# Measuring Software Engineering

**Software measurement** is the study of pieces of software themselves: their properties, relations between these properties, and which among these are interesting and which aren't.

One can think of a piece of software as an entity which is a detailed, functioning implementation of some algorithm (an abstraction of the software). Software measurement can provide insight into the quality of the software as a "solution" to the algorithm, allowing different versions to be compared and contrasted, and suggesting potential improvements.

Software metrics can range from simple (measuring the number of lines in a file, functions in a program or files in an application) to more complex. One such complex measurement is ingeniously named “software complexity”:

Software complexity is a measurement referring to the internal properties of a piece of software. In general it is a function of the interactions between a number of entities. As the number of entities increases, the interactions between them rise exponentially. This creates complexity, rendering the software more difficult to understand purely by virtue of the amount of memory required to reason about it.

In 1977 Maurice Halstead developed a set of complexity measures (this was part of his effort to raise software development to an empirical science). In fact, his measures encompass much more about software than just complexity: the equations are designed over properties of software and the relations between them. [1]

Another pair of measurements is coupling and cohesion. These provide insight into dependency and efficiency of entities within an application.

Coupling is the degree to which a module, class or other construct is tied directly to others. For example, the degree to which one class is dependent on another in an object oriented language. [2] A highly coupled program is fragile; making a change in one place may create a bug somewhere else.

Cohesion is the extent to which the parts of a system work together to produce the desired output. Just as a team in which each member adds their unique strength renders the team greater than its component parts, a cohesive program uses the best parts of each of its entities in an efficient way to do its job. Conversely, a team which is lacking in diversity and expertise will not be doing anything very unique or useful.

In general it is best to aim for low coupling, high cohesion and strong encapsulation. Aiming for these ideals will result in better readability, code which is easier to understand and easier to change. That being said, a program with zero coupling is completely useless. No information could be input or output. A balance must be struck which suits best the requirements of the situation.

Having covered the tools we can use to inspect software, we can understand that the **measurement of software engineering** is distinct from software measurement. Whereas the latter's domain is the space of software objects, measuring software *engineering* is the science of gaining insights into the engineering process itself. The stuff that this science deals with is human behaviour. We ask questions like, "*What are the properties of our development practices (as a function of the data we are allowed to get our hands on)*?"; "*Which of these properties are useful to us in improving the quality of our end products*"; etc.

Measurement can even be distinguished from a process plan, which details how one intends to go about developing a product (eg: Waterfall plan, Agile development). Instead, measurement is concerned with gathering data on what has *actually* happened, and using the information to try and improve quality and efficiency in the future (ie: with the intention of highlighting and improving poor techniques or environments. Although it could be argued that to measure or not is simply an implicit component of any process plan, I think it is useful to conceptually distinguish them: there are measurement techniques and then there are process plans and they serve different functions).

Measurement can range from simply recording bugs in a text file, to analysing frequency of repository commits, to a maintaining a full-blown database of checklists, reports and graphs for all aspects of the development process.

An entry in *ecomputernotes.com* breaks measurement down into 5 steps[3]:

- **Formulation:** This performs measurement and develops appropriate metric for software under consideration.
- **Collection:** This collects data to derive the formulated metrics.
- **Analysis:** This calculates metrics and the use of mathematical tools.
- **Interpretation:** This analyses the metrics to attain insight into the quality of representation.
- **Feedback:** This communicates recommendation derived from product metrics to the software team.

So we see that software measurement can in fact used as a component of engineering measurement, providing insight into the quality of the software produced.

A measurement process can be used in many ways, from advising an individual engineer on how they can work better, to determining which configurations of employees make the best teams, to determining pay or letting employees go.

**Why would anyone want to measure software engineering?**

Individual developers will be interested in improving their own efficiency, both for monetary profit and to reduce time wasted coding (which is never a good thing). Similarly, employers and companies have a huge incentive to make their developers more efficient, as this translates to increased profit. This is why companies such as Google and Microsoft are well known for their in-office perks: free food, gaming rooms, music rooms, sports facilities, etc. These facilities are not there because the companies just want to make people feel good. They have found (presumably) that these environments improves their engineer's development process.

