Measuring software engineering

Intro (250)

Draft 1: 2537

Draft 2: 3154

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

**Personal Software Process**

Personal Software Process or PSP, developed by Watts Humphrey, is a set of guidelines intended to help software engineers document their software development process and use that data to help them understand what changes to make in order to write software more efficiently. There are many variations of PSP but almost all place a large emphasis on the software engineer trying to predict how long parts of a future project will take them and then comparing their prediction to collected data after the project’s completion. The main objectives of PSP, as outlined by Humphrey, are for the software engineer to improve their planning and estimating skills, to make commitments they can keep and to help manage the quality of their projects.

The Personal Software Process relies on the software engineer consistently taking note of every step of their development process. The original PSP, when proposed by Humphrey, was developed before IDEs or always-running syntax checkers in code editors, so this included even documenting all syntax errors. Although there is a minimum of required data, the developer is free to document as much extra data as they please as they themselves will later be processing the data they take using software they write or spreadsheets. 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 that event, and when they have enough data collected, analyse it to verify or refute their theory. The problem many software engineers had with PSP is that it is very time consuming as everything has to be recorded manually and the collected data had to be processed manually. It is a more mentally taxing and time consuming task than the alternative of just filling out a premade form or having the data automatically collected.

The data collected using PSP is only useful to the software engineer collecting it if they plan to use the analysis for self-improvement. PSP is not very useful to management. Management cannot use the data to gauge the employee’s performance as all of the data is submitted by the employee who might have an agenda to tamper with the data in order to receive a raise or any other reason. Two trivial examples would be the employee reducing the amount syntax errors they made to make themselves look more mindful or recording that it took them longer to write some code than it actually did and then using that undocumented time to check their social media.

Mainly PSP was found to be too burdensome, having way too much overhead for general performance tracking and since is only really used by software engineers in edge cases when they want to analyse something that can’t really be tracked automatically. In most other cases PSP is seen as a person whose whole job is to make computers do work efficiently deciding to spend a lot of time doing a job by hand that that the computer is really good at doing by itself more efficiently.

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

**Development Process Watchers**

Hackystat has been in active development from 2001 to about September 2013. The main idea behind it is for the users to attach ‘sensors’ in their software development tools and then have those sensors automatically and unobtrusively track their software development process. Hackystat supports over ten development tools including Emacs and Vim.

Comparing team members in Hackystat. (Image may be found [here](https://doi.ieeecomputersociety.org/cms/Computer.org/dl/mags/so/2013/04/figures/mso20130400573.gif))

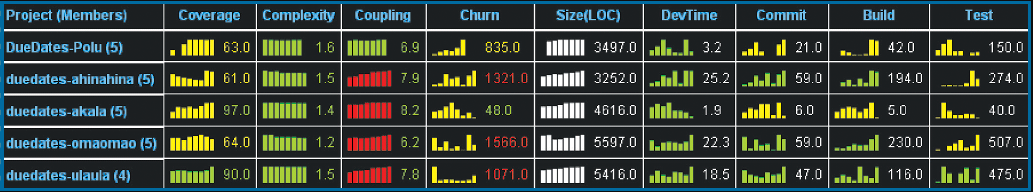
Hackystat documents every change made. (Image may be found [here](http://csdl.ics.hawaii.edu/wp-content/uploads/2012/10/zorro.png))

