Measuring software engineering

Intro (250)

Draft 1: 2537

The ways in which the software engineering process can be measured and assessed in terms of measurable data (2000, ~250 each)

**Personal Software Process**

Key phrases: Can lead to rich, high impact analytics

There are actually many variations of PSP but they include manual data collection, manual analysis, using spreadsheets

Personal Software Process or PSP was developed by Watts Humphrey. It is intended to help management or software engineers themselves to document their software development process and use that data help them understand what changes to make in order to write software more efficiently by giving them a set of guidelines to follow. A large part of PSP relies on the software engineer trying to predict how long parts of a project will take them and then comparing their prediction to actual real world data. The main objectives of PSP are for the software engineer to improve their planning and estimating skills, to make commitments they can keep and to manage the quality of their projects.

The Personal Software Process relies on the software engineer laboriously taking note of every change they made. Before the time of IDEs, this included syntax errors when proposed originally by Humphrey. It is very free form as the software engineer is free to document as much as they want as they themselves will later be processing the data they take. If the software engineer has a theory that some external event may have an effect on their productivity, they are free to take note of how that event actually impacts their work and then analyse the data to verify or refute their theory. Although the Personal Software Process is very freeform and open it is also very time consuming, as everything has to be recorded by hand and then that data also has to be processed by hand. On top of that, people hate having free choice, they love options. Having complete freedom is very difficult as the engineer must actively think of the best way to record data and how to process that data. It is a more difficult than just filling out a premade form and having a pre made algorithm then analyse the data as someone else who made it most likely had more time to put more thought into the algorithm which would most likely do a better job of analysing the data.

Also, NOT USEFUL TO MANAGEMENT if management wants to track their employees’ performance rather than the engineer using it for their own personal improvement, this brings up another question. All the data is submitted by the employee who might have an agenda to tamper with the data for a raise or another reason, ie reduce the amount of times they make syntax errors or make it look like an algorithm they wrote was more difficult than it was and just slack off idk.

Mainly PSP was found to be quite lumbersome for general performance tracking and is only used by software engineers themselves in edge case when they want to look at something that can’t really be tracked automatically as psp has a lot of overhead for very little gain, basically the human does a job that a computer is actually really good at doing by itself.

W. S. Humphrey, The Personal Software Process, 2000.

**Hackystat Development Process Watchers**

Hackystat has been in active development from 2001 to about September 2013. Has support for emacs, vim and many other environments(?). It has been a key change/pillar in the measuring the software engineering process. The main idea behind it is for the users to attach ‘sensors’ in their software tools and then have those sensor automatically and unobtrusively track the software development process.

<https://doi.ieeecomputersociety.org/cms/Computer.org/dl/mags/so/2013/04/figures/mso20130400573.gif>

<http://141.44.17.27/cms/images/stories/media/dumke/hacky07.jpg>

The most frustrating part about PSP for most software engineers was having to constantly interrupt their flow to manually document some change, then go back to their work. Hackystat watches every change, for switching files in the editor, to writing a new method, to making a single character change. It also focuses deeply on testing, making sure each method written has a test with good code coverage. It has an emphasis on test driven development.

Hackystat can also track group projects. Analysing who made what change, who corrected it etc. Hackystat can measure if the users builds early and often a pillar of agile development. Using that info it can deduce a lot, how well the developer tests, how often they build, how long they spend in their ide, how often they commit and many other stats. All of that information is displayed in a nice table for the management where they can easily compare each team members’ performance at one single glance. The management can also if they are so inclined see exactly what changes at what time and how long it took them were made by the developer themselves documented to the second, eg. moved class at 12:43:05, added method 12:45:21 and also see exactly the text contents of that method, typos and all. All this information is collected completely autonomously without any interaction from the developer.

http://csdl.ics.hawaii.edu/research/hackystat/

https://github.com/hackystat

**Mining software repositories**

Open Hub, Cast, Synopsys, Sonar

These programs focus on analysing the rich data available on the development process from version control repositories and the like. Each tool has its strengts for inctance open hub focuses on…, They look at the already available data. This means that the software engineers do not need to set anything up at all. They just write their software, test, commit and do not have to think about data gathering at all. Then, when the software is finished, or at any point in the process, they can open up Sonar and get insights into their development process including software security, software quality, testing, general data about how much is written a day, lines of code, languages used and lot and lots of other info.

