**Measuring Engineering**

**Christopher Nixon**

**16323992**

“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.”

1. **Introduction**

Measuring the process of any kind of engineering is a natural and sensible thing to do. It is important to understand how a process works in order to try and identify the places where it could be improved or made more efficient. Software engineering is no exception, however it can be somewhat complicated to measure software engineering. The main problem is that the primary resource in software development is human resources, in particular the time and skills of developers. If one was to consider the system as a whole there are certainly inputs (developer’s time) and outputs (code produced) but the amount of factors which go into the process as a whole make it very complicated to measure. One of the key problems is that an important metric to measure is the productivity of an individual developer, or a team of developers working together and this is no small task.

One of the problems with implementing this is that any given developer uses a range of tools to effectively complete their work. These tools are all developed independently of each other and trying to implement a system which connects all of these tools and gathers information from all of these tools is very difficult. Trying to assign metrics to the interactions of developers with each other as they work together to solve a problem is very difficult, and cannot be done in a haphazard manner. The implications of assigning a ‘productivity rating’ to a developer/team of developers is that this is a measure of their value to the company. There are ethical implications to consider, as touched on later in this report.

This report consists of five sections: section 2 is an overview of the various metrics which could be used to measure software engineering, section 3 takes a look at some of the currently available platforms to gather these kinds of metrics. It should be noted that some of the metrics listed below are hypothetical, and are not necessarily measured by any currently available platform or system, but are worth consideration. Section 4 is a brief look at some of the algorithmic approaches available in computing this sort of data to provide analysis and the final section touches on the ethical considerations involved in measuring software development.

1. **Measurement of Software Engineering**

The quantity and types of data that can be gathered on the process of Software Engineering is constantly increasing and as such the list of data sources described here is by no means complete. We can however divide the data sources into three general categories; data gathered on employees, environmental factors, and other miscellaneous data points.

* 1. **Employee data**

Human resources are the most used resource in any software engineering project and as such being able to quantifiably measure the human effort needed to complete a project is important to software engineers. Data gathered on employees is either with respect to the work that the employee is producing or more personal data which indirectly might affect the work the employee produces. Naturally the intent is to maximise the productivity of employees. Data is gathered on employees to obtain a measurement of productivity, and to ascertain what factors are contribute to, or are detrimental to productivity. Since the productivity of very good programmers is an order of magnitude better than the average, it is important to determine what exactly are the qualities that a very good developer has so that the average may be improved[1].

In order to gain meaningful insights into how developers are working, a certain quantity of data is required. One of the key problems is that gathering this data manually is both time-expensive in an industry where projects are frequently late and inaccurate[2][3]. There is thus a need to gather this data automatically, which is also challenging given the variety of relevant data that might be gathered. A summary of the types of data that might be gathered is provided below.

**Code Quality**

Although the quality of work, in this case code, that a developer is producing can seem like a multi-faceted concept, it can be broken down into quantifiable areas. These include:

* Peer review feedback, which can be in a numerical format.
* Test Coverage: a percentage representing the amount of code which is covered by tests.
* Bugs introduced: how many bugs are introduced by a developers code.

**Productivity**

The productivity of a developer is not always simply a factor of the quality and quantity of work they are producing, as there are situations where the end product of a developer may not accurately reflect the value they are adding to the project/team. A developer may be spending a lot of their time working with their peers and sharing knowledge, which has been shown to be an effective technique for their peers to learn about new tools[4]. They may also be spending time designing and planning their work, which is an important process to ensure a high quality of code. Some of the areas which can indicate productivity are as follows:

* Code commits: the quantity and frequency of commits to the code base.
* Bug/Issue resolves: the quantity of bugs and issues fixed.
* Program usage: which programs have been used for what length of time throughout a given time frame.
* Communications: the amount and sentiment of digital interactions.