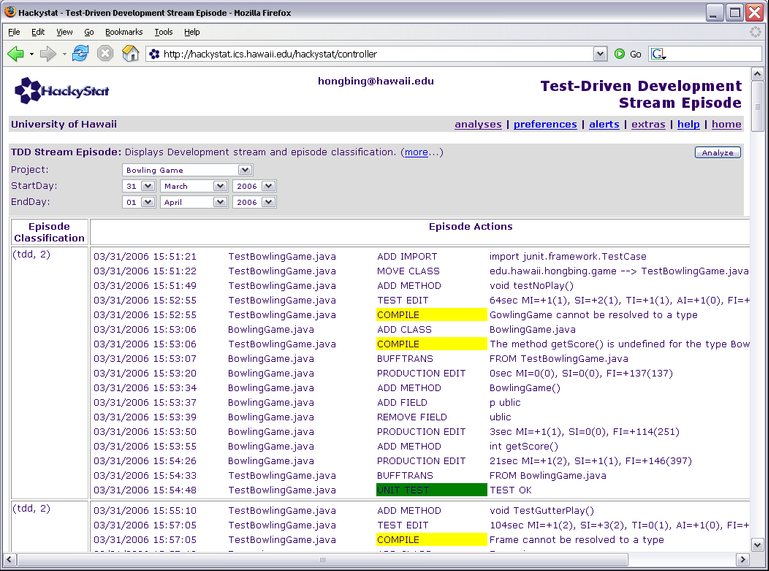
The most frustrating part about PSP for most software engineers was the cycle of having to constantly interrupt their flow to document some change, then go back to their work. Hackystat watches every change automatically, from switching files in the editor, to writing a new method, to making a single character change. From that it also calculates stats like testing, commit frequency, development time and churn.

Hackystat was made with group projects in mind, analysing who made what change and who corrected it among countless other things. All of that information is displayed in a table for the management where they can easily compare each team members’ performance at a single glance. The management can also, if they are so inclined, see exactly what changes were made at what time and how long it took the developer to make them. All of 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**

These tools focus on analysing the rich data available on the development process from version control repositories, issue tracking systems and the like. Each tool has its strengths, for instance Open Hub focuses on analysing data from open source project to produce metrics such as these:



Both Cast and Synopsys analyse software for similar stats to Hackystat as well as checking for security risk, graphing project structure among many other things.

The difference between MSRs and Development Process Watchers is that MSRs work with already available data. This means that the software engineer does not need to set anything up at all. They just write their software, test, commit and continue with their day. When they choose to see some metrics, be it when the software is finished or at any other point in the process, they can open up an MSR and get insights into their development process including software security, testing, general data about how much is written a day, languages used and countless other metrics depending on the MSR of their choice.

There is 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 from companies that agree to get their repositories mined for the competition. The community has also held 14 annual MSR Conferences so far. In 2018 the 15th Mining Software Repositories Conference will be held in Gothenburg, Sweden.

Open Hub, [openhub.net](https://www.openhub.net/)

Cast, [castsoftware.com](http://www.castsoftware.com/)

Synopsys, [synopsys.com/software-integrity.html](https://www.synopsys.com/software-integrity.html)

MSR Conference, [2018.msrconf.org](https://2018.msrconf.org/)

**Using information already available on employees**

It is certain that a lot of useful data about the software engineering process can be extracted by analysing the code, but that is only a part of how the software engineering process can be measured. With the advent of powerful AI we are starting to be able to analyse messy real world data a lot more accurately. Many factors such as the employee’s personality, education, past experience, social skills and their dev team’s chemistry can help with predicting or evaluating the quality of software systems. In other words we can not only deduce the quality of a piece of software by analysing the code itself but also by analysing the people who wrote it.

The first step would be to analyse as much of each software engineer’s code as possible using an MSR or some other in-house tool. Each software engineer would then be assigned a score based on how good their code is. The higher the score, the better the software engineer’s code. Let’s call this score the Employee’s Code Performance Score (ECPS) for the purpose of exploring this subject.

Employers already hold a lot of information on their employees. Combining each software engineer’s ECPS with data like past experience, education, past training, earnings, work hours, holidays, cultural background etc. could then be used to find trends between an employee’s personal data and how well they perform as an engineer. If the company has a large number of employees quite an accurate model can be created in this way.

This would then allow the company to make statistically valid assumptions about how good a potential future hire is just from reading their CV, before even inviting the job seeker for an interview. The candidate’s expected ECPS could be calculated using their personal details. This expected ECPS may then be used by the company to invite only the highest quality candidates.

