MEASURING SOFTWARE ENGINEERING

BRANDON DOOLEY

#16327446

TRINITY COLLEGE DUBLIN

Introduction

In today's world we live in a software industry filled with a rapid transition from development, to testing, to production thanks to methodologies such as Agile development and DevOps. This rapid deployment pipeline comes hand-in-hand with a responsibility and need for companies/organisations to measure and assess their developers from a professional standpoint.

Within these continuous integration and deployment environments there is a pressing need to rapidly ship new versions of code and update products on the fly. But the question is how can the organisations employing these developers draw useful metrics from this workflow to ensure productivity levels remain high, work is distributed evenly and tasks are being completed across the entire team.

There have been numerous attempts to measure the productivity of software developers within the industry. When measuring productivity in any other work driven environment it can be quite simply reduced to the equation:

$$Productivity = Output \div Input$$

But what defines input and output – especially useful – metrics within the software industry is quite a hard question to grasp a conclusion on. Is it as simple as counting the lines of code from a given developer per day? But what if these lines are defective/useless or that developer had in fact been researching into a current task to ensure that it is completed efficiently and correctly. Should a developers hours spent coding be tracked and used as input metrics? These are the questions that will be discussed throughout the following report.

Productivity must also be considered from a number of different perspectives. It is not enough to just evaluate the productivity of a given developer within an organisation. This developer/individual must be analysed in parallel with that of the organisation in which they work. When attempting to measure organisational productivity one must account for the productivity of all those who influence the life of a developer such as those within other departments and also their respective managers.

The overall goal of tracking and analysing software metrics is to determine and improve the progress of a project and determine the end quality and estimations. Project managers who use such metrics aim to; increase return on investment, identify areas which need improvement, manage workloads and overall – produce a product of desired quality at an optimal cost and time balance.

Measuring and Gathering Metrics

There are endless methods of gathering metrics from a software development team ranging from counting the lines of code written by a given developer on a given day to measuring how much of an impact each minute spent working by that developer has had towards a given project.

When approaching a basic metric such as how many lines of code are written by a given developer each day there are quite a number of factors that must be taken into account. Just because a developer has produced over 10,000 lines of code this week does not mean that these lines are optimised, bug-free, the best approach and easy to read. This output must be measured in parallel with the testing done on the code such as through methods of unit testing and code coverage to ensure all possible scenarios have been accounted for and handled correctly. Unit testing and Test Driven Development (TDD) can add an additional 15-35% increase in initial development time which may come across as substantial, but this increase in development time also brings with it a 40-90% decrease in pre-release defects. From this same survey 78% of developers thought that TDD improves the overall productivity of the programmer and 92% believed that TDD yields higher quality code. (George & Williams).

While TDD metrics may be an excellent way of measuring your developers productivity from a usefulness standpoint, consider a scenario where a method is found with an npath complexity in the millions. This code could probably be refactored into a simpler structure but at the same time the business impact of this must be considered above the theoretical reasons for making such a refactor. In most cases it might not be cost effective to take such an approach, instead the team should agree on levels of compliance and standards to which their code is to be upheld to.

Measuring the impact of a developers code must be analysed from a number of different perspectives. Changes or additions with say 100+ consecutive lines are to be considered of a lower impact as they are more than likely to have been quite easy to implement when compared to an addition that spans across multiple different files with additions and deletions. It is likely that the latter scenario took a lot more planning in development and a more thought-driven approach taking into account various crucial dependencies between files.

Computational Platforms Available

A wide range of companies and organisations in the software industry today hold their code base on repositories such as GitHub. With over 2.1 million repositories hosted, GitHub provides some useful analysis and metrics into the

development lifecycle of a project and also the team working on it. A GitHub commit can tell us a lot about a given developer and allows us to draw some particularly useful metrics from the source code they are contributing to a project.

GitHub commits can be used to give a brief overview of the output of a given developer within a team and to gather metrics such as lines of code added or deleted. These commit statistics however do not provide enough detail to draw much useful conclusions from this output with regards to how much of an impact these contributions have had towards a project. One insight these commit metrics can provide is how important a given developer may be within an organisation due to their contributions to the code base. A developer who has seen the majority of the code pass through from the beginning of a project be it from their own contributions or by reviewing pull requests will have a much larger value than a newly recruited developer, even at a senior level. If any bugs were to arise it is quite likely that this developer may be have knowledge of how to resolve the issue even it was not their own personal contribution.

