Measuring Software Engineer

Processes

Jack Gallagher | Computer Science & Business | 10/11/2017

###### Measuring and Assessing the Software Engineering Process

##### 2.1 Measurable Data

The software engineering process consists of a set of inter-related activities that transform inputs into outputs while exhausting resources to achieve the transformation

Measuring the productivity of software engineers has always been an issue within the IT industry. In other industries productivity can be simply measured, for example in the manufacturing industry by measuring the cost to produce a batch of items, divided by the number of items produced will provide an accurate measurement of productivity. However, in software engineering it is not as white. Two main factors to consider when it comes to measuring productivity are

1. Effort required to build the system (Input measure)
2. Size of the software that is to be delivered (output measure)

Measuring productivity is an important aspect of software engineering and it provides many benefits.

1. It determines if one methodology produces a faster result than another
2. It finds the most cost-effective techniques and tools
3. It establishes optimum team sizes for different projects
4. it tracks the overall cost difference between using experienced resources versus inexperienced
5. It can be used to compare your team to industry competitors

##### 2.2 Quantitative Measurement Techniques

##### 

I believe that through quantitative process measurement techniques it can make assessment and measurement of productivity accurate and reliable.

Quantitative measurement techniques are used to collect and analyze data in numerical form to which mathematical and statistical techniques can be applied. These techniques provide work product data that can be used to indicate where process improvements are needed and to assess the results of process improvement initiatives.

This collected data has many uses, however, in regard to software engineering processes it is a useful source for measurement. For example, the number of defects discovered during software testing and the staff-hours expended can be collected by direct measurement, and the productivity of defect discovery can be derived by calculating defects discovered per staff-hour. (ncsu.edu)

Process measurement analytical paradigms is characterized as relying on “**quantitative evidence** to determine where improvements are needed and whether an improvement initiative has been successful”

There are several different analytical paradigms such as…

1. **Experimental and observational studies**: This involves setting up controlled experiments in the organization to evaluate processes.
2. **Process Simulation**: Used to predict software engineering process outcomes if the process is changed in a certain way, ultimately initial data collected is used as a basis for the simulation.
3. **Orthogonal Defect Classification**: ODC is a technique that can be used to link faults found with potential causes. ODC can be used to group detected defects into categories and link the defects in each category to the software process or software processes where a group of defects originated. (ncsu.edu)
4. **Statistical Process Control**: Through the use of control charts and their interpretations, this is an effective way to identify stability or instability in the process.
5. **Personal Software Process**: PSP is a structured software development process that is intended to help software engineers better understand and improve their performance by tracking their predicted and actual development of code.

These analytical paradigms all contain measurable data and contribute towards the measurement of software engineering processes.

##### Software Metrics

There are many software metrics used to measure software engineering processes.

Software metrics are used to describe the wide range of activities concerned with measurement in software engineering. Some basic software metrics include lines of code, comment density, number of classes and interfaces, program load time, bugs per line of code and many more.

Lines of code

LOC is a measurement technique whereby productivity is measured based upon the number of lines of code in the program’s source code. There are several advantages of using LOC as a metric, it directly relates to the end product, measurement is a straightforward process and it measures software from the developer’s point of view i.e. what he actually does.

