 GitHub screenshot

Algorithmic approaches

This section covers the possible algorithmic approaches available to software engineers when they are developing their code.

Lines of Code

There are positives and negatives to using lines of code in measuring performance as discussed before in the report, here is how it is calculated.

There are three main ways to go about this:

Firstly, simply counting each separate line of code in the source file that is being analysed. A text editor can carry this out for you. This can be useful for the quick indication of the scale of a project.

Secondly, we can count the number of logical lines, which is the number of statements in a piece of code, rather than the number of lines. A statement may be a line of code, or it could be several. In Java, C/C++ and C#, one quick method of achieving this is to count the number of semicolons, though this may not be entirely accurate due to the existence of loops and other various coding process.

Thirdly, the number of lines of code the program compiles can be counted. In other words, the number of executable statements the code compiles during runtime. This is a more reliable method as there is no worry about code formatting styles or coding processes that are occurring the code. However, this method does not include abstract methods or interface definitions, which are important features in the code, and so is not perfect.

Cyclomatic Complexity

As previously explained, this is the software metric that can determine the complexity of a piece of code by allotting the number of linear independent paths through the code.

It can be computed by using the code being analysed to build a control flow graph, made up of nodes and directed edges. To translate code to a graph, the nodes must each correspond to a command until each command is given a node. Two nodes are then connected to each other with a directed edge if the second node is to be executed directly after the first one in its path through the program. Thomas McCabe Sr, the developer of cyclomatic complexity, defines it as V(G) = E – N + 2, where E is the number of edges and N is the number of nodes.

Ethical Concerns

Ethical concerns play a huge role in measuring developer productivity. Incorrectly used methods to measure productivity can easily backfire and cause a decrease in productivity and create a toxic work environment.

For example, if developers were being criticized regularly for their work too harshly and the results were posted online for everyone to see, it would make it very uncomfortable and unpleasant for the developer to work in that environment and would most probably intimidate them and destroy their confidence. Other employees may jeer at those who aren’t performing well for whatever reasons, which will mostly lead to worse results.

Another concern would be that of privacy. Managers and supervisors must be sure to obtain consent from their developers for them to monitor their performance and utilize their data. If this does not happen, employees may revolt against their leaders due to discomfort and dissatisfaction.

Conclusion

As seen above, there are many different ways to measure the software engineering process. However, there is no one good way about it, so multiple methods must be incorporated to get as accurate a measurement as possible. Even then, you can never be sure if what you're measuring is the correct metric to be focusing on.

Every company/leader/manager should look at the development team as a whole as well as the product to be delivered and work out the best metrics from there. Ensuring all ethical concerns are met is of utmost importance as well, in order to make developers content.