SOFTWARE ENGINEERING MEASUREMENT REPORT

He Liu 17301050

Introduction:

There is no doubt that we are in the information age. Every day we contact with plenty of machines and interact with the software of them. Software engineering is playing a significant role in our society, it is quite different from other engineering. Because it's still a developing field at a high speed.

Unlike civil engineering or hydraulic engineering, they have already formed a lot of objective laws, but software engineering is not mature enough. To construct a building, all different construction teams must follow drawings and construction requirements. All the teams will build the same buildings according to the blueprints settled before, basically we can only measure them by their speed. However, building software is more like, Party A says, "I want a building that can lives 100 people." Then all the team will construct their own designed building, that means we need to measure them from all kinds of aspects.

Also building software is surely not a "stable" thing. When a building was just constructed, the function of it is to live people. After 10 years past, its function is still living people. But a software could be updated every day, it could be totally distinct from the original version.

"*The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software*." This is the definition of software engineering from /IEEE90/. "*As indicated by the word quantifiable in the above definition, at least one key professional society has acknowledged measurement as an integral part of a well-conceived software approach, and not merely an adjunct*." Said by Horst Zuse in 1997. With only about one hundred years history of software engineering and about 50 years history of software measure metrics, people are still seeking metrics for software engineering measurement.

In this report, I will briefly describe what data can be measured to improve engineers' performance, and some commonly used methods and concepts in the industry for software engineering measurement and some algorithmic approaches. Finally I will talk about the ethic issues behind them. But before jump to the theme, I'd like to list some history events of SE measurement.

History:

The groundwork for software measurement was established in the sixties and mainly in the seventies last century. Software specialists creating or inventing software measures was because they agreed that highly modularization and simple structure contributes to higher reliability of software systems.

- The earliest software measure is the Measure LOC (lines of code), earliest attempt made by Wolverton in 1974.
- In 1975, the term Software Physics was created by Kolence, and Halstead introduced the term software science in 1977.
- In the middle of 1970s, Measures of McCabe and of Halstead were created.
- In 1977, Laemmel and Shooman extended Zipf's Law to the programming world.
- In 1977 Gilb published a book which is one of the first books in the area of software measures.
- In 1977 IDSL and LC were proposed by Hecht, followed by a proposal of software complexity measurement by McClure in 1978.
- In 1978 Already published a paper about methods to measure programming quality and productivity.
- Albrecht introduced the Function-Point method in 1979.
- In 1980, Oviedo developed a Model of Program Quality, and Curtis published an important paper.
- In 1981 Ruston, Harrison, Troy proposed some important measures.

Other fundamental papers:
Weiser 1982   Emerson 1984   Longworth 1986   Thuss 1988   Ott 1989
Ott 1991   Patel 1992   Ott 1992   Rising 1992   Dhama 1994

Measurable data:

Now we want to consider about data. For measurable data, we need to think how could we use them, how to make them impact on performance measurement. But first, we need to think what are the characteristics of measurable data?

Measurable data must reflect software engineers' performance and abilities. So the question turns to be what aspects we want to see from software engineers.

Productivity	Productivity must be the most significant aspect of an engineer. To achieve a same goal, the one who got the correct answer first will always be better than others. High productivity means someone thinks fast, implement fast and make less mistake, since he needs less time to fix bugs.
Quality	Software Engineer should also care about the effectiveness of the code. Code must be meaningful and flexible.
Knowledge Base	Software Engineer should also care about the effectiveness of the code. Code must be meaningful and flexible.
Reliable	Someone is reliable means he can produce codes stable. Without peak and trough, he always writes out codes with high accuracy at normal speed.
Communication	Usually a software is made by a team, an engineer should be able to harmonious communicate with others.
Reflective ability	Everyone makes mistakes, but please do not always make the same mistake. A good software engineer should remember his mistakes, learn from the previous bugs and make it directly right next time.

After understood what make reflection to software engineers, certain measurable data could be discussed.