Very unlike PSP, the developer does not need to take any time out of their work to document anything. Their work is never interrupted. There is no configuration for the developer. There is also nothing is running in the background, documenting everything they type. They do not need to plan anything, it takes no processing power. Whenever they want, be it every day, at the very end of the software process or never they, are free to run Sonar and have it analyse their version control repository and give them insights about their development so far. This is by far less intrusive than PSP, but the only difference is that PSP is highly customisable to document anything under the sun where as sonar etc looks for specific data.

There is actually quite a large community of developers who are greatly interested in this topic. There are plenty of competitions that focus on extracting as much interesting useable data as possible out of open source software repositories or repositories that agree to get their software mined for the competition. In 2018 the 15th Mining Software Repositories Conference will be held in Gothenburg, Sweden.

https://www.openhub.net/

http://www.castsoftware.com/

https://www.synopsys.com/software-integrity.html

https://2018.msrconf.org/

**Using information already available on employees**

We are now able to and then rate the code not only by how well the tests run but also by the quality of the people who write it. A lot of information about the software engineering process can be extracted by analysing the code, but with the advent of AI other algorithms we are starting to be able to analyse real world data a lot more accurately. A lot of other factors such as the employee’s personality, social circles and social skills, past experiences and their dev team’s chemistry can help with predicting or evaluating the quality of software systems. Employers already have a lot of information on their employees. Data like work hours, past experience, education, training, earnings, holidays, cultural background etc can be used to find trends and try to model how good of a software engineer a person is. This data can be combined with already available data from MSR tools or tools like hackystat or PSP to draw correlations between how the employee performs as an engineer and their metadata. If the company has large number of employees a pretty accurate model can be drawn up, allowing the company to make statitstically valid assumptions about how good an engineer is just from reading their CV, before they even write one line of code for their company. This could be used as a sort of gatekeeping mechanism for the company to higher only the best candidates or allow the company give a candidate a score and calculate the candidates worth using Worth = Score/Starting Salary or a more sophisticated algorithm to maximise profits and hire only the top team members. Let’s call this score Employee’s Coding Potential Score for the purpose of exploring this subject.

The company could also ask their employees to take personality tests and compare the results of those tests to the employee’s performance on hackystat or the like to try and find trends between personality and the quality of the code they write. With enough data, the company would be able to statistically deduce what employees are good together in a team, which teams to shuffle around, which employees have more potential in higher up positions in the company and which employees they could think about letting go.

**Monitor employee behaviour while on their computer**

An employee’s Internet usage patterns and history can be used to try to estimate how engaged or focused the user was while writing a certain piece of code. Browser data would have to be processed and compared with data from MSR or Development Process Watchers. If the employee was checking social media or browsing non code related websites while they were writing an algorithm it could indicate that the employee may have been distracted and the algorithm they wrote may be sub-par (Maybe called health of the software? See next paragraph). These tools have been available for a long time with the most popular being Browse Reporter and

One step above that would be tracking exactly what the employees do on their computers. A popular tool for that ActivTrak keeps a detailed log of everything an employee does on their computer and also take regular screenshots. The software can deduce at what times the employee was working and at what times they were doing something else on their computer and at what times the computer was left on but nobody was at it. This same as with the browsing history can be couple with MSR or Development Process Watcher data to provide an even more accurate estimation of a piece of software’s health of how tuned in the employee was when writing it.