Automated tools that plug straight into developers IDE's and also the repositories housing a projects code base can be quite useful for analysing metrics within an organisation and also for the purpose of quality assurance (QA). An example of such a platform is the likes of TestRail, a product which integrates directly into both GitHub which holds source code and Jira which hosts relevant development tasks and Agile development boards. TestRail does not gather source code productivity metrics such as individual productivity with regards to lines of code produced. Instead it serves the purpose of allowing teams to monitor their testing metrics, providing detailed user-friendly interfaces which allow both developers and project managers to closely monitor the testing quality, test results, code coverage and other useful metrics relating to the individual tasks they are working on. This is extremely useful from a QA perspective as it allows project managers to quickly ensure that individual tasks have been tested accordingly and that a desired percentage of code coverage has been reached. By using these metrics both project managers and developers can significantly reduce the amount of bugs that may arise within a given project thus increasing productivity in the long term by reducing the need to spend time fixing bugs on old issues that may arise down the line in the future. (Test Case Management - TestRail, n.d.)

Tools such as Jira which was developed by Atlassian are widely used throughout the software development industry today especially within organisations operating on an Agile methodology. Jira allows product managers to easily map out and design 'sprints' into tasks and assign these tasks to individual developers within a team using a scrum board. These tasks can then

be tracked throughout the lifecycle of a sprint as developers update the progress of each individual task as they proceed through their workload for a given sprint. This also gives developers within a team full visibility into what is next to come allowing for project managers to deliver maximum output in minimal cycle time. Jira also provides some in-house reports with regards to various aspects of the Agile development workflow related to each individual sprint. Product managers can gain access to useful charts and metrics such as burndown charts which shows the total work remaining and the likelihood of achieving the sprint goal. Jira also integrates directly with the likes of GitHub and Slack allowing for developers to get their code reviewed extremely quickly and reduce the overhead time spent on getting code from development to production. (Jira Software - Features, n.d.)

Algorithmic Approaches Available

As mentioned above productivity can from a vague standpoint be measured as output divided by input. However, in order to account for other factors within the life of a software developer this equation must be changed.

In the software industry, productivity has historically been measured using the equation:

$$Productivity = ESLOC / PM$$

- *ESLOC = Effective or equivalent source lines of code*
- *PM = Person Month*

ESLOC must be greater or equal to the number of source lines created or changed. Using this metric can be quite easy when all of the source code contributions are brand new. However, in a professional environment there are constantly bugs arising and code needs to be fixed on the go continuously. As a result the equation must be refactored once again to account for this. When modifications occur there are several other factors that have to be accounted for. Modification requires an understanding of the code to be changed as well as the underlying system(s) or architecture(s). If there is insufficient documentation a developer must perform *'reverse engineering'* to figure out how the code works and how it should be modified accordingly. Modifications also must respect any existing dependencies and as a result impose more difficulty on the developer. The complexity of a modification also increases if the modification must be done in a new language or has to be compiled with a new compiler or linked with a new linker etc.

To account for all of the above complexities an adaptation adjustment factor (AAF) can be applied on the modified code as follows:

$$AAF = 0.4F_{des} + 0.3F_{imp} + 0.3F_{temp}$$

- F_{des} = *% of software requiring redesign and reverse engineering*
- F_{imp} = *% of software that must be physically modified (re-coded)*
- F_{temp} = *% of software requiring regression testing*

Using this adaptation adjustment factor the previous ESLOC variable can be re-written and defined as:

$$ESLOC = = S_{new} + S_{mod} + S_{reused} * AAF$$

- S_{new} = *new lines of code*
- S_{mod} = *modified lines of code*
- S_{reused} = *reused lines of code*

While having an equation such as the one described above can be quite useful, it is not easy to count ESLOC for developers within an organisation. External tools/scripts must be used/designed in order to scrape statistics off of an organisation's code repository and calculate the resulting ESLOC metrics. However, when ESLOC metrics are used continuously over multiple different projects within an organisation they can be significantly effective in allowing a cross-comparison between the productivity of developers throughout a given development task. (Chatterjee)

Another algorithmic approach available that can be used to measure the complexity of a program and thus measure the productivity of the individual producing such program is Halstead complexity measures which were introduced by Maurice Howard Halstead in 1977 as part of his treatise on establishing an empirical science of software development.