So, is measuring software engineering actually effective? What evidence is there that it is worth the trouble? Joel Spolsky (CEO of Stack Overflow) asserts that "*any metric can be gamed*"[4] , meaning that whatever isolated property/system of properties one uses to judge, a subject can (and will) tend towards satisfying just those targets, instead of holistically improving the quality of their output (ie: you incentivise making work look good on paper, rather than creating a good product). Conversely, Nader Akhnoukh argues that such metrics are poorly designed: metrics *do* work when they are used appropriately.[5]

In this essay I will explore and review the history of software engineering measurement, the prominent metrics used today, and touch on potential developments in the future of this field.

**A Story of Engineering Measurement**

The Software Process Program was a research group founded in the 1980s at Carnegie Mellon University by Watts Humphrey. The aim of the program was to improve methods of understanding and managing software engineering as a process. Why was this important to the so called "father of software quality?" Having a background in physics but also having experience with innovative technology and software companies (he joined Sylvania Labs in 1953 and later became a vice president of IBM), Humphrey would have been aware of the need for rigorous, empirical methods of analysis for software engineering (which to this day have not been perfected). As well as this he was acquainted with the practical difficulties that growing software teams face.[6]

In 1989 Humphrey published "Managing the Software Process", a book intended to emphasize the principles of good software design within an organization. This book was the starting point for his later work on the personal software process (PSP) and the team software process (TSP) concepts.

PSP could be viewed as one attempt to implement what Fred P. Brooks Jr (author of "No Silver Bullet") envisioned when he recommended that the software community should "learn good hygiene". In the 2009 PSP "Body of Knowledge", Humphrey echoes Brooks, perhaps revealing the origins of his intentions with PSP:

"*Since the PSP is a set of practices and methods that enable software developers to control their own working lives, when competent professionals learn and consistently follow these engineering and scientific principles...it is our hope that the software community will develop the courses, the support tools and methods, the certification and qualification programs, and all of the other required elements to enable the widespread adoption of these methods.*" [7]

Key components of the PSP cycle are prediction, measurement and "post mortem": in the prediction phase engineers are required to specify their intended development of code. Measurement occurs during the development of the software; the engineer is required to (manually) collect data on their behaviour and efficiency (eg: time spent coding, bug error rate, program size (NB: software measurement!)). In the post mortem phase, the data generated during the development process is analysed and compared against predictions. Furthermore, data accumulated by the engineer from any previous PSP analysis is incorporated into their estimation for future programs. PSP encourages an evolutionary maturation of the engineer's techniques by introducing these practices gradually (see PSP0-PSP2).

The team software process (TSP) serves a similar function for allowing evaluation of a team of engineers, with the intention of maximizing group cohesion and helps organizations establish a mature and disciplined engineering practice that produces secure, reliable software in less time and at lower costs. [8]

The SEI actually provides TSP as a service, providing coaches and training programs. An organisation who want to use TSP may prefer to train their own member of staff to ensure privacy. This can be useful since the companies most in need of such training will be ones which are just starting out and need to develop good habits while maintaining the edge that their "secrets" give them.

**How do PSP and TSP differ?**

PSP is in fact a prerequisite for all team members participating in a TSP program. PSP is designed to augment an individual's performance, and only after that has been implemented does team optimization (TSP) make sense.

Unlike with PSP, which focusses on engineers, TSP requires training and effort for all types of team members. Team leaders and management must participate too.