**Personal/Health**

This is a category which encompasses all personal and health related data sources regarding employees. Not all of these data sources are currently gathered and it is important to note that there are ethical concerns around some of these data sources which are explored in section 5. These include:

* Personal Details
* Location tracking within company grounds, potentially through a GPS-enabled badge
* Health/Heartrate: potentially measured through the use of a Fitbit or other such wearable device, as seen in Figure 4.



Figure 4.

* 1. **Environmental Factors**

There are many factors which contribute to a working environment which is contusive to productivity. The direct relationship between a working environment and the productivity is well documented, for example by a nationwide UK-based study across multiple sectors which concluded that a good office environment can increase productivity by up to 20%[8-10].

Some of the physical factors which affect an indoor environment quality are as follows[11]:

* Indoor air quality and ventilation
* Thermal comfort
* Lighting and daylighting
* Noise and acoustics
* Office layout
* Biophilia (the link between nature and humans) and views
* Look and feel
* Location and Amenities
  1. **Miscellaneous Data Sources**

Software Engineering can involve large teams working on very large projects. It is important to keep track of the progress of the project as a whole. There is no single unified approach to producing a large piece of software, but in general a project is completed in separate phases. A project might begin with a requirements phase, followed by design of the system, implementation of the design, verification/testing that the system works, and finally maintenance of the system, as in the waterfall methodology. In any such software development methodology, these phases can be measured.

Additionally the amount of developers who are working on a project can be measured, giving some indication of the scale of the project.

1. **Available Platforms**

There are some platforms available to help collect and analysis these data sources. Some of these platforms can be used for automatic collection/analysis of PSP data. The PSP(Personal Software Process) is a methodology for monitoring software development. PSP requires data points such as development time, bugs discovered/fixed and overall project size.

Mylyn Monitor[12]

This plug-in for the Eclipse IDE allows of fine-grained data on how the development environment is being used. It captures your interaction with the development environment such as mouse clicks and commands entered, and stores these in a log format. Artefacts such as files, types and methods get assigned a degree-of-interest based on how recently and frequently you interact with them. Mylyn uses this to create a streamlined workspace, making what it determines as the most useful objects the most accessible. It could be used in a different context however, to perhaps determine how efficiently a developer is using the IDE. An example of the use of Mylyn Monitor is shown in Figure 5.