Consider a given software problem. Let:

- n_1 = *number of distinct operators (e.g. +, =, printf, int…)*
- n_2 = *number of distinct operands (e.g. a, b, c)*
- N_1 = *total number of operators*
- N_2 = *total number of operands*

Using these numbers we can proceed to derive several measures that when combined will allow us to analyse the problem at hand from a mathematical perspective.

Using the above numbers we can define:

- *Program vocabulary:* $n = n_1 + n_2$
- *Program length:* $N = N_1 + N_2$
- *Calculated program length:* $N' = n1 \log_2 n1 + n2 \log_2 n2$
- *Volume:* $V = N \, x \log_2 n$
- *Difficulty:* $D = \frac{n1}{2} \, x \, \frac{N2}{n2}$
- *Effort:* $E = D \, x \, V$

The difficulty definition above relates to the difficulty in writing or understanding the program which accounts for a developer performing a code review and also the developer writing the code. The effort measure can be translated into actual coding time by using the following relation:

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

One can also estimate the time required to test using the following equation. Note that *k* is the stroud number which has an arbitrary default of 18. This can be adjusted to suit a team's own testing conditions based on how critical the code may be, team background, etc.

- *Time required to test:* $U = \frac{E}{k}$

Halstead also derived a delivered bugs metric which attempted to make an estimate for the number of errors/bugs that would arise from a given developers solution.

- *Number of bugs:* $B = \frac{E^{\frac{2}{3}}}{3000}$

Whilst quite revolutionary for its time and still somewhat applicable to the modern day software industry, Halstead's metrics do not hold quite as well for modern programming languages and tools which allow programs to be resolved and written much more quickly. They are still quite a useful benchmark for analysing actual metrics within projects such as time taken and bugs produced

but should be considered quite lightly and tested over multiple projects before approaching a conclusion upon their relevance to any particular team or developer. (IBM Knowledge Center, n.d.)

Methodology Approaches Available

The software industry today is home to many different software development methodologies such as Waterfall, Agile and DevOps. With each of these methodologies comes their own unique methods of measuring the developers within these teams and their rates of productivity and throughput.

Within an Agile development workplace, the product lifecycle is usually broken up into what is known as '*sprints*', these are usually two week periods where a set number of pre-defined tasks or goals are worked on and hopefully completed. Before each sprint there is also an estimation session where developers and management will meet and discuss the estimated time necessary for various tasks within the sprint. This gives both developers and managers a crucial method of measuring both the estimated time necessary for a developer to complete a task and allows the developer themselves to become aware of the rate of productivity necessary in order to complete this task within the agreed deadline.

The tasks within the sprint are then spread between individual developers in a team for them to begin working away on at their own rate. Some Agile workplaces will have daily stand-ups or meetings where they will discuss their progress through various tasks within a given sprint and any difficulties or blocks they may be facing. This also gives those attempting to measure their team of developers are direct insight into how progress is going and whether the previously discussed estimation time will be met.

Teams operating using this methodology also commonly use what is known as an Agile board. This allows for teams to track what tasks are available within the given sprint and who is working on what. It allows for priorities to be set and also for the estimated time to be included. This gives project managers a useful platform where they can gain an overview of how the current sprint is going and the rate and which tasks are being completed by each developer. Tasks on the board can also be placed into sub-categories such as 'in development', 'blocked', 'needs review', 'completed'. This is also extremely useful for project managers as it allows them to see the lifecycle of the current tasks and also to understand that a task may not have been completed yet as a developer may be waiting on a feature to be completed by another developer and thus be 'blocked'.

Within a workplace operating on a DevOps methodology there is a shift in focus from product releases in short bursts to a what is known as ‘Continuous Integration’ or ‘Continuous Deployment’. DevOps is focused on improving the time to market, lowering the failure rate and shortening the lead time between bug fixes which allow for minimal disruption to the overall product say from the perspective of a SaaS organisation.

While this is a fantastic way of going about development within a company/organisation it can be quite difficult to measure or gather metrics on the individual developers and the tasks they are completing when compared to an Agile methodology as discussed above.

A company/organisation operating with a DevOps methodology can very quickly become flooded and overwhelmed with a large number of tasks that are stuck in development and are not being pushed across the line to deployment. It is for this reason in particular that close attention must be payed to particular metrics within these workplaces.

