

Measuring engineering

By Gabin TODESCO



Software engineering

Trinity college

Michaelmas Term 2019

Summary

[Introduction 2](#_Toc24149182)

[Software engineering metrics 3](#_Toc24149183)

[Productivity metrics 3](#_Toc24149184)

[Process metrics 5](#_Toc24149185)

[Quality metrics 6](#_Toc24149186)

[Notes (temporary) 6](#_Toc24149187)

[Data measurement methods 7](#_Toc24149188)

[Available platforms 7](#_Toc24149189)

[Algorithms 7](#_Toc24149190)

[Sources 7](#_Toc24149191)

# Introduction

Software engineering is a really important part in the computer science world as software is a strong link between computer science engineering and the other industries, it is an important component of our modern world. For a long time companies have undertaken software development projects and as they are companies they are constantly trying to improve the development part to get faster into the exploitation phase of their products. Over time a lot of investment has been made to develop methods and tool to increase the development process efficiency and boost the teams productivity, to speed up the process with out losing in quality of the final product. The best way to improve the efficiency of a development team is to get feedback on their actions, the way they work and its evolution through time, keeping track of their improvement. Therefor the way to do this is using a chosen set of indicators, measurements : software engineering metrics.

The software engineering metrics are useful in two ways, first the software engineering metrics that measure the development, the way it is built and the software or application performance metrics that measure the performance, the customer satisfaction of the final product. Here we are focusing on the first type of metrics, the ones that can be called software development metrics or software delivery performance metrics, overall the metrics that measure the development of the software.

There are many kinds of software engineering metrics that but it is important to know when to use which one and how. The metrics are indicators, they are useful to determine the quality and productivity of a delivery process and identify the areas of improvement, predict the future of it and help to manage the resources, workloads, priorities, teams and team members, it helps the take of decisions about the project direction.

It is important to make the distinction, those metrics are measuring the development process and are indicating the global state of the development, but it should not be use as a reference assessment for the productivity and efficiency of a team or an individual. The data that those metrics are showing can not be used as so, it is the work of the manager, if he estimate that there is possibly a problem, to talk with the one(s) involved to know why and how the data are so. It is the responsibility of the manager to discuss with the team to understand the metrics results and conclude of the improvements that are necessary. The productivity and efficiency of an individual is hardly measurable as it depends on so many parameters that a software couldn’t handle, the measurements of the delivered work is not representative of the work effort produced by an individual or a team and should not be used as so. Too many companies are focusing on the numbers but don’t get their real meanings.

# Software engineering metrics

It is important to determine which to use and when because some could just don’t make sense or not be representative of something useful for your team improvement. And some can be just bad for your team, showing something your team should not focus on will make them want to make the values better even if it does not help the working process. A good metric is easily understandable and make sense to your team, it must be simple, computable, consistent, unambiguous, independent of the development environment (programming language) and easily calibrated and adaptable. The chosen metrics must be coherent with the company priorities, you can’t focus on everything. The best is to take a fewer objectives to focus on and define their priorities, the most important objective(s) and the secondary one(s). Also what is important is not the numbers themselves, but their trend, you observe its evolution through defined shot periods of time (usually a sprint).

## Productivity metrics

These metrics are measuring the productivity but they are the most controversial ones because they are usually badly used and over time development teams learned to hate them, because they are not representative of the work effort, because a developer that spend a week to solve a big problem that affects all parts of the software will have less good result as the one who made 20 minor bugs corrections or features implementation that are not mandatory, the second is technically more productive but the most important for the project is the first one. For most of them it is necessary to look at the evolution of the team values over time to find and compare to the “normal” speed and productivity of that team so it is easier to find out that the team is struggling and that some members are stuck on some tasks. It is necessary to fin the points that can be enhanced.

**Project or Sprint burndown**

This metric is about the project status, its advancement. There are a few different ways to measure it, some considers the number of tasks to be done, but it isn’t really representative as all the tasks does not have the same importance in reality. A way to improve this method is to define importance points (story points) to each task, but some task can be harder than it seems and with the last example, resolving the big problem would be an around 5 points task that would be way less than doing 20 minor tasks of 1 point each. You better be comparing the team’s progress on the current release over previous ones to ensure that the development is on time.

**Ticket close rate**

It is the number of tickets or story points the team solved during a certain amount of time (usually a sprint). It is one if it is not the most misleading metric if using the ticket count, and using the story points does not make it really better. This is the most relevant example of why you should never use metrics to evaluate individuals, because as we seen with the example, some tasks require a big amount of time, and as the ticket points are not referring to the amount of time the task require but its importance, it is not representative of the real work that is being accomplished. However this metric can be useful to identify if a developer is stuck on a specific task, it can be a way to see that someone is not made for the kind of work he got assigned to. So again, it is important to see the metrics data as a raise of attention and to lead to discussion with the team, a good way is again to compare with the “normal” values to spot a problem.