**Are there any downsides to PSP?**

In a 2002 study, Finnish researchers found that PSP did not improve the estimation skills of their test subjects, however product quality was improved. Nor did it reduce productivity, indicating that there are indeed benefits to the practice.[9]

Many engineers complain about the process overhead, and invasive nature of the analytics (eg: someone may be fired on account of their defect rate, even if they are still producing good code, this is all the company might see/have to compare)

**Why am I talking so much about PSP?**

Humphrey admits that "*the PSP is almost certainly not the last word in software development practice*". But PSP and TSP are among the most well-known metrics for attempting to improve a software project or team's functioning. Despite its infamous measurement overhead and invasive analysis, it was one of the first successful attempts to measure inaccessible yet useful data, and provided software engineers with a way to reflect on and improve their craft in a way which hadn't been fully considered until its development. It is commonly the standard to which other metrics are compared and it is still under development. In a similar spirit to Agile development, PSP encourages evolutions to its methods/ additions to its body of knowledge from experienced users. This allows PSP to improve over time, a definite advantage. All of the above make PSP a solid baseline to build our measurement techniques from. [10]

## Under the streetlight analogy

In a 2013 article, Philip M. Johnson uses an illuminating analogy to help explain the varying efficacy of different measurement techniques. The analogy comes from an old joke (after which the Streetlight Effect, a common observational bias is named):

*A policeman sees a drunk man searching for something under a streetlight and asks what the drunk has lost. He says he lost his keys and they both look under the streetlight together. After a few minutes the policeman asks if he is sure he lost them here, and the drunk replies, no, and that he lost them in the park. The policeman asks why he is searching here, and the drunk replies, "this is where the light is".*[11]

Johnson points out that traditional measurement techniques (ie: using readily available data, such as total lines of code written per hour) are akin to searching where the "light" is rather than bringing light to where you want to see: just because the data is all that's available, doesn't mean it provides the best answers.

The PSP is a good example of a technique which provides a wealth of relevant data, specific to the individual user/software and which wouldn't be otherwise available. The downside is a large overhead of manual effort for the programmer, as well as a certain level of invasion into their privacy. This makes PSP "*like a campfire*": powerful yet inflexible.

An example of a platform which attempts to remedy this is the LEAP toolkit. Developed by the Collaborative Software Development Laboratory (CSDL) at University of Hawaii, LEAP's was intended to build on the original PSP by automating data analysis. One goal was to minimise the conclusion error rates found in using PSP. LEAP is lightweight because it relieves users of a lot of work. However the downside is that this inherently results in less flexibility, as the user cannot tailor the results to his or her (perhaps unique) needs, this being the key benefit of the PSP!

Hackystat is another toolkit to come from CSDL. It was designed by first "blindly" building automated data generation techniques, instead of designing towards a specific process/ type of development (ie: not considering how the data collection techniques might be used). During use, the Toolkit attaches sensors to development tools which gather process and product data. This is sent to a server from which higher-level analyses can be carried out.

While Hackystat has many strengths where other processes are poor (particularly in its powerful automatic collection of data), by virtue of this it generates a weakness: socio-political objections to the ethics of using such a powerful monitoring tool. [11]

**State of practice : what is used today?**

Today many platforms are available, however the most common tools used for software analytics tend to resort to "searching under the streetlight": Sonar, Atlassian, DevCreek and many more platforms all use readily-available data (eg: defect-tracking, lines of code) to draw conclusions about software quality. As explained above, while this is suitable in some cases, it can be totally useless and even misleading in others. [11]

Why have these tools risen to prominence over methods like PSP, when they seem like a step backwards? These methods are probably so popular because they are the low-hanging fruit of measurement techniques: as with any meme they can be expected to be the most widespread due to their ease of uptake. People naturally dislike the uncomfortable and painstaking process of manually logging data themselves, and similarly are uncomfortable with infractions on their privacy. This combined with a lack of research and education into the importance of analytic techniques leaves many engineers opting for the path of least resistance.

**What can we expect in the future?**

Humphrey may have been the father of software quality but he is certainly not the end of the line. He admits in the PSP Body of Knowledge that PSP itself continues to be unfinished. While advancements have been made, the field still suffers from a lack of a cohesive, accepted framework for measuring progress.

