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.

Abstract:

This report attempts to capture the Software Engineering Process and examine the ways it can be measure. It approaches this from a few different perspectives initially trying to accurately quantify the process with a number of tools, and then talks about the ramifications of those tools.

Ways the software engineering process can be measured:

The software engineering process (SEP) is an arbitrary process that has no standardized quantifiable metrics by which to examine it. That is to say collecting data and comparing SEP's is a non trivial process.

Thusly we must create our own and apply them in an attempt to capture the development processes and effectiveness of the developed product. The SEP differs from project to project, and organisation to organisation. Certain projects may prioritise problem coverage, while others could value security or run-time. Each of these are valid metrics for measuring the results of the SEP but what we desire are tools to examine the process as it is in motion, so as to refine and improve development efficiency.

We can break down software measurements into two groups: direct measures & indirect measures: Direct measures are metrics that don't depend on any other variables to examine, an example of this would be lines of code.

Indirect, or derived measure are metric that require multiple variables to examine, they're a function whose domain is an n>1 tuple e.g M: (X,Y,Z) -> A. An example of these metric would be programmer productivity, or defect density (the number of defects in a project divided by the size of the project.)

Direct measures are easy to collect, but generally give us less insight into the SEP than indirect measures. Lines of code can give a surface level insight into the size of a project, but there are so many unaccounted for variables that it does not give us an accurate depiction of the entire SEP.

In order to accurately measure the SEP we must capture both direct and indirect metrics and need a reliable way of doing so.

The paper "searching under the streetlights" by the University of Hawaii gives us a valuable insight into how we might go about capturing these metrics and the tools

available for us to do so. It will be my main focus and source for this section of the paper.

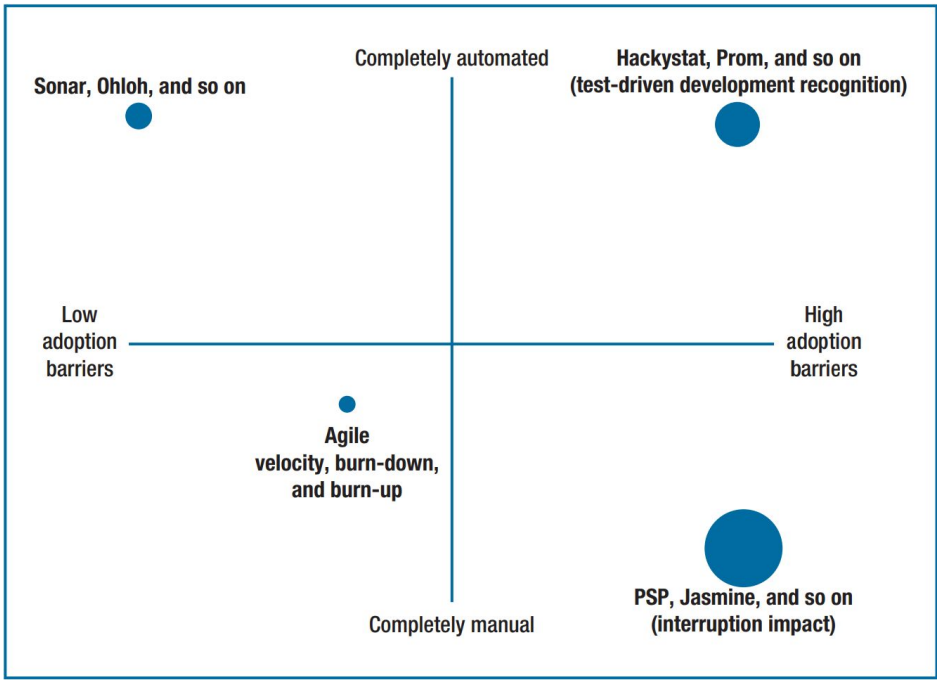
Searching under the streetlights is a useful metaphor for finding the balance between using easily available but useless (or not very useful) data and creating a large overhead on a project by spending so much time trying to collect useful data for measuring the SEP. Base on the form of observational bias of searching for a lost item under the streetlights because it's easier to see, regardless if the lost item was lost there.

