Measuring Software Engineering Processes
Sean McDonagh - 15319517

Report Specification - "To deliver a report that considers the ways in which the software engineering process can be measured and assessed in terms of measurable data, an overview of the computational platforms available to perform this work, the algorithmic approaches available, and the ethics concerns surrounding this kind of analytics. Reading list available here."

Introduction / Methods of measuring Software Engineering Processes

The purpose of measuring the software engineering process is to gain insight into what works and what doesn't work when carrying out, from start to finish, the creation of a piece of software. Just for clarification, I will highlight what a SWE process is. "Software engineering process often represent a networked sequence of activities, objects, transformations, and events that embody strategies for accomplishing software evolution." - Final Version to appear in, J.J. Marciniak (ed.), Encyclopaedia of Software Engineering, 2nd Edition, John Wiley and Sons, Inc, New York, December 2001.

The most interesting metrics that are worth pursuing are those that are concerned with an individual/group of SWE's performances. The metrics and measurements of the performance of an individual/group of SWE's gives a direct indication of the effectiveness of a SWE process. The purpose of measuring anything is to find its effectiveness hence making the pursuit of the effectiveness of a SWE performance a worthy task in relation to trying to measure the SWE process. Here is an example of a possible question that might be answered finding out the performance of a group of SWE's: does the Waterfall method enable software engineers to carry out tasks at a consistent frequency and hence make the most of their time and hence render the Waterfall technique a good Software Development Model. Or is it that there are too many stalling times due to Tech Leads having to develop the next massive set of tasks for engineers because they have completed their previous tasks from the past 5 months and have created a lot of bugs during the 5 months, hence rendering the Waterfall technique a negative Software Development Model due to the time that developers are left without tasks.

The question now becomes, in what ways can we measure the ongoing performance of SWE's such that we can obtain meaningful data on whether a Software Engineering process is effective or not. The following is a list of meaningful/measurable data in relation to the performance of a SWE and hence the effectiveness of a SWE process.

Code-based data sources

- One obvious way of measuring an SWE, which I have seen on multiple articles, is the amount of code an SWE has written. It gives a clear indication as to how productive an engineer has been. I do question the authenticity and ethicalness of this data in representing the **actual** performance of an SWE. My reason for questioning is the following: one SWE could have written 100 lines of high quality, effective, space efficient/time efficient code and another SWE might have written 200 lines of low quality, ineffective, space/time code that crashes that same day. Hence illustrating that lines of code is a bad metric to use when measuring the performance of a SWE. Although, it could be seen as a kind of binary metric to be used in checking whether a SWE has done any work at all.

- I think a good metric to use when measuring the performance of an SWE is that of the complexity of the code written. The Halstead's complexity metrics algorithm exists for testing a piece of code's complexity. When I say complexity, I don't mean how intricate a program might be, but rather I use the term in relation to computational speed/ amount of space used by a program. Testing a program's complexity measures a piece of code's use of space and time, and hence is a clear indication of how an SWE is performing, since space and time are two very limiting things when it comes to running a program. Hence this is data that would be of great interest.

- Other data that might be meaningful to obtain on a SWE to determine his/her performance is the number of bugs that he pushes/commits to a repository. A bug is generally viewed as something which inhibits a program in doing what it was intended to do, and hence having a programmer who is pushing a lot of code that is doing stuff that it is was not intended to do is an obvious indication of possibly poor performance and hence rich data to gather on SWE's.PSP (Personal Software Performance) software suites use the number of bugs uploaded as a piece of data that can be used to improve a SWE performance.

- Another metric that could be used is that of the time taken of a SWE to complete a given task when fully dedicated to it. If a SWE is working within a company that requires their SWE's to complete tasks that are more or less unchanging over time, then data on how long this task has taken for a given developer to finish could be collected and used for comparative purposes. Each SWE's time could be effectively measured against the times of those that have completed a similar task in the past, hence measuring SWE performance on historical performances. I am not sure if platforms exist, such as TFS/Jira, that have this kind of functionality, hence I'm not sure if this data can be obtained.

Non-code data sources

- An example of data that could be used for evaluating an SWE's performance is that of the tracking of the number of meetings that an SWE attends and the level of contribution of such an engineer. This is particularly useful data if one knew whether the SWE was/wasn't stuck on a problem/ or not. The high attendance of a SWE to meetings, and seeking advice from other SWE's on the problem that he has encountered illustrates that the SWE is taking affirmative action to solve his issue. Hence capturing the data on attendance of meetings and the related contribution of such meetings represent useful and meaningful data.