**Line Of Code (LOC) produced**

This metric is measuring the number of code lines added, modified or deleted, but such as for the ticket close rate it is not actually relevant. Taking an example, a major bug fix can be made changing only one line of code after many researches and tests when another developer get the credit of hundred lines for importing a library, changing the headers or the comments… It is hard to find a situation in which this metric is useful because the number of lines depends on many things, the same feature can be made with 1 to 10 lines of codes for the same result. But it can eventually show if a developer is stuck on a task.

**Code churn**

The code churn id the ratio of a programmer code that is an edit of its own recent work expressed as a percentage. It is the amount of added, modified and deleted code lines over a short amount of time divided by the total amount of added code lines. A high churn can be caused by a non-productive work, the existing features are being modified and nothing is being created/added, the project does not go forward. But the thing is that developers usually test things, mostly at the beginning when problems does not have clear solutions, they try things, change a lot, and when they find the correct thing to do they do it all over again to do it clean, to do it well. Yes unclear requirements, indecisive stakeholder, difficult problems, prototyping and polishing can be the cause of a high churn, but the rework can be for optimization of the code to get better performances, of fixing a bug… The churn will change along the project and it does not necessarily means there is variance in the work done, it depends on the team, on the work, the specifications… But again it is interesting to keep an eye on the evolution, a sudden raise of the chum can indicate that a developer is having a hard time solving a problem. It is important to find a good balance between a project that is constantly being enhanced but does not goes forward and a project where new features are added everyday but nothing is being reworked and it can lead to serious problems such as major bugs, unoptimized code, a loss of performance and quality. So only the big and sudden changes in the churn rate are good to take in consideration.

**Refactoring rate**

The refactoring rate is similar to the code churn but concerning the old code, the legacy code. Refactoring code is sometimes necessary to enhance the software, to allow the add of new features, to make a code optimized for the use we have of it, keeping everything working with the new changes. Even if it is actually hard to compute actual refactoring, it can be useful to see the trend of the team and its evolution through time, ensuring that enough refactoring is done as it is an essential part of the software development.

**New work**

Complementary to the code churn and rework, the new work rate is the ratio of added codes lines that does not replace anything over the total number of code lines added, modified, deleted. As I said, having rework and code churn is necessary, but for a company, having a stagnant product is worse than having technical debt. A project must go continuously forward even if it slows down, it have to continue going. A good thing is to observe the rates of new work, code churn and refactoring all together to see the trend of the team. The values will evolve along the state of the project and the team’s trend that it is showing permit to identify the tasks on which the team is focusing and allow to act if you see that the team is not focusing on the same objectives as the direction wants.

## Process metrics

The process metrics measure the performance of the software development process, they are an indication to the collaboration of your team as a well formed and collaborative team will be more productive and deliver better quality result.

**Lead time and cycle time**

The lead time is the time it takes from the start to the delivering of a project. Having the history of the previous lead times of your team is useful to estimate the time it will take of the current one. It is good to know the time it usually takes for this team to do a similar task. The cycle time if when using continuous delivery, the time it takes to change the software system and implement that change in the production. It is a more precise measure and permit to understand what part takes the more time. The lead time for example can be easily reduced simplifying the decision making and reducing the wait time.

**Deployment frequency**

The deployment frequency is the frequency of the deployments. It is better to have fewer changes at each deployments but to have more of them and having them often. A smaller deployment is easier to test and release. It is a good complement to the lead time and cycle time as the comparative with them evidence the result of the time spent.

**Commit frequency and active days**

Commit frequency and active days measure the active times of your team allowing you see the days during which your team has been active and show the way non coding tasks are affecting the work. The non-coding tasks such as planning, meetings, review… are inevitable for the health of the development process, but those tasks should not be a burden to your developers, with these metrics you can monitor the impact those tasks have on your team. As pushing code is the principal way to bring value to the project, it can be interesting to monitor the commits of each day and make the parallel with other metrics. Also, it is a good way to see if the commit rate is sufficient, it is well done to have as much commits as possible to provide saves of your work and the fewer there is in a commit, the easier it is to manage it.

**Pull requests-related velocity**

This metric is about the number of pull request during a defined amount of time, it can be the number of pull request opened or the number of pull request merged per week or the average time to merge which is related to the cycle time. It gives you a good preview of the global work velocity of the team. As for the productivity metrics it is interesting to watch the variations of these values as for instance, if the values does not raise hiring more people in the teams, that means there might be a problem in the process like technical debts. And if the values are increasing too quickly it can be a sign of a quality issue.

