Measuring Engineering

***To deliver a report that considers the ways in which the software engineering process can be measured and assesed in terms of measurable data, an overview of the computationial platforms available to perform this work, the algorithmic approachs available, and the ethics concerns surrounding this kind of analytics.***

In a competitive market, businesses and enterprises are constantly trying to bring the best product or service to the consumer at the best price, they are doing this as these businesses are competing in the market place to win over the consumer to their product, it is therefore also in the best interest of the consumer to seek the best price for the product that they seek or price to quality ratio for that product if quality is a factor that they are considering. Thus, stakeholders in enterprise are constantly seeking ways to automate and optimize elements of their production line to reduce the cost of production of their goods and services to allow themselves to compete effectively. It is this constant optimisation by the capitalist that incentivises skills and ultimately brings commodities to consumers for their best possible price.

For this reason, stakeholders in technology companies that produce software goods and services to consumers and other industries need to utilise some methods of attempting to optimise their production line. For a strictly software company, their production line is largely made up of the labour of those writing the code and programs for them. Apart from ensuring their staff are granted access to modern computing devices that inevitably reduce start up times, processing times, are overall more reliable and effective and providing the means to which the staff can all collaborate, the company also needs to ensure and enforce to some degree, the efficiency of their individual staff. In the same way that the stakeholders of a stone masonry company would be interested in how many bricks a mason can clean and lay per hour/day for a given task or what progress this an apprentice or novice is making to achieving the desired productivity, a software company is also interested in the quantity of work that their staff can produce per day. In this report, I will be detailing the ways in which the software engineering process can be measured and assessed in terms of measurable data, I will be giving an overview of the computational platforms available for this work, detailing some of the algorithmis approaches available for these measurements and finally giving my own view on the ethical concerns and responsibilities a software company should have when conducting these measurements.

While the assessment of productivity of the given example of a stone mason may be quite simple (most likely too simple) and have little variance on a weekly basis, given that they know how to correctly clean and lay the stone, how does one measure the productivity of a software developer? Does one simply assess that they can succesfully write software and simply count the lines of code they write and hope that that remains constant? Does this then leave possibility for disparity in different individuals software writing process not to be represented correctly in our simple analysis? I will now begin to detail the structure of the work performed in a software developing company and how that leads to available data that can be and is analysed by these industries of their staff to determine their staff’s productivity.

The development of software within a company begins with selecting the software process required for the service. In recent times, the “Agile” software process has seen increasing adoption to now be the more widely used process compared to the traditional “waterfall” process. ( source: <https://techbeacon.com/survey-agile-new-norm> ) The “Agile” process places emphasis on the continuous development of the software to meet changing needs and to cater for inevitable unforseen irregularities in the software design process that the traditional “waterfall” method enforces. The Agile development process allows for developers to plan and develop incrementally during the creation of the software. For measurability, this allows versions of software to be released frequently, each adding new elements of functionality, bug fixes and stability. The benefits of the Agile method over the traditional method are reducing the cost of being able to cater for changing customer needs, making it easier to get feedback with new versions able to be tested as well as generally overall a more rapid deployment of the service. One of the reasons for the Agile method to be so well adopted recently is the availability of data gathering and data analysis tools that can now almost replace project managers whose job largely consists of attempting to estimate and plan the time frame for the development process. With frequently changing consumer needs and with only a single stable product being released in a waterfall model developed product/service, the use of Agile can be used instead to measure the overall software process time by means of measuring the time taken for individual stories and sprints to be completed.

This combined with an estimate of the number of sprints and stories required for the next version of the software allows for more accurate analytics on the time taken for each version, how much functionality is added for each version and for measuring the progress of the quality of the product given by customer feedback. The traditional waterfall method releases one singular product at the very end of the production to the consumer without really getting intermediate feedback on the product, this lack of measurement of consumer’s desires and progress towards a desirable product for the consumer is definitely a contributing factor for why the waterfall method had over 3 times the failure rate found in the 2015 CHAOS report (waterfall : 29% vs agile: 9% ). (source: <https://www.infoq.com/articles/standish-chaos-2015> ).

One reason that could have seen a large shift toward the agile development process rather than the well documented waterfall process is the availability of powerful version control software. Version control software is essential to the modern software implementation process, it allows for development teams to all have access to a code base providing functionality of reverting to older versions of the code base, merging the contributions of multiple team members and many other functions. This technology being used in many tech projects has facilitated the unnecessity of large amounts of documentation that the waterfall method implemented. It also permits easily bouncing between different stages of the process without necessarily having the whole next stage of the process put on hold for all contributors, with an agile method in conjunction with version control certain teams and functionalities can backtrack to other stages of the process while not halting the whole development of the project. With commit comments and build versions as well as analytics that can be abstracted from the use of the software in calculating sprint and story times and making well calculated estimates, the old method of making estimates at design time results in better informed software estimations and measurements. This is an example of how an enterprise can use software measurement to optimize the production line by replacing large amounts of documentation with technology.

The most commonly used version control software that I mentioned in my last paragraph is git. Git facilitates a directory of files that any amount of contributors can alter and contribute to, in a company this will be the staff. The software will collect details on how many lines a contributor commited to the code base, how many commits each user made, the comments made about each commit, when the commit was made as well as lots of other pieces of data about the contributors and their individual commits and thus, contribution to the project. Employers and stakeholders also would like to use measurement methods on the individuals contributing to the project as well as measurement of the project.