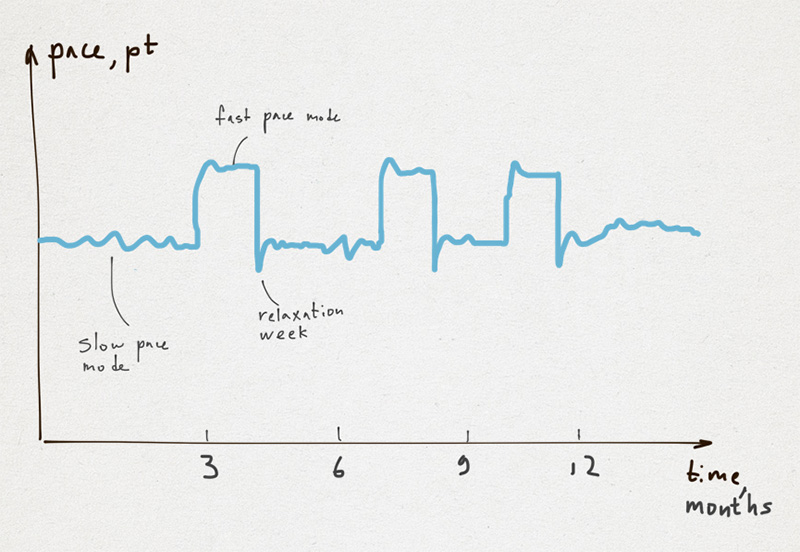
However, despite its benefits it has costly disadvantages. Programmers may be rewarded for writing more LOC based on the misconception of higher management thinking that the more LOC, means the more productive programmers were. This is not the case a lot of the time as object orientated languages have polymorphic abilities and intelligent algorithms reduce code size. If anything, measuring productivity by this metric will only cause engineers to copy and paste reams of code, discourage simplicity and make unnecessary comments.

Velocity

A very important metric to measure software engineers is speed. Velocity is the rate of software change and it is measured by deployment frequency and time from commit to deployment. Speed is essential as projects will have deadline dates and clients will need period estimates of project completion. Speed will also illustrate if a team is blocked or if they are stable and increasing. Here is a taxonomy that illustrates high, medium, and low performers based on speed.

|  |  |  |  |
| --- | --- | --- | --- |
|  | HIGH PERFROMERS | MEDIUM PERFORMERS | LOW PERFORMERS |
| DEPLOYMENT FREQUENCY | On-Demand | One a Week to Once a Month | Once every 1 to 6 Months |
| TIME FROM COMMIT TO DEPLOYMENT | < 1 Hour | 1 Week to 1 Month | 1 Month to 6 Months |

However, there are two sides to this metric. Speed can be short term or long term e.g. a sprint vs a marathon. By rushing engineers to complete projects quickly can lead to a burnout and ultimately reduce productivity making measurement much harder. Where as a long-term approach involves intervals of working at fast, moderate and slow speeds, which prove to be much more effective in the workplace.



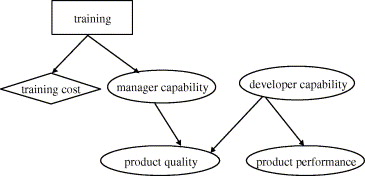
Profitability

Productivity of software engineers can be measured by analyzing the financial returns on what each team is delivering. This measure assumes that if developers are making more money for the business then they must be more productive. However, this is not a fair or effective measure of productivity. There are many factors beyond the engineers control that can affect the outcome and returns of projects and it can sometimes take extended periods of time before a company will see the financial returns from a project.

BBN’S

Despite the academic and knowledgeable success of software metrics there are some serious flaws. LOC, velocity and profitability all present issues when it comes to measuring the productivity of software engineers and despite the academic success of software metrics its implementation in the IT industry has had woeful results.

1. There is no match in content between the increased level of metrics activity in academia and industry.
2. Much academic metrics research is inherently irrelevant to industrial needs in both scope and content
3. Much industrial metrics are poorly motivated and executed.
4. LOC (Lines of Code) and other limited metrics are still widely used as a software metric despite being a very poor measuring tool.

So how can we use software metrics in measuring the software engineering process?

Bayesian belief nets (BBNs), have proven to be useful in several different projects and so much so Europe have pioneered the use of BBNs in the area of software assessment.

A BBN is a directed acyclic graph with an associated set of probability tables. Nodes in the graph represent variables and edges represent conditional dependencies.

Some benefits of using BBNs include:

1. Explicit modelling of` ignorance and uncertainty in estimates, as well as cause-effect relationships.
2. Makes explicit assumptions that were previously hidden hence adds visibility and auditability to the decision-making process.
3. Intuitive graphical format makes it easier to understand chains of complex reasoning.
4. Ability to forecast with missing data.
5. Use of `what-if?' analysis and forecasting of effects of process changes.