LOC (Lines Of Code): A very classic measure metric. It’s the most widely used techniques in cost estimation. It allows a simple comparison to data from many other projects, the historical ones included. It’s not good for an engineer counts too much to LOC, it increases the complexity and redundancy to the program. On the other hand, someone counts too little to LOC shows lack of engagement or maybe his code is too concise to read by others. So, LOC reflects to the term ‘reliable’, ’Productivity’, it shows whether an engineer has stable production or not. But this can’t show the quality of the code.

Commitments: Here ‘commitments’ doesn’t only mean commit to the git or github. But means every time an engineer or team uploads the code to the server. More attention should be paid on these aspects of commits.

- Commit frequency – this shows the reliability and productivity.
- Commit conflict – if someone always has conflicts with others in his commits, then it shows the problem of communication and may also show his lack of Reflective ability.
- Commit time – I believe that the precise time of someone’s commit shows his reliability. If he often commits his code close to the deadline, then he’s not a good time arranger.

Code coverage: A good engineer or team should exactly know what’s the goal, what should be written next and what has already written. Be always aware of what can current code do and what can’t do is a good ability. Before a developer commit to the team, tests should always be done. Giving all the simulations of possible inputs, thinking about what the correct outputs are. Using unit tests to find out bugs before finds them in the real-time running. Test coverage shows engineers’ knowledge base and reflective ability. They have the experience of imaging what might happen.

Comments: This topic can be separated into two parts:

- Comments to oneself – developers who like to write comments for themselves are more reliable and has better reflective ability. They are more engaged in the work and more efficient when look back. Others could be more easily to understand their code as well.
- Comments to others – developers who like to write comments when conflicts are happening or some other engineers facing troubles on their codes. They not only are good communicator but they also have larger knowledge base. This kind of people usually have potential leadership ability.

Code Dependencies: Are codes independent enough in the project? Though a software needs to be built step by step, but every part of the program should be modular, their heritance relationship shouldn’t be too complicated. Developers with large knowledge base knows all kinds of libraries but not rely on libraries. They should always have their own code logic.

Fix Rate: After issues and bugs were founded, how many of them have been fixed in time. This counts to a developer or a team’s reflective ability.

The discussion above related to a few aspects of measurable data, there are many more of them. Now let’s talk about some famous computational platforms.

Computational Platforms:

There are many computational platforms that could combine, refine, analyze the measurable data above and then present them back to the developers and managers.

I will introduce three typical ones, the first one CMMI & PSP & TSP, could improve the performance for all personal engineers, teams, and companies. Then OSSMETER is a really good tool for open source software for both personal engineers and teams. Finally, CAST is a company that provides complete solution to software companies and business.

CMMI & PSP & TSP

(Capability Maturity Model Integration)
(Personal Software Process)
(Team software process)

PSP:

The Personal Software Process (PSP) is a framework of techniques to help engineers improve their performance and their organizations as well. It's a Disciplined approach to measuring and analyzing their work.

PSP assists software engineers to:

- Improve their planning and estimating skills.
- Make commitments and schedules they can keep and meet.
- Reduce defects in their projects.
- Manage the quality of their plans.

TSP:

The team software process (TSP) provides a framework to create a team environment for establishing and maintaining a self-directed team. Supporting disciplined individual work as a base of PSP framework.

TSP assists software engineers to:

- Ensure quality software products
- Create secure software products
- Improve process management in an organization