In order for DevOps to work smoothly and effectively, project managers must ensure what is known as a single piece flow is present within the development lifecycle of any given team. This means that the max work in progress (WIP) must be no higher than one across every single phase of the of the value stream of a product. This tends not to be the case within a lot of companies/organisations which commonly have much higher WIP levels, in particular as the project/product approaches delivery.

A useful metric that is commonly used within DevOps workplaces is what’s known as ‘batch sizes’, these are the average number of backlog items at any stage of delivery. Batch sizes can be used as an indicator of release cycles that may be too long. As the time taken to release spans out, so too does the build-up of the batch size of tasks waiting to be completed. Batch sizes can be used hand-in-hand with other performance metrics to analyse and measure individual developers within a team and maybe draw a conclusion as to why the batch size is increasing at an undesirable rate.

Other useful metrics within a DevOps environment include the likes of deployment frequency, deployment time and lead time. Since the goal of DevOps is to constantly and frequently deploy code these are metrics that should be closely watched by any project manager within such an environment. Metrics must also be gathered and measured from the perspective of error prevention and error resolution. Such metrics include the likes of mean time to detection (MTTD) and mean time to recovery (MTTR), both of these metrics should be kept as low as possible to ensure that defects are caught and resolved

at an optimal rate in order to ensure the product is operating as desired and is cost effective. (Devops Performance Measurement, n.d.)

Ethics and Implications Surrounding Software Development Analytics

Whilst some of the methods discussed above can be extremely useful for an organisation with a software development branch they also bring with them some serious ethical concerns and implications.

Introducing systems or methodologies that cause developers to be under constant watch and measure from the second they walk into the office does not create a comfortable environment in which to work and also strongly hinders productivity. Instead of focusing on the task at hand, many developers may feel they are under direct pressure to meet timing and productivity level quotas. Developers may channel all of their resources into meeting these quotas and produce a poor end product that does not meet the desired standards of the project managers and customers. For this reason the project managers within an organisation must think carefully about how they want to roll out and make use of their desired measuring metrics.

Having developers under constant monitor within a workplace also introduces some ethical concerns such as a quite serious invasion of privacy. Whilst it may be useful to have an automatic quantitative software development measuring tool built in to developers IDE's to track coding metrics, organisations must be careful not to take this too far and start infringing on people's privacy. Proceeding further than this and attempting to gather metrics from a developers daily life within the workplace such as how long their breaks take, how frequently they use the bathroom, who they talk to etc. may introduce an extremely hostile and somewhat dictatorship-like environment that will serve a negative impact on the productivity of developers rather than increasing their productivity.

It is for reasons like this that software giants such as Google, Facebook and Amazon have transitioned their workplaces into enjoyable and comfortable places to work rather than a strictly enforced organisation that is solely focused on producing products at the highest rate. Giving developers a comfortable platform upon which to work will no doubt increase their productivity significantly more so than the latter option.

Conclusion

In conclusion, from the reasons discussed above I believe that the field of software development measurement and gathering metrics with regards to software development is extremely complicated as there is no one way to define the productivity of a software developer. Both languages, methodologies and tools are constantly changing within the field of software engineering and as a result of this any definitions of productivity that have been defined in the past (such as Halstead's metric) have become somewhat irrelevant in the field today. Measuring software engineering should be done on a human-to-task productivity level rather than on a source code productivity level.

I believe the methodologies of both Agile and DevOps have introduced a new and useful way for project managers to gather metrics and measure their teams productivity. These methodologies have also introduced a fast-paced environment for developers to work in which keeps productivity levels high and reduces the need for scrutinising metrics to be gathered. There still remains a need for organisations to play close attention to their QA and testing metrics especially with the large shift in modern society with a large number of companies pushing towards SaaS.

Bibliography

Chatterjee, D. (n.d.). *Report - Measuring Software Team Productivity.* Retrieved from Berkeley.edu: http://scet.berkeley.edu/

Devops Performance Measurement. (n.d.). Retrieved from https://blog.versionone.com/devops-performance-measurement/

George, B., & Williams, L. (n.d.). *An Initial Investigation of Test Driven Development in Industry.*

IBM Knowledge Center. (n.d.). Retrieved from https://www.ibm.com/support/knowledgecenter/SSSHUF_8.0.0/com.ibm.rational.testrt.studio.doc/topics/csmhalstead.htm

Jira Software - Features. (n.d.). Retrieved from Atlassian.com: https://www.atlassian.com/software/jira/features

Test Case Management - TestRail. (n.d.). Retrieved from https://www.gurock.com/testrail