Computational Platforms used to measure Software Engineering processes

For each of the methods listed above there exists a computational platform, which can be used to gather the relevant data.

Code-based data sources analytics platforms

- Every platform like GitHub and Bitbucket, has built in analytics which will illustrate to viewers the amount of code that was committed on a given date. As I said above, I see this as a purely binary metric when analysing the performance of a SWE. Specifically with GitHub, it is possible to see all contributors to a specific repository since its inception. Platforms generally visualise such data for each contributor using a 2D graph.

- One major platform which is used by many large multinational companies to run Halstead's complexity metrics algorithm is IBM's Rational Test Realtime. The following are some of the major companies using this platform: HCL Technologies, Valeo, Delphi Automotive, UTC Aerospace Systems, Vertex Pharmaceuticals Incorporated. IBM's Rational Test Realtime also supplies functionality that calculates the McCabe Cyclomatic number in order to measure the number of linearly-independent paths through a program module i.e. Control Flow. This platform is not as widely used in the testing of code and hence companies are not desiring to test the complexity of programs, since it is only IBM and Verifysoft which supply such metrics in their testing platforms. This is purely a hypothesis as to why only IBM and Verifysoft provide such metrics on code.

  Another platform that exists for testing code complexity is Verifysoft's Testwell CMT++ and CMTJava suite's. This platform performs other analytics such as Lines of Code metrics, McCabe Cyclomatic number ( ), and Maintainability Index. The Maintainability Index is a number which indicates the likelihood of the project becoming too large/complicated to maintain with a given set of resources. This is all important data in the measurement of a SWE's performance.

  VerifySoft also provides other service that basic testing framework would also provide; such as code coverage, unit testing, functional testing, etc.

- As mentioned above, the platform to use when desiring the number of bugs that a SWE contributes when committing to a repository, is that of a PSP (Personal Software Process) suite platform. Not only will it give a performance review for each commit made to a repository, but it will also track as to whether a SWE is improving over time in their pushing of a decreasing number of bugs. Not only does it give the number of bugs, but it will also give a description of the bugs that have been encountered, and log the related error messages.

- I do not know if this functionality exists in any of the Repository servers like GitHub/ Bitbucket or TFS, the ability to group tasks that are of a similar nature and to compare the time taken for SWEs to complete such tasks and to give a performance result based on the time taken by any given SWE compared to the total average.

Non-code-based data sources analytics platforms

- Although I do not think that the technology exists to gather data on when a SWE seeks help off another SWE at a given time, there does exist some interesting technology that could lead the way forward in the capturing of such data. For example, the company Humanyze, which is an MIT spin-off from a research group looking into the development of tracking badges, have released tracking badges for industry use along with specific software. Supplying their own analytics engine, something like this could be of use in the conquest to capture data about whether developers seek the help of others, what they talked about, was their chat productive, etc. This is the kind of data that should be available via their software dashboards, as I have gathered from their website https://www.humanyze.com/case-studies/.

What Algorithms are used?

The following are the algorithms that are used on each on each of the platforms mentioned above.

- With regards to GitHub/Bitbucket and TFS, in order to compute the number of lines that each contributor would have contributed on a specific project, the Diff algorithm is used. This project calculates the number of lines contributed by a SWE to a project, by looking at the number of lines contributed in the previous commit/push by the user, and adds it to the number of lines taken away plus the number of lines added. An example of this is Paul Heckels Diff function.

- As mentioned above, the algorithm used in both the IBM's and Verifysoft's platforms to calculate the complexity of a program is that of the Halstead's complexity metrics algorithm.

- Algorithm used to find the number of bugs generated by the compilation and execution of a program is the same as compiling and executing the project from an IDE and finding the number of errors that have been logged. The algorithm that should be used in PSP to determine whether someone is getting better or not is the time taken to complete tasks and their accuracy in completing them.

- Assuming the time taken to complete certain sets of tasks by individuals exists, then the algorithm that implements the functionality of comparing the

times taken and to determine the performance of a SWE through a given time is to simply add a time to a set of times and get the average time. Then get the result of the sum ($time_{SWE}$ - $time_{AVERAGE}$). This would give the performance in the format of a negative number meaning the SWE is performing below average, above average otherwise.