BBN’s provide data that can be properly measured and would be a highly effect software metric in measuring many software engineering processes including productivity.

##### 3.0 Overview of the computational platforms available to perform this work

There are several companies that analyze and measure software engineering processes. These companies provide the services to other firms to analyze their code. Some major leaders in this market include…

1. Semmle
2. Codacy
3. CodeClimate
4. GitPrime
5. Semmle uses patented technology to create a knowledge base using all available data regarding the software development process i.e. source code, issue tickets, development costs. Users can access and query this knowledge base and results are illustrated in different user interfaces designed for the unique needs of developers, managers, and executives.

Semmle enables software engineering teams to gain actionable insights from the code they create. These insights help software leaders make data-driven decisions that improve software delivery, and organizational development and efficiency. (semmle.com)

1. Codacy is a platform that aids developers in code review and quality management and monitoring. It allows firms to track project quality evolution with easy to use user interfaces, it estimates time to fix code errors and issues and also provides reviews regarding security concerns, code style violations, best practices, code

coverage and other software engineering metrics. (Codacy.com)

1. Similarily CodeClimate provides high end software analysis for firms. It focuses upon code coverage and maintainability. They provide easy github integration, week to week progress summaries as well as long term visualizations of code quality. (CodeClimate.com)
2. GitPrime is another computational platform that measures the software development process and also accommodates in the communicative gap between engineers and stakeholders. They use data from GitHub, BitBucket or any git bases code repository to aid engineers and companies alike to *“faster, optimize work patterns, and advocate for engineering with concrete data.”*

##### 4.0 Algorithmic Approaches Available

computational intelligence -Supervised + unsupervised learning algorithm -algorithm->train -> decision -false positives -opinion based -how accurate can these kind of techniques address the questions that people have about algorithms

Artificial Intelligence has progressed significantly from the early 1950 workings of Turing and McCarthy. An advantage that software engineers possess today is that they can manipulate software, to attack the challenges posed by the production of systems of software engineering projects. AI algorithms are well suited for such complex software engineering problems, because they are designed to deal with one of the most demanding challenges of all, the replication of intelligent behaviour.

##### 4.1 Computational Intelligence

Computational Intelligence (CI) provides a software engineer the unique opportunity of taking advantage of current technologies such as fuzzy sets, neural networks and evolutionary computation of genetic algorithms.

Fuzzy Sets

Fuzzy sets form a key methodology for representing and processing non-numeric information. Fuzzy sets exploit imprecision in order to make system or process complexity manageable. Fuzzy sets are often seen in requirements engineering, software costing, software quality and software risk assessment.

Evolutionary Computation

Evolutionary Computation embraces genetic algorithms. These algorithms are biologically inspired based upon the process of natural selection, evolutionary computation aims to capitalizing on the strength of natural evolution to bring up new artificial evolutionary methodologies.

Neural Networks

Neural Networks off powerful and distributed computing architecture equipped with significant learning abilities. It can be defined by 3 main components the cell-body which processes the information, the axon, which is a device enabling the signal conducting, and the synapse, which controls signals. It aids in the representation of nonlinear and multivariable relationships that can be learned from experimental data. Neural networks have been used in the development of software reliability models, self-organizing maps in software reusability, and in software quality models.

Effect

CI technologies either play or could play a significant role in Software Engineering and lead to completely new development paradigms or innovative approaches to existing methodologies. They benefit several areas of software engineering such as the role of neural networks in software reliability. Evolutionary computation aids in the development of software and its evolution rather than developing programs from scratch or a mixture of new and old code. Fuzzy sets aid in the linguistic measurement of the relationships between probabilities, effects and outcomes e.g. relationships between risk-of software-failure probabilities and fault-tolerant software development effort.

##### 4.2 Machine Learning

Machine learning explores the construction and study of algorithms that can learn from data Machine learning is the study of systems models that, based on a set of data (training data), improve their performance by experiences and by learning some specific domain knowledge. there are three basic machine learning strategies:

1. supervised learning (the learning strategy that is supervised by a teacher)
2. unsupervised learning (learning without supervision)
3. reinforcement learning (learning by interaction with the environment).