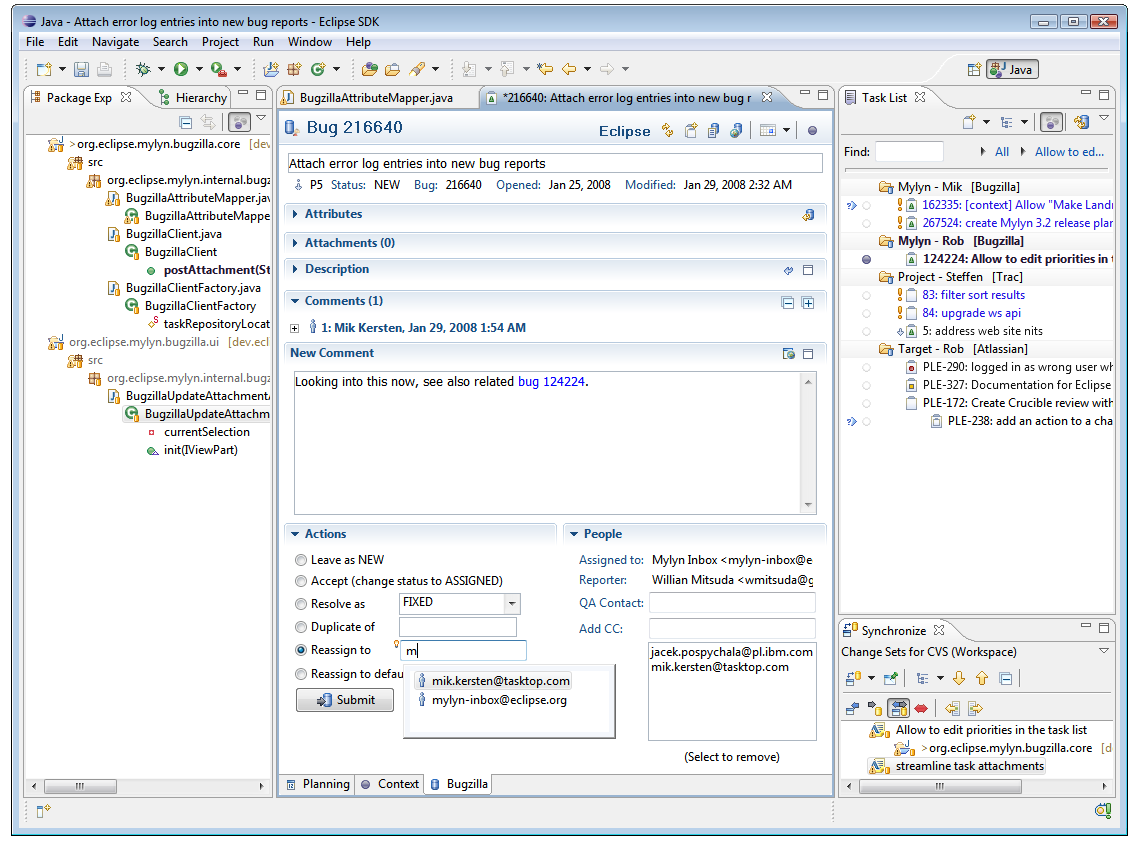


Figure 5.

Hackystat[13]

According to the website, “Hackystat is an open source framework for collection, analysis, visualization, interpretation, annotation, and dissemination of software development process and product data.” Hackystat can be used to automatically collect and analyse PSP data[6]. This automatic collection of data is done by attaching sensors to development tools. These sensors monitor the usage of the development tools and send this information back to a sensor-base, a collection of all the information from all the sensors. The ability of this software to automatically collect information is very important, as previously stated automatic collection of this information is much more effective than manually collecting it. The interpretation and visualisation of the raw collected data is another very useful aspect of this platform. An example of some of this visualisation is provided in figure 1.

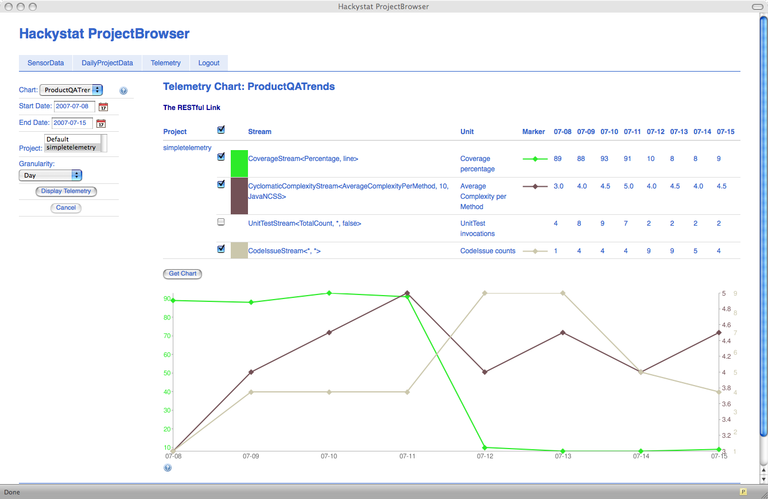


Figure 1.

PROM(PRO Metrics)[5]

This is a tool for automatically collecting and analysing software metrics and PSP data. PROM collects and analysis data at varying degrees of granularity: personal, workspace and enterprise. This allows for the preservation of individual developer privacy: a developer may access only their own data, and a manager only has access to the same data in an aggregated format. Developers can use PROM to discover their inefficiencies and improve their approach to software development. PROM uses plug-ins to development environments to record PSP data from users which is then collected in a plug-in server. The architecture can be seen in Figure 2.