Measuring the SEP is a crucial part of software development as it tracks progress and identifies inefficiencies but the time spent measuring can create an overhead that takes away from development time, itself becoming an inefficiency in the project. The issue of finding this balance is addressed in the following graph [fig 1.]

The points on Figure 1 represent different methods of collecting data on the SEP, which I will go through in detail in a moment. Before that it's important to consider the X & Y axes They denote the distinction between data that is collected manually & automatically (on Y) and the barrier to adoption. Barriers to adoption can be a number of things - program complexity, size, but is normally human factors, like in the case of Hackystat, developer discomfort with the level of data being collected. This is something I'll address further when speaking of the ethics surrounding the SEP.

The PSP is a software dev process designed to improve performance by tracking predicted and actual development of code. Using the PSP a developer tracks their own progress and manually fills out a number of data fields including project plan summary, time recording log, defect recording log, process improvement proposal, size estimation, time estimation, and design and code checklists.

While thorough and personalized this involves the developer filling out each of these fields manually, which adds both a high barrier to entry (as devs won't typically want to perform this work diligently and accurately) & a large overhead as dev time is spent tracking this info.



[fig 1]

The PSP also seems, at least theoretically, vulnerable to manipulation & discrepency . Developers that are behind on deadlines could inaccurately fill out forms, or introduce human error to the data collection process.

Automated data collection tools solve this overhead and the barrier to entry problem outlined in the PSP, but can introduce new problems. Hackystat, for example, service-oriented architecture in which sensor are attached to development tools to automatically gather SEP data and send it to a server where higher level analysis can be performed on it. While this is an extremely efficient means of collecting data, developers express concerns over the intrusive nature and transparency of the data collected. When developers have their entire process monitored, which can then be scrutinized by anyone with access to the server, it can lead to an uncomfortable work environment and, in turn, less productivity. Client-side tools can collect minute by minute data on the SEP. Developer discomfort came in three main facets: FIrst, he unobtrusive nature of data collection, considered a feature by Hackystat many devs didn’t want data collected about them without their knowledge. Second, the fine-grained data collection is viewed as intrusive as it provided transparency about each developer's style, which in turn creates discord in a development team.

This issue is solved by software like Sonar & Ohloh, Data collection is entirely automated but the service simply applies analytic techniques to the data. Overhead is low and the data is generally uncontroversial - it focuses on the product characteristics rather than the developer behavior that resulted in them. While these process have low overhead and low barrier to entry they correspond best to the searching under the streetlight metaphor, as the data is easy to gather and non-offensive, but often not as useful as that collected by more intrusive or higher

overhead systems. The field of how to measure the SEP doesn't seem to be converging on a single correct answer but rather a number of techniques, old and new, that explore the tradeoffs between simplicity, effectiveness and acceptability.

Additionally fully automated data collection does not account human factors in the SEP that could be accounted for with a manual process, for example they can't account for interruptions to the coding process and blindly collect data. If someone left their IDE open when they went to lunch that would be recorded and cause data discrepancy.

Finally to speak for a moment on the Agile software design approach: Agile SEP's follow a different structure and so the tools outlined above, for the most part, don't exactly apply. While you could still use an automated data collection tool it would give less insight as the documentation, goals. The agile manifesto explicitly values individuals & interactions over processes and tools.

However it is entirely still possible to measure the SEP in an Agile work environment. We need simply examine the workers adherence to Agile principles on an n-monthly review basis, Additionally tools like velocity (stories completed/time in a SCRUM system) serve to give us insight into Agile developments. While velocity doesn't measure the quality of the product delivered those concerns are addressed by bringing the customer into the development process and ensuring their satisfaction. While this paper doesn't focus on measuring the SEP of Agile environments I thought these factors were worth mentioning as Agile processes are becoming more and more popular.