Supervised Learning

Supervised learning is using an algorithm to learn the mapping of a function from the input to the output. The goal of supervised learning is that the mapping algorithm is so effective that the output variables for the data can be predicted.

It is called supervised learning as the process of the algorithm learning from the training dataset can be compared with a teacher supervising the learning process. The answers are known, and the algorithm makes predictions on the training data and is corrected when wrong and stopped when it achieves an acceptable level of performance.

Supervised learning can be grouped into regression and classification problems.

* **Classification**: A classification problem is when the output variable is a category, such as “red” or “blue” or “disease” and “no disease”.
* **Regression**: A regression problem is when the output variable is a real value, such as “dollars” or “weight”

Unsupervised Learning

Unsupervised learning is when the machine only has input data and no corresponding output data. The goal for unsupervised learning is to model the distribution in the data in order to learn more about the data.

There are no correct answers and there is no teacher and the algorithm is left to its own devises to discover and present the interesting structure in the data.

Unsupervised learning problems can be further grouped into clustering and association problems.

* **Clustering**: A clustering problem is the discovery of the inherent groupings in the data, such as grouping customers by purchasing behavior.
* **Association**:  An association learning problem is the discovery of rules that describe substantial portions of your data, such as people that buy X also tend to buy Y.

Reinforcement Learning

Reinforcement learning is inspired by behaviourist psychology. It allows machines and software agents to automatically determine the ideal behaviour within a specific context, in order to maximize its performance.  Feedback is required for the agent to learn its behaviour, this is known as the reinforcement signal.

The reinforcement signals generated allow the machine to learn its behaviour from its environment. This behaviour can be learnt once and for all, or keep on adapting as time goes by. If the problem is modelled with care, some reinforcement learning algorithms can converge meeting the global optimum, this is the ideal behaviour that maximizes the feedback. (ref)

##### 4.3 Analysis of Data-Driven Algorithmic Decision-Making

The history of human decision making has been tainted. Too many times has it been in the hands of the wrong individuals particularly when it comes to resource allocation, fairness, justice, and other public matters it is damaged with countless examples of extreme bias, corruption, inefficient and unjust processes and outcomes. Is it time to turn to machines?

Companies and the public sector have become inundated with large scale, human and behavioral data that they have now turned to **machine learning-based algorithms** to approach complex problems due to the limits of human decision-making. Data-driven algorithms reflect a search for objectivity, evidence-based decision-making, and a better understanding of our resources and behaviors. Not only that, these algorithms are designed to analyze massive amounts of data from various sources and then, based on the criteria, select the information most relevant to their intended purpose. The ability of these algorithms to identify, select and determine information of relevance and create a solution beyond the scope of human decision-making. This establishes a new kind of decision optimization ensuring the correct decisions are made. However, it is not as transparent as it may seem with issues arising surrounding the ideas of security of personal information (computational violations of privacy.), the lack of transparency, information asymmetry and social exclusion/discrimination.

##### 5.0 Ethical Concerns

Although the measurement of software engineering processes is particularly important for firms we must consider the ethical implications involved. We must consider how the data is collected, what purpose is it for, its implications and how much should firms have access to?

##### 5.1 How data is collected

Common procedure is to collect data during test phases and perform a complete analysis of the data after project completion. However, to ensure accuracy, data collection, validation, specification and changes should occur simultaneously with development. (umd.edu)

Validation of data is a key feature of measurable data. Validity checks allow for easy correction of data, as data is consistently recorded this ultimately minimizes human error such as forgetfulness over time. Data must be carefully validated to ensure a consistent view of the data collected for all those operating and analyzing the data.

Specification is also an important feature of data. It ensures data collectors have a vivid idea of the objectives of each project. Without the specification of goals, data collected may be useless and unmeasurable.

Changes and error corrections are critical to the data. Data needs to illustrate the type of errors and what was done to rectify it and how much effort was required to do so.

