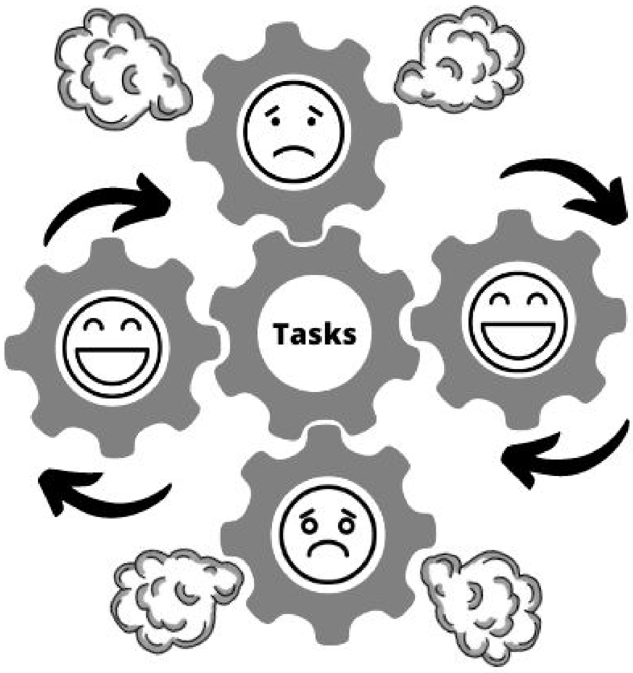
A **cognisant & considerative** approach for measuring software engineering processes

By James Rowland (18324013)

Content :



Fundamentals :

Software metrics are an integral part of modern software engineering. More and more customers are specifying software and/or quality metrics reporting as part of their contractual requirements. Companies are using metrics to better understand, track, control and predict software projects, processes and products. The term software metrics means different things to different people but since we are focusing on measuring the software engineering process specifically, I will have a more refined approach to this topic. If a metric is to provide useful information, everyone involved in selecting, designing, implementing, collecting, and utilizing it must understand its definition and purpose. We need to take into account numerous different factors from project cost and effort prediction and modeling, to defect tracking and root cause analysis, to a specific test coverage metric, to computer performance modeling…… to derive useful information.

Identifying entities is the first step in setting up our assessment pipelines. Assessments are used extensively in most areas of production to estimate costs, detect problems, assess quality, and monitor progression. To measure, we must first determine the entity. A software engineering example could be selecting our hypothetical API endpoint as our entity. Once we select an entity, we must select the attribute of that entity that we want to describe. For this example, the speed of processing queries and the speed at sending responses could be two attributes of our entity. Finally, we must have a defined and accepted mapping system. It is meaningless to say that this API's processing speed is 15 or its response time is 10 unless we know that we are talking about queries per second and responses per second, respectively. For successful measurement we need to make sure these three aspects are defined properly. If we fail to calibrate these three aspects of measurement we could be undermining our whole process of assessing our engineering processes as we wont be able to find the root of the problem.

Model of measurement maps the basic building blocks of the process model of input - process - output to software entities which we can discuss. Software entities of the input type include all of the resources used for software research, development, and production. Software entities of the process type include software-related activities and events and are usually associated with a time factor. Examples of process entities include defined activities such as developing a software system from requirements through delivery to the customer, the inspection of a piece of code, or the first 6 months of operations after delivery. We might want to measure the complexity, size, modularity, testability, usability, reliability, or maintainability of a piece of source code. Software entities of the output type are the products of the software process. Examples of software output entities include requirements documentation, design specifications, code, test documentation (plans, scripts, specifications, cases, reports), project plans, status reports, budgets, problem reports, and software metrics. Now that we have a good understanding of the fundamental entities that make up software engineering process we need to understand the fundamental measurement of processes available. This is what measurement function analysis is. The measurement function defines how we are going to calculate the metric. Some metrics are modeled using mathematical combinations (e.g., equations or algorithms) of base measures or other derived measures. An example of a derived measure would be the inspection's preparation rate, modeled as the number of lines of code reviewed divided by the number of preparation hours. Many measurement models include an element of simplification. This is both the strength and the weakness of using modeling. When we create a model to use as our measurement function, we need to be pragmatic. If we try to include all of the elements that affect the attribute or characterize the entity, our model can become so complicated that it's useless. Being pragmatic means not trying to create the most comprehensive model. It means picking the aspects that are the most important