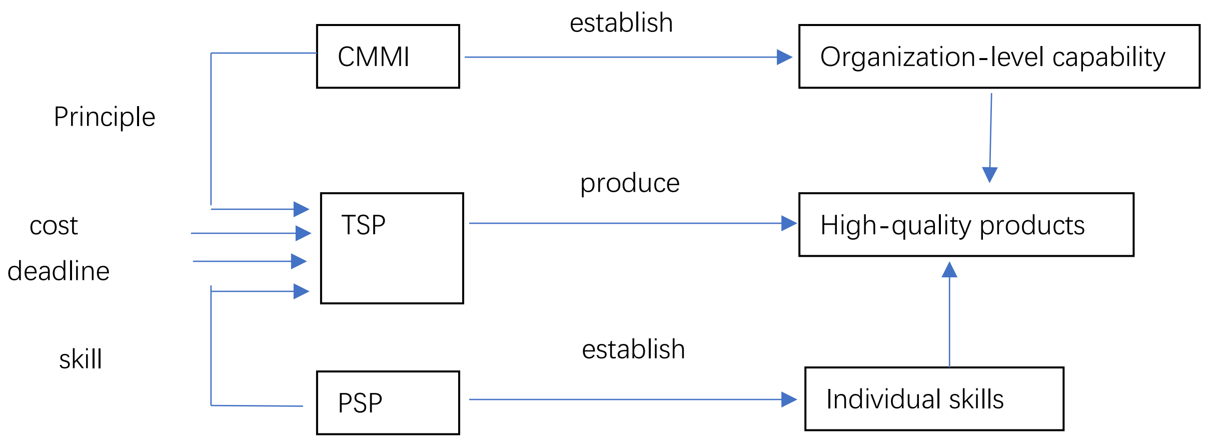
PSP skills are used in a TSP team environment

CMMI:

The CMMI framework is a reference model consisting of best practice descriptions for a broad range of engineering activities, covering the entire product life cycle from requirements definition through delivery and maintenance.

The TSP strongly supports the key practices of the CMMI, especially the project-level practices it targets. An organization can build or tailor its CMMI based process improvement effort around the consistent, high-maturity operational framework provided by the TSP.

CMMI/PSP/TSP for small businesses (graph blow):



Here are more than 10,000 of organizations using CMMI from over 106 countries. Also, more than 130,000 people have learnt CMMI.

OSSMETER:

(Automated Measurement and Analysis of Open Source Software)

Purpose: Deciding whether an open source software (OSS) meets the required standards for adoption in terms of quality, maturity, activity of development.

Features:

- Involves exploring various sources of information including its source code repositories to identify how actively the code is developed, which programming languages are used, how well the code is commented, whether there are unit tests etc.
- Bug tracking system to identify whether the software has many open bugs and at which rate bugs are fixed.
- Listing the number of downloads, the license(s) under which it is made available, its release history etc.
- It Use novel contributions on language-agnostic and language-specific methods for source code analysis, combined with Natural Language Processing (NLP) and text mining techniques such as question/answer extraction, sentiment analysis and thread clustering to analyze and integrate relevant information that extracted from communication

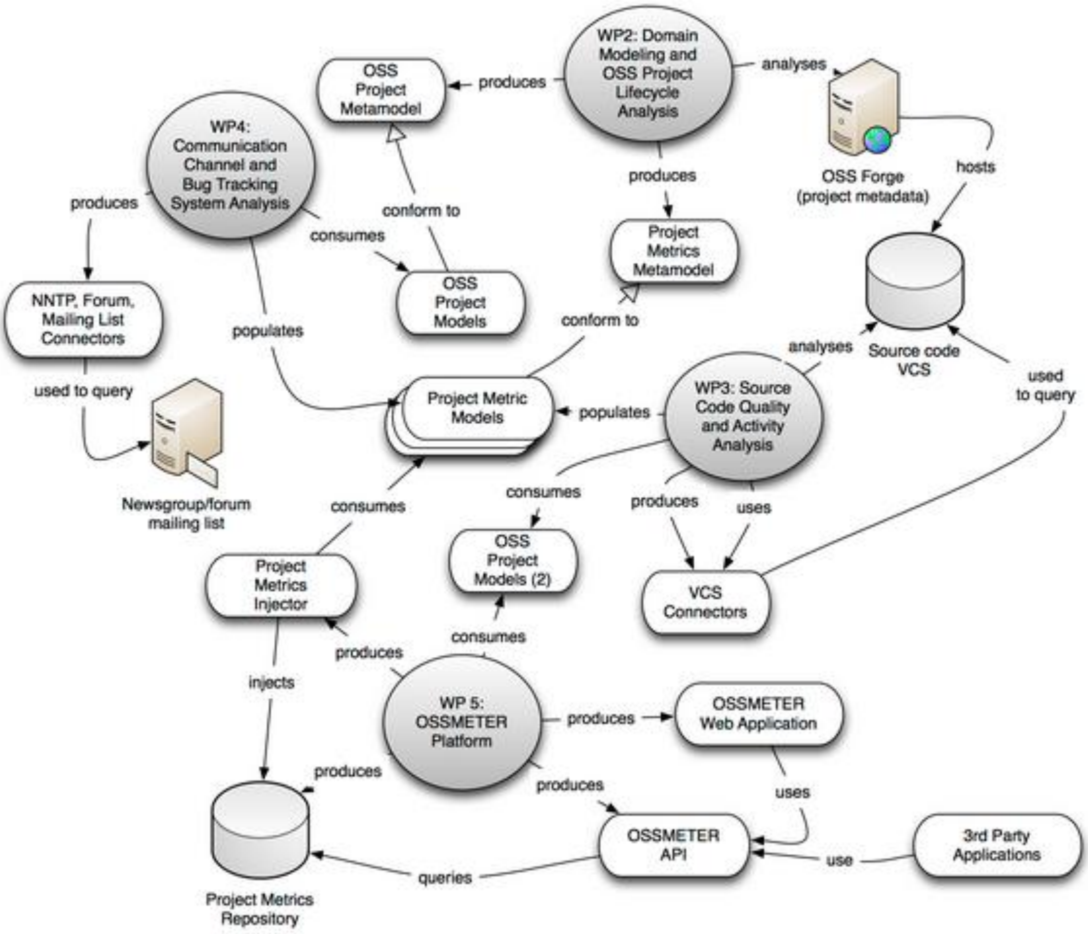
channels.