Alternatively, a company looking to maximise profits but to also hire candidates of a good enough quality could use each candidate’s expected ECPS and calculate the candidate’s ‘Bad Deal Score’ using something like:

B = Bad Deal Score

S = Estimated Starting Salary

E = ECPS

B = S/E

From the example below, Alice has the highest expected ECPS and the highest starting salary. If the company is looking to hire only the best candidates they would hire her. If the company is looking to maximise profits, they would look at each candidate’s Bad Deal Score instead of their expected ECPS and see that hiring David is the best choice, followed by Alice and that hiring Bob is a very bad deal.

|  |  |  |  |
| --- | --- | --- | --- |
| Candidate’s Name | Expected ECPS | Estimated Starting Salary | Bad Deal Score |
| Alice | 85 | 45,000 | 529 |
| Bob | 12 | 27,500 | 2292 |
| Ciara | 50 | 30,000 | 600 |
| David | 75 | 30,000 | 400 |

This method may also be used by management, depending to their goals (e.g. keep only the best, maximise profits) to help them decide which current employees have more potential in higher up positions or which employees they can think about letting go based on their ECPS or Bad Deal Score.

**Monitor employee behaviour**

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 an MSR or a Development Process Watcher. 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 likely be sub-par. These tools have been available for a long time with the most popular being Browse Reporter.

One step above that would be tracking everything the employees do on their computers. A popular tool for that, ActivTrak, keeps a detailed log of everything an employee does on their computer while 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 using it. This, similar to data from Browse Reporter can be coupled with MSR or Development Process Watcher data to provide an even more accurate estimation of a piece of software’s concentration value, i.e. how tuned in the employee was when writing that piece of software.

ActivTrak is not 100% accurate for the purpose of determining a piece of code’s concentration value, though. For example, all an employee must do to have their code marked with a high concentration value is make it look like they were focused at the time of writing that code. Code is only marked as low if ActivTrak notices the user interacting with non-work related programs around the time of writing that code. ActivTrak is quite easy to fool in this case, all the employee would have to do is keep two windows open, say a YouTube video on the left and documentation for some API on the right. All that needs to be done in order for their code to be flagged with a high concentration value is scroll around a lot to simulate reading on the documentation window while in reality watching YouTube. ActivTrak would record that time as if the employee spent 99% of their time reading documentation and in turn the code written would be marked with a high concentration value.

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 hardware can tell exactly that. It can even tell if a person is reading a line of code while writing it, reading documentation, watching a YouTube video or staring at the same spot while daydreaming. This way it can be known for certain whether the person was focused while writing a piece of code and thus that piece of code can be assigned an accurate concentration value.

Lack of sleep has a large impact on work performance. There is a lot to calculating how well a person has slept so I won’t get into it for this report. All we need to know is that fitness trackers that can be purchased on Amazon for as low as £20 are able to accurately track a person’s sleep quality during the night.

From data collected on the fitness tracker it is possible to accurately estimate how long an average sleep cycle for that person is. A person needs to sleep for about 5 sleep cycles to feel well slept given that they have a low sleep deficit over the previous 7 days.

Say Alice’s average sleep cycle is 93 minutes. This means that she must sleep for about 7 hours and 45 minutes to feel well rested, given that she slept well in the previous week. So for this example Alice’s required daily sleep is 7.75 hours.

Using each night’s sleep duration, quality and her required daily sleep we can calculate how well rested Alice is feeling at that point in time:

|  |  |  |  |
| --- | --- | --- | --- |
| Day | Duration | Quality | Effective Sleep |
| Mod | 7 hours | 94% | 6.7 hours |
| Tue | 5 hours | 72% | 3.6 hours |
| Wed | 8 hours | 89% | 7.1 hours |

Over the past 3 days, Alice’s total effective sleep was 17.4 hours. She should have slept for 23.25 hours. This means that she is missing 5.85 hours of sleep in order to feel well slept. Over the past 3 days Alice has accumulated a sleep deficit of 25.2%.