Priorities & Assessment :

Static vs Dynamic assessment is the two traditional classifications of assessing software metrics. The first says collect data on everything and then analyze the data to find correlation, meaning or information. This method usually focuses on building large histories of data and takes a more rigid approach to assessing trends. The second school of thought was what I call the shotgun method of metrics. This usually involved collecting and reporting on whatever the current "hot" metrics were or using whatever data was available as a byproduct of software development to produce metrics. This “hot” method focuses on analysing what is going on right now and so that we can best determine the state of the software engineering process now. Dynamic assessment is a modern, more-nuanced approach to assessment. Dynamic assessment says “We need to measure what's going on right now, if what's going on now changes, our assessment changes”. Its a more forgiving and flexible approach. With this approach we would set our goals and standards with an agile approach. We would focus on the next week to two weeks to determine what needs to be done, and from that we would determine how we are going to determine how effective we are being. With this approach we have to allocate more time on determining new goals and organising new measurement protocols. With a static approach we take a more direct line towards our goals. This approach has a linear progression and as a result, is much much easier to measure and assess. We can also set up permanent testing mechanisms with this approach, and can collate data and determine standards with this approach as this approach doesn't tolerate change. The power of this approach is that it is easier, less expensive for resources and more reliable as we can build a database of analytics which we can refer back to.

There are serious problems with both of these methods. The problem with the static method is that if we consider all of the possible software entities and all of their possible attributes that can be measured, there are just too many measures. It would be easy to drown an organization in the enormity of the task of trying to measure everything, keep track of everything and try to predict downfalls of every aspect of our organization. One of my favorite quotes talks about “spending all of our time reporting on the nothing we are doing because we are spending all of our time reporting.” The problem with the second method can be illustrated in Watts Humphrey’s quote, "There are so many possible measures in a complex software process that some random selection of metrics will not likely turn up anything of value.". It is recommended we keep our methods of measuring and assessing software engineering open to change so that we can optimise our approach.

Modern Assessment consists of a fundamental shift in the philosophy of software metrics. Software metrics programs are now being designed to provide the specific information necessary to manage software projects and improve software engineering processes and services. Organizational, project and task goals are determined in advance and then metrics are selected based on those goals. These metrics are used to determine our effectiveness in meeting our goals. With this approach we are only measuring what we need information about. The foundation of this approach is aimed at making practitioners ask not so much "What should I measure?" but "Why am I measuring?" or "What business needs does the organization wish its measurement initiative to address?".

CASE STUDY :

I think it's interesting to compare the two most well known software engineering procedures as they both have their own individual way of measuring and assessing the quality of the software engineering processes that they cause. This will form the methods which are chosen to assess and measure the developer.

Agile assessment for software engineering implicates that the requirements of the client and the development team develop gradually. In this approach, the development and testing processes are aligned with the demands of the customer. The **benefits of Agile Testing include** : Minimal planning required due to the simple structure, Little documentation, Testers work jointly with developers on the project, Best fit for small projects. It also works fine for long-term, Bugs don't pile up. It takes place simultaneously with the project development, Quick issue solving. There is a separate testing place for every iteration. Continuous feedback. The Team and the Product Owner are always on the same page about the project development and quality assurance. Test-driven development. When using Agile, testing is performed during implementation.Flexible and supports changes. **Shortcomings of Agile Testing:** Requires continuous efficient communication between testers and developers. Blurred organizational structure. Only senior programmers can make crucial decisions during team meetings.