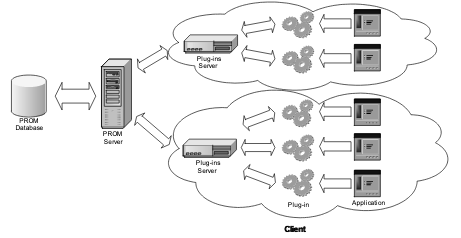


Figure 2.

1. **Algorithmic Approaches**

One approach is a ‘Rate Process Performance algorithm’. The structure of the algorithm can be seen in pseudocode in Figure 3.



Figure 3.

This algorithm rates developer events against base practises, it is quite conceptual in this form as it is designed to be applicable to various assessment practises, processes and base practises. It uses various customisable scales which can be specific for each process or event, this allows for highly granular control over the automatic ratings which it provides. Crucially, however, the overall rating is a result of both an automatic rating and a manual rating. Having a manual rating of a developers process built in as part of the system is important for both accuracy and ethical reasons, as will be further discussed in section 5.

Machine learning is another potential algorithmic approach to measuring software engineering. Since there are so many different factors that contribute to a single developer or a team of developers being productive, it is very hard to correctly weight and take into account every factor. A neural network can be very effective at taking different data points that have obtuse relationships with each other and producing a system that determines these relationships.

Neural networks can be used to first determine the coding time of any given code base change within a version control system such as git using a neural hidden Markov model. A different type of neural network, a deep mixture density network can then be used to, given a piece of code, produce the standard time it would take a developer to produce it[15]. This is an example using just the code a developer produces and the time it takes a developer to produce it. These kind of techniques could be extended to be used on the very large data sets which would be produced from more granular data on how a developer is working, such as that which would be produced by a platform such as Hackystat.

There are obstacles to overcome when using neural networks, they require very stringent validation to ensure they are not producing false results. It would be extremely complicated to try and set up a system of neural networks to take into account even all those measurements of software development listed in this report, and as previously stated there is only a sub-set of all possible measurements listed here. Additionally these types of systems do not guarantee a high degree of accuracy on individual cases, they work best when representing large sets of data.

1. **Ethical Concerns**

When measuring the performance of individual developers there are certain ethical considerations to take into account. As previously discussed, due to the nature of working as a software engineer, there is a very large amount of data that can be gathered on the work that an engineer is doing.

As an employee a developer has a right to know what information is being collected on them and to consent to the collection of this information. This can get complicated very quickly: what to do in the situation where a large number of employees do not want to consent to any information being collected on them? This would skew the results of any measurements of the rest of the workforce and the teams/company as a whole and may even result in the collection of any measurements at all becoming pointless. Another complication would be if an employee is happy to consent to certain types of information being collected, but not others. Consider the situation where an employee is consenting to being monitored as to which programs he is using throughout the day, but does not consent to the use of more granular information being collected about the manner in which he is using these programs. Any of the current platforms which were previously described are already complicated systems, and trying to manage the collection of different variations of data from different employees adds another layer of complication.

One also has to consideration the ethical considerations of how this technology could develop and possibly be used in the future. Consider the hypothetical situation where we could gather large amounts of data on every employee and use some sort of system to accurately rate the productivity of employees. Would developers be comfortable with this system automatically, and with no manual input from management recommend actions to be taken to improve the productivity of an employee? Perhaps the system could recognise that a particular employee Bob is less productive in the hour and a half immediately following his lunch break whenever he goes for a lunch break and sits with Alice (Image recognition and CCTV footage could be leveraged to recognise the social clusters within an office, in this scenario). This system could then recommend that Bob no longer sit beside Alice during his lunchbreak.

Completely removing the human, or manual element from a system which could have such a large effect on the working lives of developers could potentially be unwise and unethical. I found It interesting that in the first algorithm proposed under Section 4, there was a manual rating for every event as well as an automatic one.