- Users can register, discover and compare OSS projects through a cloud-based platform.

Supporting for:

- Developers and managers who are responsible for deciding on the adoption of OSS, as it will enable them to make decisions on difficult situations and uniform quality indicators;
- Developers of OSS as it will enable them to monitor the quality of the OSS projects they contribute to, promote the OSS they contribute to using independently calculated quality indicators, and recognize related projects for establishing synergies;
- Investigators for the software which produce OSS, are always allowed to monitor the quality and assess the impact of the produced software even after the end of the projects.

The graph blow shows how OSSMETER works:



CAST AIP(Application Intelligence Platform)**:**

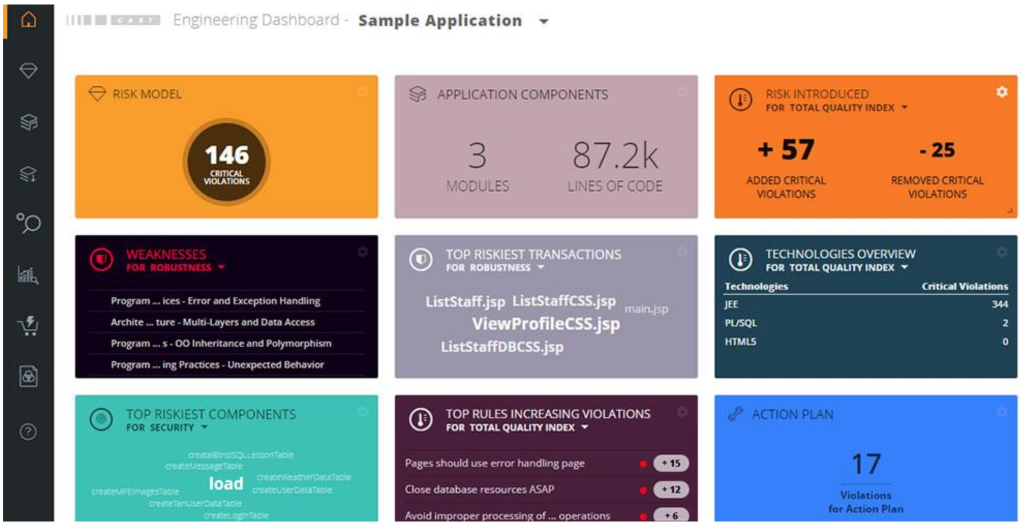
Introduction:

CAST AIP is an automated solution for completing the software measurement process. It can judge multiple languages in a complex infrastructure. Regardless what technologies an application uses, AIP can break down the app at the source-code level to see the quality, security, productivity, and many other factors that based on the number of business functions it is supposed to accomplish. It is a management solution to make certain projects remain within scope and to obtain desired code quality levels.