Assessment of measurable data:

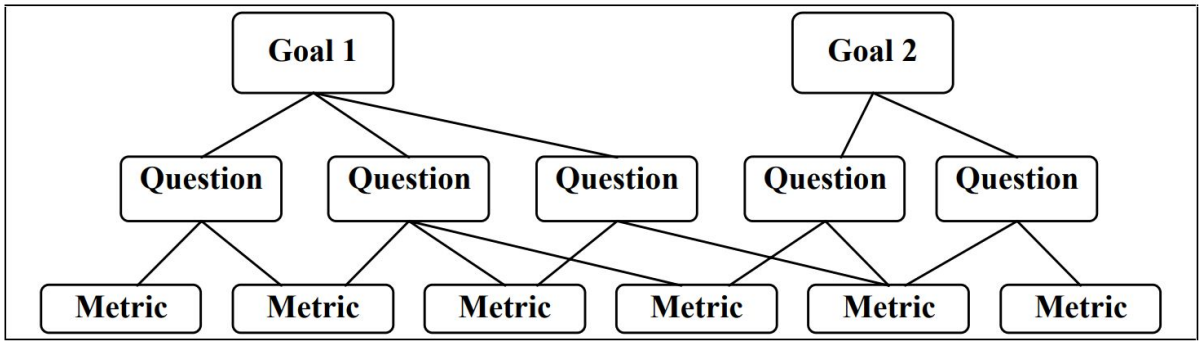
Assessment of measurable data is a key part of measuring the SEP. While in many cases it can be done automatically by the tools outlined above, often we must defined usefully analytics to derive meaning from collected data.

Even with the assessment tools like HackyStat and Sonar above, we must value certain data attributes above others and tailor our assessment of data to projects on a case by case basis. Certain project may value security and speed while others value code coverage and scalability. A useful tools for assessing this is the goal question metric approach outlined in the paper of the same name by Victor R. Basili, Gianluigi Caldiera, H. Dieter Rombach2.

On a conceptual level a goal is defined for an object, with respect to models of quality, points of view, and relative to a particular environment. These objects can include products like specifications, designs, programs & test suites, processes like specifying, designing, testing and interviewing and resources like hardware, software and personnel.

A set of questions are then outlined to characterize the assessment/achievement of a certain goal Question attempt to capture the object of measurement with respect to identified quality issues and determine it's quality from a selected viewpoint. Finally a set of data is associated with the defined questions in order to answer them as quantitatively as possible. The data can be objective or subjective.

Figure 2 outlines this process



[fig 2]

Our assessment of measurable data is defined by the metrics at the bottom of this hierarchal system but it is the goals and questions that give meaning to those metrics.

In the system being outlined by the paper, the metrics would be recorded using some form outlined above like the PSP or Sonar, and meaning would be given to it by outlining meaningful goals and queries of the data.

Outlining goals involves specifying a purpose, often to improve or refine, and given quality issue, e.g. time, size, defects, via a defined object. As defined above these object can be specifications, designs, or processes, which means the GQM model can be applied to any stage of the SEP. We must also define a viewpoint for our goal, most likely from the perspective of a project manager or customer.

Once these goals have been outlined, (e.g to capture programmer productivity) we devise questions around this process. E.g. What are metrics that current impact productivity?, Is the current productivity satisfactory from the perspective of Project Manager? Is the performance improving?

Now we can take the data we have collected and apply it to these defined questions. The tools used for data collection can be useful in showing us graphed trends which make answering these questions easier, especially questions of projected improvements. The application of our collected data to these questions gives us metric through which we can view the SEP.

The GQM is just just one model of assessing data however, and it will be useful to talk about data assessment in the abstract and how it can be use to improve the SEP. The general format of data assessment is that collected data is correlated with modeled attributes to assess compliance with expectations. Without these models of expected behaviour the data collected during the SEP can be meaningless and abstract.

The data collect is only attributed meaning in the context of the software project and the expectations/goals of that project. Additionally, as seen in the GQM, certain group of data are useful to different people, for example metrics like code coverage, security and runtime are useful to the customer/project manager as they assess the effectiveness of the proposed solution, while metrics like time spent coding, number of commits, and velocity (in the case of Agile development) are useful for the programmer to understand their workflow and make improvements to their efficiency.