- There exists many algorithms to use to evaluate the performance of a SWE and are used as intermediates in trying to attain data that would give an indication as to how a SWE is performing. Another algorithm, which would not be used by any of the platforms mentioned above, to my knowledge, and which is less widely used in industry is the use of statistical/ machine learning methods to predict whether a piece of software will fail. The following: (*Yeresime Suresh*, Lov Kumar, and Santanu Ku. Rath: Statistical and Machine Learning Methods for Software Fault Prediction Using CK Metric Suite: A Comparative Analysis*) illustrates various machine learning techniques to detect software defects.

- There exists multiple statistical models that can be used to predict software failure, such as linear regression analysis, and univariate/multivariate logistic regression analysis. There likewise exists multiple machine learning models, such as gradient descent, Levenberg Marquardt (LM) method artificial neural network.

- The following:(*Saiqa Aleem1, Luiz Fernando Capretz1 and Faheem Ahmed2, Benchmarking Machine Learning Techniques For Software Defect Detection*) illustrates that the best method to use when trying to detect bugs in programs is a SVM (Support Vector Machine) approach. It has the best ratings in terms of accuracy, and F-measure but one of the lowest ratings in MAE.

Ethics surrounding this kind of analytics

- In my opinion, this kind of analysis can be a good resource to a company if used effectively. I believe that this kind of statistics can be used to motivate people out of fear and hence have a more destructive effect than positive. Hence the motivation to make use of these available statistics needs to be analysed with discernment.

- One question that I have is as to whether it is currently ethical/moral to deduce the worth of a SWE's participation in a company to a set of statistics, when the platforms available to us cannot give insight into many aspects of a SWE's contribution in the engagement of work with others. I believe it is not. The

statistics available to us at the moment can only provide a narrow view as to how a SWE is performing/contributing as a whole. The contribution of code is only a small percentage of what SWE's as individuals contribute to a team / company.

- I believe if one looks at the statistics as to how complex a SWE's program is, then the only individuals that will exist on teams are those that don't cooperate well within a team. My reasoning for thinking this is because those who excel at programming are those who have high IQ's, and those who have IQ's have in the main lower capacity for EQ, or emotional intelligence (the following paper illustrates this: *Adrian Furnham, Relationship between Cognitive ability, Emotional Intelligence and Creativity*). Those who score low in EQ are less likely to be effective communicators (the following paper outlines this: *Hassan Jorfi, Saeid Jorfi, the impact of emotional intelligence on communication effectiveness*).

- Hence, the employment of SWEs who are exceedingly good at writing code might inhibit team performance. Yet, modern platforms available today to monitor the performance of SWEs allow managers to make shewed, one-side, and unjust decisions with regards to hiring and firing such technical specialists.

- If one wants a team to produce a successful project, (assuming that one of the tools needed to produce a successful project is effective communication), then SWE's with high EQ are needed, and hence these factors must be addressed in performance reviews. But there doesn't exist any statistics (from any of the papers that I have read) that highlight the EQ of an employee/ or his level of influence on a team, in a positive manner. This is more than likely going to be achieved by future technologies (one would hope).

- Hence, a point of view that not everything that is being contributed by an employee is going to be highlighted by the statistics available needs to be taken account of by managers etc.

How should we assess the work of SWE's/technical experts?

- When measuring the performance of SWE's, it appears to me to be essential to focus on the twin challenges of achieving excellence in technical work along with achieving excellence in people management and development. To focus solely on measures of excellence as reflected in the number of bugs committed / code complexit is disadvantageous in its unintended consequences, not least the tendency to marginalise and ignore the human and organisational aspects of performance in modern organisations.

- Essentially, what seems to be required is a balanced approach to measurement that attends to both its social and technical aspects. Factors deemed central to the social dimension include:

    - The technical expert's personal journey of development and change.
    - The technical expert's progress relative to his or her career development goals and objectives.
    - The technical expert's formal and informal influence on individuals and groups in either peer mentoring or peer coaching roles.
    - The technical expert's contribution to the effective formation and functioning of organisational teams; and
    - The technical expert's contribution to enhancing performance at individual, group, and organisation levels.