With automatic software measurement process, companies can get accurate, objective and repeatable results whenever they want. The base score functions as a continuous measurement for monitoring every aspect of a project. If a organization has problem with mitigating risk, improving code quality, decreasing technical debt, or receiving the desired amount of productivity, then CAST AIP can be a good helper in the software measurement process, ensuring the organization is working more effectively throughout the development life cycle.

Key Features:

- Ability to quickly identify multi-component software flaws.
- Ability to validate against industry standard rules.
- Drill-down to identify specific code location of critical flaws.
- Architectural flaws inside the riskiest objects and transactions.
- Insight into why specific defects are occurring and guidelines for fast remediation.
- Action plans based on optimized list of high impact issues.
- At-a-glance summary of your software health in terms of robustness, security, efficiency, changeability, transferability and quality.
- Accurate sizing metrics that show what you need to manage and how far you've gotten
- Heatmap to quickly find where you are at highest risk
- Trending analysis to benchmark performance over time
- Fast facts on the evolution of your software



Program ... ices - Error and Exception Handling
Archite ... ture - Multi-Layers and Data Access
Program ... s - OO Inheritance and Polymorphism
Program ... ing Practices - Unexpected Behavior

Technologies	Critical Violations
JEE	344
PL/SQL	2
HTML5	0

Pages should use error handling page	+ 15
Close database resources ASAP	+ 12
Avoid improper processing of ... operations	+ 6

The graph above shows the engineering dashboard from CAST AIP:

Algorithmic approaches:

How to transform measurable data to some conclusions? Certain algorithms must be used. Here I will introduce two interesting approaches.

McCabe:

McCabe's cyclomatic complexity was developed by Thomas J. McCabe, Sr. in 1976. It’s a software quality metric that quantifies the complexity of a software program. Complexity is inferred by measuring the number of linearly independent paths through the program. The higher the number the more complex the code.
The higher the McCabe number (e.g. > 10) is, the more likely the program is difficult to understand and has a higher probability of containing defects. The cyclomatic complexity number also indicates the number of test cases that would have to be written to execute all paths in a program.
Calculation of CC:
Cyclomatic complexity (CC) = E - N + 2P
P = number of disconnected parts of the flow graph (e.g. a calling program and a subroutine)
E = number of edges (transfers of control)
N = number of nodes (sequential group of statements containing only one transfer of control)

After knowing the complexity of the current program, adjustments could be made during the process of the project.

A Multi-objective Genetic Algorithm for Software Development Team Staffing Based on Personality Types:

This is not a complete theory, but it’s interesting.

Constantinos Stylianou and Andreas S. Andreou designed an algorithm to focus on optimizing human resource usage based on technical skills and personality traits of software developers. The goal of the proposed approach is to allow project managers in the industry to staff their projects with the most suitable teams to take account of the technical knowledge and skills of available developers as well as their personality traits and abilities.

Background: The Five-Factor Model

- **Neuroticism**: reflects the level to which an individual is predisposed to experiencing negative emotions, such as sadness, embarrassment, fear and anger.
- **Extraversion**: refers to the level to which an individual engages with their external world through interpersonal interactions, as well as their energy and predisposition to experiencing positive emotions.
- **Openness**: to experience concerns an individual’s tendencies regarding intellectual curiosity, creativity and variety in interests and experiences.
- **Agreeableness**: involves interpersonal orientation with regards issues, such as compassion, social harmony, cooperation, and trust.
- **Conscientiousness**: relates to the degree of self-discipline and control of impulses, and also ambition and organization.

The desired level of each of the domain was determined so as to ascertain whether a profession requires either a {1:low, 2:medium or 3:high} level of that particular domain.