Validation, specification and changes are paramount to measurable data.

##### 5.2 What is the purpose of the data

Firms will use this data to measure the work performance and effort asserted by their software engineers. If collected incorrectly, it can have profound consequences for the software engineers involved, whether it be unemployment or a missed opportunity for a promotion. I believe it is critical for firms to ensure they have accurate measuring procedures in place as it will reward hard working individuals and punish those who are less conscientious.

##### 5.3 Implications

However, are these implications unethical? I personally don’t believe so, this is how most, if not all industry workplace performances are measure, effort asserted-> performance = reward/punishment. However, I do believe, it is pertinent that accurate measuring metrics are put in place else not only is the business negatively affected but also the employees. Ultimately, I believe If you don’t measure performance you can’t make efficiencies, improvements or guarantee customer satisfaction.

##### 5.4 How much is too much?

Can too much data be analyzed? Is this the corporate climate forcing the masses to consistently grovel and work for their needs? No, I don’t believe this is true. Corporations pay engineers to perform and achieve the corporation targets, in return engineers receive a salary, bonuses and provide the opportunity to illustrate their worth to the firm. If you are performing to a satisfactory standard this should be seen as an issue else otherwise, you know you are not performing adequately.

Ultimately, the truth does not fear investigation. Engineering data produced by those employed is a free source owned by the employer and therefore they should have access to it. Issues of personal information being analyzed I agree are a problem and should never be infringed upon, however, in the workplace all personal matters should be dealt with outside office hours. Your work and personal lives should be as disconnected as possible to ensure the security of your information.

##### Conclusion – Why we must measure

Advancements in technology have increased the demand for larger, reliable and more powerful software. Likewise, the demand for software management is increasing proportionately. Software developers, maintainers, managers and technical staff alike are repeatedly confronted with innovative technologies, competitive markets, desire for experienced personnel, and demands for faster responsiveness.

Not only that but they are constantly concerned about open-ended project requirements, uncontrollable changes, insufficient testing, inadequate training, arbitrary delivery dates, insufficient funding, and issues related to product standards, such as reliability and suitability. With so many variables to be considered by software engineers, it is crucial to note the importance of measurement.

The success of all software organizations is contingent on the ability to cast predictions and commitments relative to project time scope and delivery time. Effective measurement processes help software groups succeed as they

1. Enable people to understand their capabilities, so that they can develop achievable plans for producing and delivering products and services.
2. Enable people to detect trends and to anticipate problems, ultimately providing better control of costs, reducing risks, improving quality, and ensuring that business objectives are achieved.

Ultimately, with such significant importance placed upon technology in today’s dynamic and changing business climate, it is evident to see why measuring performance in software organizations is invaluable for success.

**Measurable data**

<http://www4.ncsu.edu/~tjmenzie/cs510/pdf/SWEBOKv3.pdf>

<https://pdfs.semanticscholar.org/4cf8/e366cb36af4e1faaef903b9691d300ad63c5.pdf>

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.63.2683&rep=rep1&type=pdf>

https://www.journals.elsevier.com/journal-of-systems-and-software

<https://pages.chef.io/speed.html>

<http://swreflections.blogspot.ie/2015/01/we-cant-measure-programmer-productivity.html>

<https://www.targetprocess.com/articles/speed-in-software-development/>

<http://www.umsl.edu/~sauterv/analysis/488_f02_papers/SoftwareProductivity.html>

**Computational Platforms**

<https://semmle.com/>

<https://www.codacy.com>

<https://codeclimate.com/>

<https://www.gitprime.com/>

**Analysis of Algorithms**

<http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=614837>

<https://www.cs.ubbcluj.ro/~istvanc/master/Lecture1.pdf>

<https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/>

<http://reinforcementlearning.ai-depot.com/>

<https://arxiv.org/pdf/1612.00323.pdf>

**Ethics**

<https://www.cs.umd.edu/~basili/publications/journals/J23.pdf>

<http://repository.cmu.edu/cgi/viewcontent.cgi?article=1231&context=sei>