Waterfall assessment for software engineering

Waterfall testing is a software testing methodology that matches the postulates of Waterfall software development. Waterfall implicates that there is a sequence of stages in which the output of each stage becomes the input for the next. Waterfall approach is also known as the linear-sequential life cycle model. As you can probably already tell, agile testing is applied together with the agile development approach. Consequently, Waterfall testing is applied in the scope of the Waterfall development. **Advantages of Waterfall Testing** Well-structured and well-documented testing process. Fits projects of any complexity. All the features of the project are developed and delivered together. Testers and developers do not need to communicate continuously. They work independently. Easy to manage. With Waterfall, every stage has its deliverables and a strict review method. Easy to adapt in case a team changes or different teams work in shifts. **Drawbacks of Waterfall Testing** It requires excessive documentation. Strict structure. Not perfect for large-scale projects. Testing takes part only at the end of the project In some cases, bugs are easier solved before they pile up. In case the client requirements are not precise from the start, it is less effective compared to Agile.

Metrics used for determining effectiveness :

Identifying the metrics consumer is the first step in measuring and assessing the software engineering process. If we don't have our metrics pipelined suitably they are useless. Metric consumers can include : functional management, project management, software engineers / programmers , test managers / testers, specialists and/or customers / users. If a metric does not have a customer, it should not be produced. Metrics are expensive to collect, report, and analyze so if no one is using a metric, producing it is a waste of time and money. The customers' information requirements should always drive the metrics program. Otherwise, we may end up with a product without a market and with a program that wastes time and money. By recognizing potential customers and involving those customers early in the metric definition effort, the chances of success are greatly increased.

Reasons behind priority of metrics will, in the end determine the direction our development process goes. We have to choose correctly so that we meet both internal and external goals. It’s hard to get blisteringly fast development and expect to have no problems with code standards. Its difficult to get a completely polished software system that is water tight and a complete product within a short deadline. There's' trade offs we need to be aware of and important decisions are what metrics and going to be our priority for determining effective engineering.To measure productivity managers can use two types of metrics: Size-related metrics indicating the size of outcomes from an activity. For instance, the lines of written source code. Function-related metrics represent the amount of useful functionality shipped during a set period of time. Function points and application points are the most commonly used metrics for waterfall software development, while story points are the usual metrics for agile projects. The productivity metrics you choose to track should be consistent so use clear definitions so the number has meaning, auditable so outsiders can prove the viability of the measures, available so we can use these for benchmarking and repeatable so different groups of users will essentially get the same result. Instead of exploring individual metrics, I'm going to take a more holistic approach. Let's explore different groupings of metrics and see what works and what does not work. The reason behind this is theres no point in discussing one metric at a time as we wont be able to explore how these metrics can work with each other.

Primary developer metrics are what we need to focus on now. The first type of process is **formal code metrics**. This is a basic form of metric which needs to be taken with caution. Measuring the effectiveness of a developer simply by the number of lines of code they wrote/edited isnt a clear sign efficiency. We can however derive some simple views about the state of a developers software from these metrics, but they shouldn't be a deciding factor in measuring and assessing software engineering. This is definitely useful information in this kind of metric that we can use, but it shouldt be relied on for decisions. We can use github analytics and progression tests for this kind of metrics. We can see how much code is being rewritten and use the amount of time being consumed by a task.

Developer productivity metrics—Such as active days, assignment scope, efficiency, code churn. These metrics can help you understand how much time and work developers are investing in a software project. This kind of metric is important and is a more fundamental / base type of metric. This metric is important for both management and the developers. We need to pay close attention to these kinds of metrics as it allows us to be cognisant of what has been done, what is being done, and what is left to do. We can use github for collating data related to this kind of metric. I also think its interesting to use general data like log-in time, break time, how much time is really spent for development, how much time is being spent researching……. The making us aware of trends / whats occuring in the environment. Im not going to dive too deep into measuring if a engineer is actually developing or if they're playing candy crush on their phone. It would deprive us of more time to talk about more interesting types of measurement. With these kinds of metrics we can deduce current bottlenecks in progression. If we see that a large proportion of time is being spent on research for section A, while production is flying ahead on section B, we probably need to come together and discuss if we need to partition our resources differently to help section A catch up. These kinds of metrics help estimate the progression towards the final product too. If we think about what's going on in the work from home world due to coronavirus, we can imagine a whole new world of problems. Lets say that, pre coronavirus we were going to complete our project before our due dates. Now that our teams are working from home we see that the network team have lagged behind significantly, while the frontend team are finished. We can deduce that something about working from home has corrupted the network team and we can investigate further and deduce that their testing measures are too slow for working from home. Hopefully i painted a reasonable picture on how we can analyse this type of data for measuring and assessing software development.