Encoding and Representation:
Each software project consists of tasks that need to be carried out. For each task, developers need to be assigned to perform the activities involved. Therefore, since it is the selection of developers that forms the basis of evaluation, each project task is denoted by a string of bits, and each bit represents one specific developer. If a bit in the string has a value of ‘1’, then this signifies that the specific developer is assigned to work on the task, whereas a value of ‘0’ indicates that the developer has not been selected for the task. Overall, if a software project consists of T tasks and there are E available developers, then each solution would be represented by an individual in the algorithm using (TxE)bits.

Each developer is simply required to be rated based on each of the skills required by task activities in a normalized form in the range [0, 1], meaning that low possession of skill will be denoted by a value closer to zero, and high possession of a skill will be denoted by a value closer to one. P

Then they provided 3 functions and 2 constraint

Technical Skills Objective Function(f1)

$$f_1 = \max(skill\_level\_of\_assigned\_developers)\ +$$
$$+avg(skill\_level\_of\_assigned\_developers)$$

Personality Traits Objective Function (f2)

$$f_2 = \sum_{i=1}^{5} \left| desired\_level\_in\_domain_i - developer\_level\_in\_domain_i \right|$$

Team Size Objective Function(f3)

$$f_3 = \frac{1}{number\_of\_assigned\_developers}$$

Skills Satisfied Constraint(c1)

$$c_1 = \frac{number\_of\_unsatisfied\_skills}{total\_number\_of\_project\_skills\_required}$$

Developer Availability Constraint

$$c_2 = \frac{number\_of\_days\_assigned\_with\_conflicts}{total\_number\_of\_days\_assigned\_in\_the\_project}$$

Experiment

They set and observed two hypothetical software projects consisting of 20 and 30 tasks, each were created based on the input of several project managers of SME software development companies as to the basic structure, size and complexity of the type of software projects they usually undertake. The skill levels and personality traits of the available developers were selected so as to represent the best-case and worst-case scenarios for the proposed team staffing approach. For the best-case scenario, all available developers possessing the highest skill levels also possessed the most suitable personality traits. For the worst-case scenario, all the available developers possessing the highest skill levels possessed the least suitable personality traits. Through these two extreme cases, both the behavior and correctness of the optimization approach could be observed and analyzed, also the competitive nature of the objective functions could be investigated.

Result:

The algorithm managed to provide both feasible and optimal solutions when performing team staffing in the best-case scenario. Specifically, the algorithm always managed to assign the most suitable developer with respect to both the technical skill levels and personality traits possessed, and never assigned a developer who was less suited in either aspect. In the worst-case scenario, the algorithm's job was to try to balance the two objective functions, since no developers possessed both the highest skill levels and most suitable personality traits for any task. it was observed that this time the individuals of the Pareto front represented different solutions, as seen in Fig. 2. Such behavior again was anticipated since for each task either skill levels or personality traits could be given preference – but not both due to the nature of the characteristics of the available developers.

The general behavior of the algorithm was, thus, validated as correct.

Table 1. Results obtained from the first experiment using hypothetical software projects

Experiment	Average Number of Unique Solutions	Execution Time (min.)
Best-case scenario (20 tasks)	1	22.56
Worst-case scenario (20 tasks)	94	22.86
Best-case scenario (30 tasks)	1	23.90
Worst-case scenario (30 tasks)	95	26.08