Despite the range of options available, we do not seem to be converging on a single accepted best practice. Instead the consensus appears to be that each situation and context calls for blends of existing techniques; engineers and managers must find the right trade-offs for their situation.

For example while aspects of PSP are still useful and relevant, its traditional measurement of syntax errors is practically redundant with modern IDEs.

## ETHICS - What's all the fuss about?

Despite the advantages that measurement can provide, many engineers and managers are wary of and protesting against the moral implications of probing into an employee's life. But this is not limited to the software engineering sphere. To understand the ethical problems in measuring software engineering is to understand the same problems surrounding Big Data and data analytics.

Johnson (Searching Under the Streetlight) proposes that the trade-off between ethical breaches and depth of analytics is irreconcilable, that we are unlikely to ever uncover a method which yields vastly more useful analytics without ethical objections.

Data Analytics is the science (some would say art) of uncovering relevant data and gaining insight from it (collection and interpretation). With the coming of the internet and proliferation of personal devices generating more and more digital information, data analytics has become an indispensable tool for any organization wishing to be competitive, from companies to political campaigns.

While it is powerful and can bring valuable insight, like all technology Big Data remains a mere tool and is thus amoral: with this power comes the potential for far-reaching misuse as well as aid.

Even seemingly innocuous and unrelated behavioural data (such as consumption habits) can provide shockingly detailed (albeit probabilistic) insight into one's life. In "*The Power of Habit*", Charles Duhigg describes how in the early nineties the American supermarket chain Target had built a database of its customers and their spending habits. With this information they were able to build profiles of each individual, and through statistical analysis they started finding more and more effective ways of delivering targeted, individualised advertising. [12]

One data analyst found that there were very specific spending patterns for women during varying stages of their pregnancy. This allowed Target to send highly focussed advertising to individuals (eg: care products near the start of the pregnancy, prams and toys near the end).

One day an angry customer phoned Target management, complaining that they had sent his teenage daughter coupons for baby products. The manager had no idea why this was the case, but apologised to the upset father. But when the manager courteously phoned back a few weeks later to apologise again, the father told him that, unbeknownst to him, his daughter was indeed pregnant. Target had "known" a secret about this man's daughter that he himself didn't.

While this was shocking at the time, I think that no one would be surprised to find a company capable of doing this today. We knowingly leak data about our behaviour every waking hour of the day to organizations such as Google and Facebook. What obligations are these companies under to handle the information we give them? Such questions are still hotly debated.

On the one hand, the benefits for companies are tempting, and even users benefit from giving up their behavioural data (the more Google can learn about you the more relevant its search results can

be). And besides (one might say) companies aren't doing anything malicious with them at the minute.

On the other hand we are clearly toying with a slippery slope here: what it is acceptable to give up today will likely continue to be acceptable in the future, but there is no guarantee that organizations will not at some point grossly misuse this data, and by then it will be too late. Now is the time to prevent such problems, not when they arise (then it will be too late).

How does this translate to the software engineering field? Well when it comes to measuring performance and tracking progress, similar questions arise. Does a company have the right to measure an employee's keystrokes? Who owns your keystrokes? What about biometric data: is it acceptable for companies to monitor this? If so, is it only ok when in the office, or should they be able to monitor activities at home too?

Playing the Devil's advocate, one might argue that, after all, they are employing you: you are an asset to them and they want to know if you are doing things which may impact the company. If you don't like that then you can leave…

Obviously few people would agree with such an extreme position. Consider the stress such practice would put an individual under. Consider what a malicious agent could do with this wealth of data, how a tyrannical CEO might abuse employees to manipulate them.

Even disregarding these hypothetical consequences, most reasonable people will agree that constant and widespread monitoring is immoral and an overreach. Do we really want to set up systems to allow such things to be possible? To be conscious that we are paving the way for such potential misuse is surely itself a kind of malicious act. Certainly it would not be ethical.