- In the absence of appropriate measures to map a technical expert's contribution towards the social dimensions of organising as outlined above, one ends up with an ever-increasing focus on the technical work system at the expense of the broader human activity system in which the work is embedded.

- The following papers illustrate the need to refocus performance measurement of SWEs / technical experts with a view to including the technical aspect of their work along with measures of individual and group performance from a social perspective.

    1. *Daniel Katz, Social Psychology Organisations*
    2. *J. McDonagh, Social and Organisational Challenges in IT-Enabled Change*
    3. *J. McDonagh, Exploit the full potential of Modern ICT Systems*

- There also exists the question as to whether the monitoring of employees, using technology of the kind that Humanyze use, is a breach of employee privacy. Humanyze's technology consists of a badge that is worn around the neck which tracks everything that is said by the employee, tracks the emotions of the employee and finally the geographical location of the employee.

- I believe that this constitutes a serious boundary transgression between employer and employee. Having this badge around an employee's neck assumes that this employee is at work for the whole 9 hours that an employee is at work. This is of course not true, as employees have lunch throughout the day and do/talk about many other things which are not related.

- This is just one example when a technology such as this can transcend boundaries and obtaining information which is of a private and personal nature. Other technologies have this capacity when used monitoring purposes.

Conclusion

- In conclusion, it is clear that the collection of measurable data, mentioned in the first section, can give a clear indication as to whether the Software Engineering Model in use is acting effectively or not.

- For example, the use of IBM's Rational Test (**platform**) to monitor code complexity (**measurable data**) of programs, using Halsteads method (**algorithm**), might give a clear indication as to whether the Software Engineering Model is effective.

- If the code complexity of programs submitted by employees is weak, then we might possibly ask the question; are our employees being given enough time to complete tasks to a high standard? If this is the question we ask as a result of subsequent statistics/data, then we are doubting the Model which guides SWEs in completing their work. Herein lies the link between measuring software engineering performance to gain insight on Software Engineering Models.

- The ethical implications of monitoring employees and representing the performance of employees solely based on **statistics** X, was also discussed. This section does not shed light on the effectiveness of a Software Engineering Model used, but gives insight as to how measurable data mentioned in the first section should be viewed as merely a fragment of a larger collection of metrics used to assess the performance of an employee / SWE.

Bibliography

1. *Daniel Katz, Social Psychology Organisations:* https://www.amazon.com/Social-Psychology-Organizations-Daniel-Katz/dp/0471023558
2. *J. McDonagh, Social and Organisational Challenges in IT-Enabled Change:* McDonagh, J., Not for the Faint Hearted: Social and Organizational Challenges in IT-Enabled Change, Organization Development Journal, 19, (1), 2001, p11 – 20

3. *J. McDonagh, Exploit the full potential of Modern ICT Systems:*
McDonagh, J., Developing the Capability of Senior Management in the United Nations System of Organisations to Exploit the Full Potential of Modern ICT Systems, Geneva, Switzerland, United Nations International Computing Centre, September, 2007, 1, 36
4. *Hassan Jorfi, Saeid Jorfi, the impact of emotional intelligence on communication effectiveness:*
http://www.academicjournals.org/journal/AJMM/article-full-text-pdf/63A098347718
5. *Adrian Furnham, Relationship between Cognitive ability, Emotional Intelligence and Creativity:* https://file.scirp.org/pdf/PSYCH_2016021415031084.pdf
6. *Saiqa Aleem1, Luiz Fernando Capretz1 and Faheem Ahmed2, Benchmarking Machine Learning Techniques For Software Defect Detection:*
https://arxiv.org/ftp/arxiv/papers/1506/1506.07563.pdf
7. *Yeresime Suresh*, Lov Kumar, and Santanu Ku. Rath: Statistical and Machine Learning Methods for Software Fault Prediction Using CK Metric Suite: A Comparative Analysis:*
https://www.google.ie/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&cad=rja&uact=8&ved=0ahUKEwiK6ZOMq_XXAhXqJMAKHZTVBBkQFggxMAE&url=http%3A%2F%2Fdownloads.hindawi.com%2Fjournals%2Fisrn.software.engineering%2Faip%2F251083.pdf&usg=AOvVaw1IYJJ5ydlSVn0Np75uWlsh

*NOTE
The bookmarks are not in any particular order.