**Work in progress**

The work in progress is the measure of the number of tasks that are currently opened and worked on, it is a real-time indicator of the team’s current workload. This metric should be stable through the process, again it is good to monitor the changes in the trend. If the value increase it can means that the team is facing some issues, some problems they are stuck on. This number should also be close to the number of members in the team, a good way to verify that is to measure the number of opened tasks per contributors. A developer that works on too many tasks at a time won’t be as efficient as if he was concentrated on only one. It is an important thing that can be improved.

**Commit and pull request risks**

During the development of software it is common to encounter versions problems, developers are coding and pushing code and somehow the application has bugs, it means one or more previous commits produced a corrupted release. To prevent this to happen or at least decrease the chances that it happens, it can be useful to monitor the commit and pull request risks, I mean the chances that a commit/pull create an issue. It can be calculated measuring the amount of changed code, the percentage of it that edits old code, the number of edit locations in the files, the number of files changed, the severity of the old code editions, and how much changes compared to other requests in the history of the project. The goal is obviously to keep the risks down and one way to do it is to increase the commit/pull frequency as it lower the changes in each of them.

## Quality metrics

The development process take place on three dimensions, the quantity, the quality and the time. The goals of the metrics is to improve each one of them without having a negative effect on the others, it is hard to find the perfect balance as a better quality means more time spent on each feature so either more time or less features, and more features implies more time spent or more risks on the code quality… It is important for a project management to take in consideration all these parameters as each one will have an influence on the result. The quality metrics are used on releases as a good quality isn’t just good for the customer, a code of quality will be easier to work on and as a friend of mine usually says :

“You have to learn to lose time to gain time” (translated in English) – Thierry Charles a Fortinet engineer

What he meant by that is that going straight into code will surely save you the time of conception, but you will spend twice as time trying to repair the things you could have done better. Taking time to think about your project and the ways to do it well will make the realization easier and as everything is done well it is easily improved and you won’t have to modify the base code. Overall quality may not be the first or the main goal you think of in software development but it should not be neglect.

**Number of bugs**

The number of bugs and its evolution through time is a good information to have, it should increase around the middle of the development process and stabilize. The goal is, a few weeks before a release, to focus on reducing the number of bugs and focus on the most important ones. To do so it can be useful to assign a level of priority to the bugs, the number of bugs on a release will determine the overall quality of your product and so the company’s brand reputation. As usual, it is good to compare the values with other metrics to have a good view on the project state.

**Change failure percentage**

A change failure is a change that degraded features or result in a problem that needs to be reworked. The number of failure is useful to judge the global quality of the development process, the number should decrease over time as your team takes experience, if it is increasing or high and not decreasing it means there is a problem in the process but this metric alone will not be able to tell you where.

**Pull request quality**

As I was saying a bit higher, making a solid base for your software is important, the more your base code is well made and versatile, the easier it will be to work with it without having to modify it. But a common mistake is to make the base code way too complex trying to handle every possibility, but the more complex the base code is, the higher chances are that the rest of the code will be complex, and a complex code is a code hard to maintain, hard to modify. Measuring the pull request quality can give you an idea of the team’s collaboration and the quality of the code pushed to production. Monitoring that metric over time will show you if the team and team members are improving themselves. This metric can be measured by the number of pull requests that break the build divided by all the pull requests in percentage, the percentage of merged vs rejected pull requests or the number of comments by pull requests (too low and the code will be hard to maintain, to high and the work isn’t efficient).

**Test coverage ratio**

Well done, tests are necessary to tell the reliability of a program, if the tests are good which usually depends on the requirements, the more you test your program the higher the chances are to find a bug and be able to correct it. So testing ratio is an indication of your code quality, it is measured by the number of lines of the software you are testing over the number of test cases. It show how much of the program is covered by tests and how well the coverage is.

**Mean time between failure and mean time to recover**

These metrics show the evolution of the software through development, as failure is almost unavoidable, it is a good indicator of your team’s trend about the level of standard they have for the product. It is practical to set level objective to be able to keep your team on the same standard level as the requirements.

**Service level agreement**

This metric similar to the mean time to recover as it measure the number of bugs fixed and deployed by your team within a certain amount of time. It is a good way to have the quality perception from the point of view of the customer. Setting an objective of SLA is a good way to determine the final product quality.

**Defect removal efficiency**

The defect removal efficiency is an indicator of your team’s efficiency to remove defects from the product. It measure the number of defects found after the delivery divided according the number of errors found before delivery, the closer this number is to 1, the fewer defects are found after delivery. It is in some ways similar to the number of bugs metric but does not take in consideration the importance of them.