**Recent developments in Data Law**

The GDPR is an upcoming, EU-wide regulation bringing new rights for individuals by restricting the processing and analytical ability of companies, businesses and governments. A big advancement is that it will extend regulations beyond the EU itself to *external* bodies which interact with the EU. This issue of the limits of data sovereignty is a contentious political issue which has been evolving alongside information technology.

Data sovereignty is the concept that digital information is subject to the laws of the country in which it is located. A pro-business website "*intralinks.com*" states that, "*sweeping new data privacy laws, like the upcoming General Data Protection Regulation (GDPR), significantly restrict how certain types of information may be stored and used by organizations – and are being enforced by increasingly stiff fines and penalties*."[13]

However another source describes that:

> *"The GDPR builds on the current fair-processing requirements by increasing the amount of information that you must provide to data subjects when collecting their personal data, to ensure that such processing activities are fair and transparent. Organisations must provide the information in an easily accessible form, using clear and plain language."*[14]

I think this is an important point: companies should have an obligation to make what they are doing with "your" data (the data you are agreeing to give them) with as clear as possible. Without

regulations on this and putting the burden on the companies there will always be an easy way out in cases of misuse: "*they signed up to it in the fine print*".

Not all laws in relation to this subject are considered positive, and not all are in relation to companies. In UK politics, Theresa May's Internet surveillance bill was widely opposed by the online community. It was surreptitiously slipped through parliament when the world's attention was on Donald Trump's rise to power.

Another relevant legal issue is the right to be forgotten, an EU movement to put in place laws allowing deletion of embarrassing historical information on an individual. I'm unsure where I stand in this case. There are cases where someone is clearly the victim of smear campaigns or unfair slander. It could start to create precedence for censorship of the internet, which could have a much more harmful impact.

**Conclusion**

In conclusion, both in terms of measurement effectiveness and ethical considerations I think that many attempts have both positive and negative aspects, and we need to find techniques which strike an acceptable balance. The field has made a lot of progress over the past 50 years, but there is still plenty of room for improvement and new innovations.

For example, an increasing number of companies (such as consulting firm Deloitte) are using reward mechanisms uncovered by the video games industry. This (Gamification) seems like an example of a measurement/process method which will improve worker happiness and productivity. [15]

However it does seem possible that it could lead to a runaway competitive streak in employees. Game addiction is a recognised phenomenon, and merging these reward mechanisms with work could lead to more widespread "workoholism" in society.

The future is uncertain we don't know what dangers lie ahead and I think people are right to be cautious, but at the same time we mustn't stifle ourselves because of fear.

I recognise that each situation has its own ideal balance which serves all parties best, however this is impossible to implement in law. I agree with Johnson that we will never reconcile data analytics with the individual's desire for privacy. I think that the only solutions which satisfies both interests are those approaches which ensure careful attention to context and balanced consideration as we make progress.

**References**

[1] https://en.wikipedia.org/wiki/Halstead_complexity_measures

[2] http://www.codemag.com/article/1001061

[3] http://ecomputernotes.com/software-engineering/software-measurement

[4] https://www.joelonsoftware.com/2005/07/25/hitting-the-high-notes/

[5] https://www.linkedin.com/pulse/you-can-should-measure-software-engineering-nader-akhnoukh

[6] https://en.wikipedia.org/wiki/Watts_Humphrey

[7] https://www.sei.cmu.edu/reports/09sr018.pdf

[8] https://www.sei.cmu.edu/tsp/

[9] https://www.sei.cmu.edu/tsp/

[10] https://www.sei.cmu.edu/reports/09sr018.pdf PAGE 2

[11] *David H. Freedman (2010). Wrong: Why Experts Keep Failing Us. Little, Brown and Company*

[12] *The Power of Habit (2012) Charles Duhigg*

[13] https://www.intralinks.com/data-sovereignty#

[14] https://www.irishtimes.com/special-reports/ireland-us-business/ireland-poised-to-benefit-from-new-data-laws-1.3126599

[15] https://www.wsj.com/articles/SB10001424052970204294504576615371783795248