ActivTrak may not be 100% percent accurate. Maybe it looked like the employee was dozing off, but in fact they were just very thoroughly reading a lot of text on the screen, but because there was no mouse movement or key input the software thought the employee left the office but kept the computer on. Or the employee knows that this software is not perfect, so all they have to do is keep two windows open say twitter on the left and documentation for some api the employee must get familiar with on the right. All they have to do for their code to be flagged as healthy is scroll around a lot to simulate reading on the documentation window and do quick scrolls on the twitter window twice as much on the documentation window than they do on twitter, ActivTrak would record that half hour or so as if the employee spent 95% of their time reading documentation and mark their code as healthy as they were tuned in while writing it. Well, Tobii along with other eye tracking companies have an answer for that. The most accurate way to tell what the employee is doing on their computer is to know what part of the screen they are looking at. Eye tracking can tell if a person is reading a line of text or staring at the same spot while daydreaming. Although no such software exists yet it is not far off, maybe not to estimate code health yet but to replace ActivTrak definitely. Eye tracking coupled with personality/portfolio based code quality estimation from the previous section and coupled MSR or Development Process Watcher data can tell with a pretty high accuracy what percentage of percentage of an employee’s full potential the lines of code were written at. Just to illustrate my point say we have an employee named John. Their code potential score is 53. While writing a the method averageSales(int allSales[]) according to this software he was 87% tuned in. The method is then rated with a score of 46.1 (53 \* 0.87).

http://www.browsereporter.com/

<https://activtrak.com/>

https://www.tobii.com/

**Monitor employee social interactions**

With the help of AI, employers can monitor social channels such as email exchange and Slack messages between team members. Tools such as StatusToday passively monitor the communication between colleagues and display very insightful information for the management in one place. This ranges from social graphs and activeness on the social channel to how influential the employee is in their team, how critical they are in making their team run smoothly and predictions about how likely an employee is to make a mistake that will cost the company.

According to StatusToday, 84% of managers don’t know how to accurately measure the engagement and productivity of their team members. Using StatusToday data in combination with the employee’s Software Engineering portfolio from a MSR tool or something along those lines can give management very deep insights about office politics and who writes the best code and be able to rearrange hierarchies to maximise profits. It would allow management to clearly see who can be let go without making an impact on the team’s productivity. It also compares employees against each other to show who is the most critical to the smooth running of the team, who is the least and most influential in the team.

In conjunction with tools like StatusToday the employee’s physical activities can be monitored to analyse their engagement with their work, their office social circles and how all that relates to their software engineering performance. Lighthouse and Google’s nest already offer camera with human face and pet recognition. Speech recognition is also very accurate, in May 2017 Google announced that they have reached a word error rate of 4.9% in their speech recognition software. It is only matter of time before that is an option. Cameras set up at strategic places around the office could identify the employees and monitor their behaviours such as writing code, taking a break etc. On top of that they could identify social circles within the company by tracking which employees interact with whom. Transcripts of their in person conversations could then be analysed with tools similar to StatusToday to provide similar metrics such as influencers in the team, critical team members etc. This data could be paired with their code profile to deduce trends on what type of employee writes what kind of code best. The people who work together the most synergetically could then be teamed off and each team would be assigned a team synergy score, which could be calculated using their compatability and each member’s Employee’s Coding Potential Score. Each product the team writes could then be scored using their Engagement Percentage while writing each line multiplied by the Employee’s Coding Potential Score multiplied by the Team Synergy score. This would allow for a written product to be rated not only by how well the tests run but also by the quality of the team that wrote it.

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

https://www.light.house/

https://store.nest.com/

<https://venturebeat.com/2017/05/17/googles-speech-recognition-technology-now-has-a-4-9-word-error-rate/>

The ethics concerns surrounding this kind of analytics (500)

[Graph of ethics and automated-ness]

Google already has a detailed log of where I’ve been over the past two years which it sells to advertisers – if something is free, you are the product.

PSP is of least concern as all information is submitted completely voluntarily.

Hackystat, more like hacky-stalk! User are unhappy that every change they make is being watched and that info is being sent off to a server somewhere. The way it analyses data is also bad. It singles out software writing styles? Making some users seem inferior to others. Developers are often uncomfortable with management access to such data. Many developers believe that as long as the job is done well it doesn’t matter what their writing style is, but management may want to enforce it.

Sonar etc: Second, the data is generally uncontroversial; it focuses on product characteristics, not the developer behaviors that produced them

Mining software repositories

These do not focus on individual developers but instead on gathering other metrics from the data generated during the development process, unlike hackystat that focuses on developers and the team.

StatusToday this information can be quite quickly understood just from getting to know your team member and having a feeling about their personality etc but having it graphed and compared against other people is another thing.

It may as well be renamed to Unethical Hypothetical Author’s Analyser, ayooo

An overview of the computational platforms available to perform this work (Table)

Outro (250)