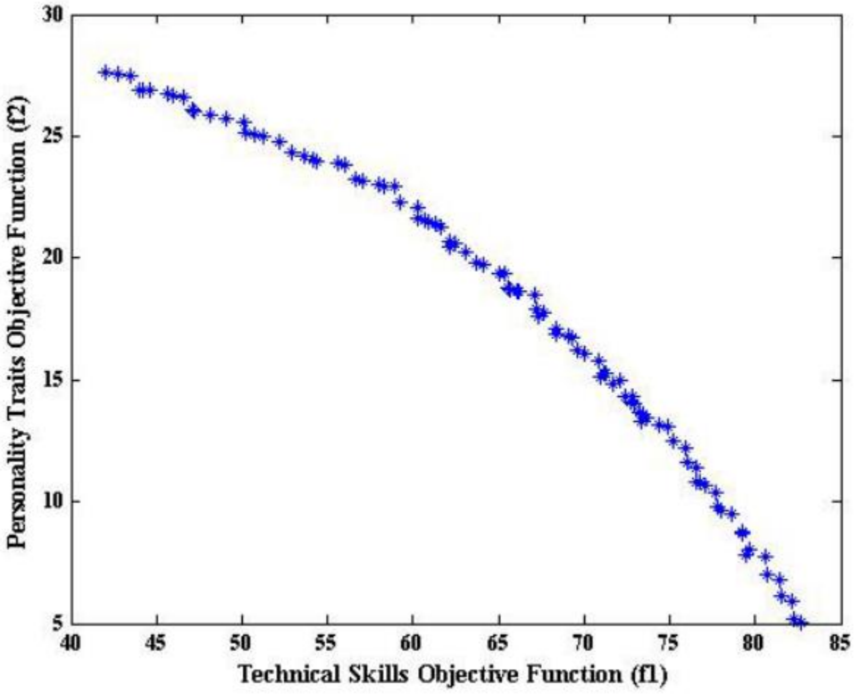


Fig. 2. Pareto front of hypothetical project (30 tasks) worst-case scenario (f_1 vs. f_2)

Ethics:

When measuring the software engineering, engineers are being measured at the same time. Though experience and personal status should be shared in the team, it’s still reasonable to ask, are all the developers happy about the measurement methods that companies ask them to use?

It is normal for the company to monitor and evaluate developers. I think it is reasonable for the company to collect and use the developers’ data to improve the quality of the work and make a reasonable analysis and feedback to the developers. The company always wants to maximize the value of each developer. However, not every employee is willing to give full play to their abilities, maybe they want to save their strength, maybe they are not willing to strive for the upper reaches. At this time, the concern and help from the company may be considered by the developer to be stressful. As mentioned above, managers should be concerned about commits situations of members in the team. Managers should train developers to let them arrange their time properly, not to rush just before deadline. But this may be perceived by the programmer as increasing the pressure on him, controlling his schedule, or even thinking that the manager is trying to add new jobs to him.

Like the TSP and CMMI I introduced before, I think they are safer and more reliable. Because they are just working modes, they just tell the developers what to do, not keep watching on developers, then give suggestions. And if the company wants to use TSP or CMMI, it first needs employees to learn PSP. This shows that it is beneficial to employees and allows employees to increase their skills.

But like the smarter and more complex platform like CAST I've mentioned, the opacity is significantly higher. Because as a developer, it’s very difficult to know what data it has obtained. Usually developers can only access its feedback, but do not know its inner principle. As far as I know, some technology companies will ask developers to run some software at the time of their work to improve and help their communication with the team and improve their work efficiency as well. But recently, South Korea’s social media company Kakao has been accused of handing over complaints from Samsung employees to Samsung and other companies, which is obviously unethical. Then, when developers use those company’s required measurement application, how to ensure and supervise that the software does not collect information about privacy of employees, this is a question worth to be considered. I think it would be an ideal situation if employees could submit data on their own initiative instead of collecting it automatically.

Conclusion:

How to measure software engineering is still a complex and worth exploring questions. Software engineering itself is an emerging field, and it is even harder to measure it and find metrics. How to balance the ease of use of the measurement method and the violation of employees is also a very difficult topic.

References:

A framework of Software Measurement(1997) by Horst Zuse pp.21,59-61

Measuring LOC and other basic measurement(2002) by Emanuel Weiss pp.3-5,8

Software Process Improvement for Small Organizations Based on CMMI/TSP/PS(2010) by Lina Zhang

A Multi-objective Genetic Algorithm for Software Development Team Staffing Based on Personality Types(2012) by Constantinos Stylianou and Andreas S. Andreou

https://techbeacon.com/app-dev-testing/9-metrics-can-make-difference-todays-software-development-teams
http://www.chambers.com.au/glossary/mc_cabe_cyclomatic_complexity.php
https://hackernoon.com/measure-a-developers-impact-e2e18593ac79
https://ieeexplore.ieee.org/abstract/document/493023
https://explainagile.com/agile/personal-software-process/
http://www.ossmeter.org/overview