However it can be difficult to assess indirect metrics that pertain to software engineers, for example not (immediately) quantifiable soft skills like communication, self-management, collaboration etc. In order to achieve an optimized SEP we will need to hone an refine these skills also.

One paper, entitled Mining developers' communication to assess software quality: Promises, challenges, perils (Di Penta, M - 2012) proposed a method of examining not only the structure of code, i.e. the introduction & solution of bugs, and problem approach, but also the communication, in natural language, between developers. These include tags, commit notes, or comments developers produce. While this concept seems fruitful, without powerful language processing software and human interpretation (causing a large overhead) it wouldn't necessarily work in practice. This is an example of a challenge faced in the field of data assessment. While natural language data is available to us, and could potentially give us greater insight into the developer process than quantifiable metrics like develop time or lines of code, its processing is rarely as efficient as assessing quantifiable metrics, which we can automate and graph.

A solution to this can be seen in the Agile Software Development method, more notably the paper "Towards Data-driven Software Engineering Skills Assessment by Jun Lin, Han Yu, Zhiqi Shen and Chunyan Miao". In which 125 Software Engineering students were examined using the SCRUM system. In it equations for Technical productivity, competence, and Morale are devise and graphed before and after a Sprint.

Overview of computational platforms:

Outlined in section 1 of this paper there are a number of computation platforms for measuring the SEP. In this section I will go beyond the aforementioned tools that track the design process via the workstation and look at some of the newer platforms for collecting mid-high level data. and compare and contrast them against each other,including the response they get both in terms of capturing the SEP and the response from developers when they are implemented.

The balance when finding computational platforms to measure the SEP is trying to get accurate and thorough data, but not disrupting the development process.Additionally there are concerns about the intrusiveness of the data collection, the impact on workplace culture, and the security surrounding the collected data.

When attempting to track higher level data regarding the SEP we must take a more abstract, and unfortunately more invasive approach; but first we should define the type of data that we mean to examine.

Beyond things like run-time, lines of code, problem coverage, development time, commit-rate, (these being low level data) in order to fully capture the SEP we want to examine factors like Happiness, Social Graphs, Routines, as well as measure values like influence, leadership, workplace dynamic, etc.

The first example is UEBA, User & Entity Behaviour Analytics. This is a technique that is being implemented in some cases as a security solution, but can be expanded to cover the entire SEP. Companies like LogRhythm and Aruba (with their Introspect project) for example have implemented UEBA as a means of cyber defense.

The UEBA uses algorithmic and machine learning techniques applied to team communication channels, files and website access history, and other online interactions between workers. Its apparent that these aspects can be monitored for data security reasons, as data value has skyrocketed, and implementations like cloud based system and access to program files from multiple platforms means that data security is no longer a matter of keeping data security at entry and exit points but also within a project.

While keeping data secure is one means of measuring the SEP, we can extend the UEBA theory to monitor social influence, an employee's flight risk, an employee's happiness in the workplace or other hard to measure, abstract concepts.

When we see deviations from the norm using UEBA we can infer data from it.

What we see more and more of is the implementation of technologies like the UEBA in the workplace to track their employees.

Humanyze is a company that specialises in personal analytics and has designed products like smart employee badges that listen to conversations between employees. It can measure how employees interact, tone of voice, track locations. All of this can be useful analytic data when trying to measure non-trivial aspects of the SEP.

Steelcase is another company that specialises in furniture that measures health aspects of employees, like posture, stress levels, heart-rate. Each of these can be useful for examining work-rate and stress level to ensure a comfortable workplace and that employees aren't putting themselves through intense levels of stress around project deadlines.

Additionally computer visions advancements could be used, in theory, to track social groups of developers in a workplace and monitor employee movements, again capturing an increasingly thorough view of the SEP.

Ethics:

This all leads fairly naturally into the field of ethics when measuring the SEP. Ethics has a massive role to play in any facet of data collection and should be considered no differently when collected about employees vs users. A number of the data collection techniques outlined in this essay have massive ethical ramifications.