**Application crash rate**

This metric measure the times the application crashed over the number of time it is used. It is reflecting of the quality of an application, obviously the goal is to make it crash as rarely as possible.

**Defect density**

This one is about the density of defects in an application, it can be computed with the number of code lines or the number of functionalities. But the number of functionalities are hardly measurable and the number of code lines depends on the language, this metric should not be used as a comparative with other projects.

**Age of dependencies**

Monitoring the age of your dependencies can make you aware that some of them could require your attention. It is an indicator of the possible technical debts and pushes you to consider all your branches.

Those 32 metrics are the more popular ones, there is many more used but some are controversial and does not provide a clear and representative value to the manager.

# Ethics concerns

Monitoring the activity of individual has always brought ethics concerns, no one wants to be monitored for every action he is doing. Not going too far into dystopia, giving the ability to keep an eye on every action to someone is scary. Monitoring the impact of the individuals on the project is only in a professional context and it can help a lot the teams for guidance into development. Now, it is understandable that companies wants to keep an eye on the development process as it is a return on investment, but as with everything there got to be a good balance. A balance between monitor the actions of people that influence on the project and monitoring every aspect of a person to evaluate an judge its ability to bring something to the project. All the metrics I have listed earlier are monitoring the contribution of people onto the project and the project itself, but there are some metrics that focus on things that are irrelevant and should not be used to judge the capacity of anyone. Some metrics such as the whitespace complexity metric focus on the way the code is written, but it is not relevant as the complexity of a syntax depends on the language and is not linked to the quality of the code. Some metrics are more focused on individuals than on their actions, for example some metrics measure the knowledge of a person on specific topics before even getting started on the project, I know this kind of metric is supposedly practical for the manager to help the team members but for me it is approaching the line that separates what is socially acceptable. I personally think that it is not relevant for the project and not necessary, I would feel bad to be judged by an algorithm over my knowledge and not my actions. But in this example it is still normal because the metric is used as it should be, a metric is an indicator, it measure things but cant automatically judge of something, a metric brings indications for the manager to discuss with team members. The manager needs to ask why the metrics values are like this and how, and this can only be achieved by discussing with the team. No metric should be used to directly judge of a team or team member work/work effort/capacity… because the values are not representative of the reality, such as a dashboard indicator light in a car can significate multiple things. Metrics measure consequences but there is almost never an only cause, it is the job of the manager to find the right cause. So using metrics onto individuals are bad because it is not representative.

# Notes (temporary)

The metrics alone can’t tell you the story, it should always be a discussion starter only. Don’t evaluate contributors on the metrics, looking for why the metrics are like this discussing with the protagonists will tell you what the metrics actually means. Metrics are indicators but can’t be relevant of an individual effort. An human interpretation is necessary to find what happens in a team.

Story points (define points for every task) but a 5 points task can be longer to implements than 20 1point tasks. Ticket close rate, the number of stories/story points the team/individual has solved during a certain amount of time [one of most misleading, never to use to evaluate individuals] can evaluate the average speed of the team and identify if someone is stuck on a task.

Lines of code or impact.

Code Churn, the percentage of developer’s own code representing an edit to their own recent work, could be the representation of a not efficient team or a team that is enhancing its code, depends on many things but is a good way to se if something is going badly compared as “normal” values.

Refactoring Rate, show the rate of refactored code over the total changed code. As the Code Churn it can show a problem when it is far from the “normal” values. There should be a good balance between maintaining code and creating new features.

## Data measurement methods

* **Function Points Analytics (FPA)**
* **Software engineering metrics** : measure the software team productivity, how the software is being built
* **Software or application performance metrics** : measure the delivered software (response time…) = customer satisfaction.

## Available platforms

* [Mavenlink](https://start.mavenlink.com/project-management-software-uk/?utm_source=google&utm_medium=cpc&utm_campaign=Search%20-%20Project%20Management-Broad-%20UK%20-%20IR&utm_content=project%20tracking%20software-%20broad&utm_term=project%20tracking%20software&utm_match=b&gclid=Cj0KCQiAno_uBRC1ARIsAB496IU5bot6KvHfwLriBFNsul-Yph-IbYpWOZP3tSNvmx7X-60iFi2Df64aAnHHEALw_wcB) : Project management, accounting, time and expense tracking, resource management, business intelligence. Monitoring financial performance.
* <https://anaxi.com/> Anaxi (development plateform)

## Algorithms

## Sources

<https://hackernoon.com/how-to-use-and-not-abuse-software-engineering-metrics-3i11530tr>

<https://anaxi.com/software-engineering-metrics-an-advanced-guide/>

<http://context-driven-testing.com/?p=49>