Using Alice’s sleep deficit and her concentration value at a point in time it is possible to assign a health percentage to every line of code that she writes.

Code health along with an employee’s ECPS (from the previous section) can tell with a pretty high accuracy what percentage of an employee’s full potential a line of code was written at. To continue with Alice from the previous example, Alice’s ECPS is 85. The method averageSales(int allSales[]) she wrote was given a health percentage of 87% according to this software. The method averageSales is therefore rated with a score of 74.

(74 = 85 \* 0.87)

Browse Reporter, [browsereporter.com](http://www.browsereporter.com/)

ActivTrak, [activtrak.com](https://activtrak.com/)

Tobii, [tobii.com](https://www.tobii.com/)

Sleep Cycles,

[psychologytoday.com/blog/between-you-and-me/201307/your-sleep-cycle-revealed](https://www.psychologytoday.com/blog/between-you-and-me/201307/your-sleep-cycle-revealed)

The Impact of Sleep on Work Performance, [solutionsatwork.brighthorizons.com/~/media/BH/SAW/PDFs/Consulting/2015-HWC-Sleep-Study.ashx](https://solutionsatwork.brighthorizons.com/~/media/BH/SAW/PDFs/Consulting/2015-HWC-Sleep-Study.ashx)

**Monitor employee’s social interactions and personalities**

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 ECPS (from the section titled USING INFROMATION ALREADY AVAILABLE ON EMPLOYEES) management can receive very deep insights about office politics and who writes the best code in order to allow management to clearly see who would be better suited in a different team without making an impact on the current team’s productivity, allowing management to rearrange hierarchies more effectively to meet their required goals.

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 cameras 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 not a push to see both of these technologies combined to create tracking cameras. These cameras may then be set up at strategic places around the office to monitor the employees, identifying them and monitoring their behaviours such as writing code, taking a break etc. These cameras could identify social circles within the company by tracking which employees interact with who. 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.

The company may also ask their employees to take personality tests and then compare the results with the employee’s ECPS to try and find trends between personality and ECPS. The Big Five traits are seen as the most accurate model of a person’s personality at the moment. A lot of information is currently known about how people with different Big Five traits get along.

An engineer’s personality data, social tracking data and ECPS data may then be used to create teams of optimal compatibility. Depending on the task, each team could be assigned with a required ratio of influencers, where statistically each team member would get along well enough with each other based on personality data and where each team has a high enough average ECPS.

Depending on the role of the team, each team may have a different mix of developers. For instance Team A, which mainly focuses on high turn-over critical bug fixes, would be assigned with people the perfect ratio of influencers to team members, people with a higher ECPS and people who are more compatible with each other. Team Z on the other hand, a team that is working on an add-on that 0.5% of the users will be using, may be built out of people who didn’t make it into the more important teams and thus Team Z would have an imperfect ratio of influencers to team members and a lower average ECPS and compatibility score than Team A.

Upon processing all information from the personalities of each team member and social tracking data that is being dynamically collected at all times each team would then be assigned a dynamic Team Synergy Score. The better the score, the better the team is working together.

Each line of code the team writes could then be scored using something similar to this:

L = Line of code’s score

E = Author’s ECPS

C = Author’s concentration and tiredness score (from previous section)

S = Team Synergy Score

L = E\*C\*S

Scoring each line in this way would allow for software to be measured not only by how well the tests run but also by the quality of the team and the person that wrote it.

Big Five, [en.wikipedia.org/wiki/Big\_Five\_personality\_traits](https://en.wikipedia.org/wiki/Big_Five_personality_traits)

<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)

* Useful for very niche cases where data can’t be automatically collected or analysed by existing software
* Time consuming
* Mentally taxing
* Collected data is not useful to management

Outro (250)

What I did

My conclusion