One of the first instances of measuring a contributor’s data was the Personal Software Process proposed by Watts Humphrey in his book A discipline for Software Engineering. This process aimed to adapt organisational software process analytics for individual developers. The process aims to improve their estimating and planning skills, manage the quality of their work and reduce the number of defects. The personal software process uses spreadsheets, manual data calculation and manual analysis to improve one’s skills, it is therefore aptly named a process as it is focused on catering for individuals needs rather than collective needs by having individuals manually collect their own data on their own working habits and derive their own meaning from the data that the process creates. Humphrey’s commented “It would be nice to have a tool to automatically gather the PSP data. Because judgement is involved in most personal process data, no such tool exists or is likely in the near future.” As the PSP yields data that aims to aid individuals in their planning and development efforts, this process yields analytics that are “fragile”, that is any single analytic may hold different meaning for different developers as the process doesn’t aim to conform a developer to a single development method rather it aims to help developers learn about their own development habits.

As this toolkit obviously has drawbacks being used in an industry due to the small amount of significance of the data being used as a comparator between developers as well as the process not having much intrinsic automation element to it, its hard to see how this process could be used. The Collaborative Software Development Laboratory at the University of Hawaii aimed to develop this process further by adding certain automated elements to the process, such as automating subsequent data collections acting as a comparator for the progress or change a developer has made to their personal development progress. They called this toolkit the “leap toolkit”, the toolkit did create certain drawbacks such as if a developer wanted to monitor a certain element of his work ethic instead of manually gathering the data and entering it into spreadsheets proposed in the personal software process, he would now need to create an automated method for this collection. Again this toolkit gave only significant data to an individual not to a collective body.

The Collaborative Software Development Laboratory decided to develop a program that attempted to collect as much information from a developer about their working habits without having any goal for these statistics called Hackystat. It did these collections automatically without developers noticing and sent them to a server. This allowed for individual statistics to be calculated, including dervied statistics.

The research found three significant social and political problems with the project. Firstly some of the developer’s viewed the unobtrusive data collection as a bug, as they werent informed of the instrumentation being installed on the device and the nature of the data colletion. This shows that developers would rather be informed about being monitored rather than to be monitored without needing any permission. The next issue was regarding the transparency of the data and how it could expose individuals working habits. The final issue found that developers didn’t want this transparent information available to their management despite being promised the information would be used appropriately.

As the data collected by Hackystat didn’t have a high end goal other than gather as much possible data about contributors, developer’s weren’t keen on this being used as a measurement of engineering, it also doesn’t consider external real world factors such as interruptions to a developers work flow by someone entering their office. The data that Hackystat collected also had questionable relevance in relation to the project such as the program collecting information on how much time a developer spends inside his/her IDE, how many commits they make, how many builds they make etc. This information only serves to force developers to conform to a work pattern that could also in turn be counter productive as the developer may be more concerned, if this information was to be used by management with how he was working rather than the quality of the work. For example a developer might see that his co worker is making twice as much commits as him that day, and making twice as many builds. A developer may then try attempt to make twice as many commits and builds to compete and seem in line with other developers habbits. This could easily be abused and cause controversy within the project about what is an appropriate amount of commits, builds etc.

As well as this Hackystat also maintained the amount of time a developer spent in their IDE. This statistic is simply a time measurement and doesn’t maintain the amount of code a developer has written in their IDE (although this would be maintained by the commits). This type of information may also lead developers to attempt to conform to a working style that is not entirely suitable to them. This doesn’t account for any use of other tools that a developer may use and may cause them to attempt to bluff their time in the IDE as a means to compete with their colleague’s. This type of software measuring hasn’t seen to be adopted by modern industries but some of the technology and ideas has been used in better measurement tools.

A better use of the information Hackystat gathered would be to compare the information gathered about a developer and attempt to derive meaning about it in relation to the overall goal of the project, an exmaple of this would be instead of gathering information simply on how many commits a developer made and to the amount of lines in the commit, a more meaningful insight would be the code coverage of each commit and per commit attempting to improve on that. Many services that provide this information exist to this day including DevCreek, Sonar and Ohloh. These services have seen adoption in industry due to their uncontroversial nature. The drawback of using solely an automated service for measuring engineering such as the one’s proposed here is that they don’t allow a single developer to deduce what would increase his productivity, it simply maintains his contribution to the goal of the project itself. The benefit of using the PSP or Leap toolkit mentioned earlier is that they have the freedom to monitor their own habits and expose their own weaknesses.

They can then choose to deduce information from the statistics that this process and toolkit gives to them. This also allows a developer to ignore “outlying” data, even further increasing the flexibility and meaning of the information. Outlying data may include for example a day when the developer wasn’t using his normal normal device for developing due to unforseen conditions, this then has the possibility of giving false information collected by Leaps automatic collection. The developer has the freedom to choose what is relevant and meaningful and what is not, the obvious drawback of solely using this approach however is how critical a developer will be of his own working habits and how willing they will be to change them. To conclude measuring software in industry requires consideration of measuring the project itself and measuring the contributors, no single tool is likely to completely measure a developer’s productivity accurately but can provide a loose indicator of individual’s performance using a combination of many different metrics gathered by automatic gathering tools such as lines of code, code coverage, code complexity, use of comments and errors.

Another example of using technology in industry for measuring software is the use of

Bibliography

Software Engineering – Ian Sommerville.

Searching under the Streetlight for Useful Software Analytics – Philip M. Johnson 2013.

Network effects on worker productivity – Matthew J. Lindquist, Jan Sauermann, Yves Zenou.

1What data ?

-taxonomy

-code / commits

-Lines of code

-technical debt

Longitudually

Communications – email – slack

Consistency

Performance

Measurable data

2Computing

Analytics

Analytics as a service

Codeclimate

Case studies

3Algorithms

Computational intelligence

Algorithm

Machine learning

False positives from machine learning, bad trends, irrelevant data etc.

4Ethical concerns, wrap it up.