Additionally, there have to be considerations about what types of data are ethical to collect. While most developers might be comfortable with the frequency of their commits to a version control system being measured, probably less would be comfortable with tone analysis being performed on all of their company communications. The use of personal information, particularly now within the EU due to GDPR regulations has to be closed monitored to stay within legal boundaries, never mind ethical ones. If information of a more sensitive or personal nature was being collected on employees (for example tone analysis being performed on all of their conversations using a recording device built into their key-pass) then there would have to be careful considerations that this information was only being gathered on company time and grounds. If an employee took their key-pass home with them and left it on their kitchen table, it would be completely unethical for it to continue recording and analysing the tone of conversations between the employee and the employee’s family.

The security that this information would have to be protected with is another important matter. Given the sensitive nature of a some of this data, it could be extremely damaging to employees if certain information was leaked. There needs to be a certain level of protection to the information kept on employees, and as the amount of information for each employee gets larger and larger, protecting such volumes of data can be a challenge. This is why there should be a policy for employers to gather only that data which is necessary on employees, instead of needlessly gathering huge amounts of data that they cannot use, and more importantly cannot protect. If and when a security breach does occur and the personal information of employees is compromised, then the employees have the right to know exactly what information was compromised, so that they can take measures to protect themselves. This can be difficult to manage when it involves large amounts of information and/or employees, which is another reason that security is not to be taken lightly.

**References**

S. McConnell, Rapid Development: Timing Wild Software Schedules, Microsoft Press, 1996.

2 A.M. Disney, P.M. Johnson, “Investigating Data Quality Problems in the PSP”, Sixth International Symposium on the Foundations of Software Engineering (SIGSOFT'98), Orlando, FL, USA, November 1998.

3 P.M. Johnson, A.M. Disney, “A critical analysis of PSP data quality: Results from a case study”, Journal of Empirical Software Engineering, December 1999.

4 . Murphy-Hill and G. C. Murphy, “Peer interaction effectively, yet in-frequently, enables programmers to discover new tools,” inProceedingsof the ACM 2011 Conference on Computer supported cooperative work,ser. CSCW ’11. ACM, 2011, pp. 405–414.

5 W. Snipes, V. Augustine, A. R. Nair and E. Murphy-Hill, "Towards recognizing and rewarding efficient developer work patterns," 2013 35th International Conference on Software Engineering (ICSE), San Francisco, CA, 2013, pp. 1277-1280.

6 A. Sillitti, A. Janes, G. Succi, and T. Vernazza, “Collecting, integrating and analyzing software metrics and personal software process data,” in Proceedings of the 29th Euromicro Conference. IEEE, 2003, pp. 336– 342

7 A. Sillitti, A. Janes, G. Succi, and T. Vernazza, “Collecting, integrating and analyzing software metrics and personal software process data,” in Proceedings of the 29th Euromicro Conference. IEEE, 2003, pp. 336– 342

8 D. Clements-Croome, Creative and productive workplaces: a review, Intell.Build. Int. (2015) 1e20

9 G. Wheeler, A. Almeida, These four walls: the real British office, CreatingProd. Workplace 357 (2006)

10 G. Wheeler, A. Almeida, D. Clements-Croome, Creating the ProductiveWorkplace, 2006

11 Al Horr, Yousef, et al. “Occupant Productivity and Office Indoor Environment Quality: A Review of the Literature.” *Building and Environment*, vol. 105, Aug. 2016, pp. 369–389. *EBSCOhost*, doi:10.1016/j.buildenv.2016.06.001.

12 <https://www.eclipse.org/mylyn/>

13 <https://hackystat.github.io/>

14 Grambow, G., Oberhauser, R. and Reichert, M. (2013) Automated Software Engineering Process Assessment: Supporting Diverse Models using an Ontology. Int'l Journal on Advances in Software, 6 (1 & 2). pp. 213-224.

15 Hélie, Jean, Ian Wright, and Albert Ziegler. "Measuring software development productivity: A machine learning approach." *Proceedings of the Conference on Machine Learning for Programming Workshop, affiliated with FLoC*. Vol. 18. 2018.