Ethics is a principled line of argumentation that centres around inate human ideas of comfort, security and freedom. However it is often tough to quantifiably justify ethical practices especially in the workplace environment where they can easily be sacrificed for quantifiable increases in productivity. When we can increase profit or output by subverting ethical practice then it can be easy for management to value ethics less.

In the previous section I mentioned how certain developers felt uncomfortable when using a SEP monitor like Hackystat, because of how transparent it was, how thorough and personal the data collection was, and how other developers could see intimate details of their coding process. The exact same can be said for the systems outlined in the above section, tracking something like employee conversations is categorically unethical as it places them in scenarios in which they lose freedom of expression, comfort when interacting with employees, and job security. Practices that focuses too intensely on data collection for monitoring the SEP without considering the opinions and feelings of the developer will lead to workplace

environments where developers are uncomfortable.

Additionally since this ethical practice is linked to their primary source of income, developers subject to these practices have less of an opt-out than consumers. One could contend that if a developer is uncomfortable with how his company monitors them they could simply not work there, but this is unfeasible.

If the implementation of unethical practice does increase productivity, then it's natural to assume that competing businesses will adopt such practices in order to not fall behind, this gives the software developer no choice but to participate in businesses like these in order to keep a job. The developer becoming a voiceless cog, who is strong armed into sacrificing personal liberty for job security and corporate interests.

Moreover, I think it's reasonable to assert that the implementation of Orwellian systems of monitoring would in fact have a negative impact on productivity, in that when developers are less comfortable, less happy and less secure in their work they will not have the motivation/ability to work as well. We see this phenomenon - a happy worker being a google worker, in the culture of successful software development companies like Google, in which workers are generally given freedom to pursue their own interests and things like in-house chefs and toys are uncommon. It stands to reason that if a comfortable work environment is conducive to good workers, then the opposite would be true when it comes to data collection.

However, we run into a common problem of observation from quantum physics here in that, in order to assess the productivity of a worker, we must measure it, but by implementing measuring tools we inherently change the total productivity by ethically infringing on the developer.

For this reason we must strongly examine the usefulness of the data collected and the ways in which we collect it. We must balance the usefulness of listening to our employees' conversations and monitoring their heart rate with the ethical intrusion that these practices entail. We must heavily weigh up the benefits we can extrapolate from this data vs the price of it's collection.

It is easy to think, especially from the perspective of a computer scientists, that collecting the highest amount of data possible would be ideal. However the collection of data has both a cost and an overhead, the cost in the scenarios being ethical infringement on employees & consumers, and the overhead being both the time it takes to collect, and the corruption of that data by the act of collecting it.

Algorithmic approaches:

One (partial) solution to this issue of ethics is the implementation of purely algorithmic evaluation of data. While it would still be critical to take into account the concerns of developers, part of the indignation mentioned with software like Hackystat to track developers was the transparency for other developers and managers to closely track and dissect an employee's lifestyle.

An algorithmic approach, in which data was collected & processed automatically, and presented in a non-offensive form would be a better compromise between the two systems. In this way we lose concerns over transparency but still gain valuable insight into employee productivity.

We can imagine a system in which managers receive a restricted view of employees data giving them an overview of their productivity, while the employee themselves can view the entire scope of the collected data, and improve aspects of their workflow based of that, without compromising freedom to their manager.

Algorithmic approaches have a massive advantage over system like the PSP for tracking employee data. They can format the data automatically into legible graph and charts the give us further insight in a digestible manner, they can also make projections based off current trends. And can handle most of the overhead in a way that non-algorithmic systems can't.

However a comprehensive algorithmic system is a complex thing, and more than like will be provided to a company by a third party, in which case one must be comfortable with handing over data to this third party, which again brings into question ethical issues.

Ultimately there is much to balance when looking to measure the software engineering process. The SEP is a vast and complex process that involves far more than just easily quantifiable metrics, there are many human factors on both employee and consumer ends that have to be considered. Generally, I believe it comes to a balance between increasing productivity and limiting overhead and infringement of employee rights.