Agile process metrics—Such as lead time, cycle time and velocity/throughput and spirit burndown. They measure the progress of a dev team in producing working, shipping-quality software features. We cant just measure what code is being written to determine the quality of software engineering occuring in our team. We also need to assess our goals and how we are handling these goals. Are we being realistic? Are we going to burn out our team? Is our current cycle time too short or is it taking too long to publish iterations? These are the kind of questions that these metrics hope to shed light on. Its too expensive to get wrong so collecting metrics and analysing this process is extremely important. This metric allows you to estimate how fast you can deliver new features to users. It's also another way to understand your team's current speed for different tasks by breaking the total throughput down to median time by status or issue type. You can pin down the exact bottlenecks affecting the team's performance and set more accurate expectations. Throughput indicates the total value-added work output by the team. It is typically represented by the units of work (tickets) the team has completed within a set period of time. You should align your throughput metric with your current business goals. If your goal is to release new bug-free modules in this sprint, you should see a large fraction of defect tickets being resolved and so on. Sprint Burndown is one of the key metrics for agile scrum. A burndown report communicates the complexion of work throughout the sprint based on story points. The goal of the team is to consistently deliver all work, according to the forecast. By tracking this metric you can obtain important insights: Consistent early sprint finishes can signify lack of scheduled work for one sprint. Consistently missed sprint deadlines, on the contrary, can indicate a gap in your planning and the fact that your team is asked to deliver too much work. Your report should feature a steep reduction in "remaining values", rather than a dramatic drop as the latter will indicate that the work was not assigned in granular pieces.

Operational metrics—Such as Mean Time Between Failures and Mean Time to Recover. This checks how software is running in production and how effective operations staff are at maintaining it. Failures are a part of software engineering so measuring these types of periods is extremely important. It can highlight weak links in the software engineering process. “If debugging is the process of removing bugs, coding is the process of writing bugs” is something that very true, testing and handling failures is a part of software engineering so we need to strengthen this by investigating this process with the same depth we investigate software development. Theres no point in creating software systems at all if theres no backup ready and we dont know how long recovery takes. Google is a perfect example of something that takes operational metrics important. Google employs a team of people called Site Reliability Engineers (SREs) whose main focus is to keep Google search and other services running. The team runs a simulated war on Google’s infrastructure that they call DiRT (disaster recovery testing). This “war” involves everything from causing leaks in water pipes to staging protests to attempting to steal disks from the servers—whatever it takes to bring down the infrastructure. The data center attacks aren’t real, but they are hard to distinguish from an actual event. The SRE Managers explains that they have “become braver in how much we’re willing to disrupt in order to make sure everything works.” Krishan explains that her role is “to come up with big tests that really expose weaknesses.” Through the information they gain from a fake attack, they know what is working, and what needs improvement.

Test metrics—Such as code coverage, percent of automated tests, and defects in production. This measures how comprehensively a system is tested, which should be correlated with software quality. In the end it is the team / management who decides the priority of the test metrics. In my experience test metrics are the easiest to understand as we come from an engineering background. Test metrics are usually to the point and require little energy to derive useful information. We can set up automated testing servers as well as backup servers that can be used if we detect a spike in failing tests.

Customer satisfaction—Such as Net Promoter Score, Customer Effort Score and Customer Satisfaction Score. The ultimate measurement of how customers experience the software and their interaction with the software vendor. There are plenty of examples of bad software that is extremely popular. The Windows OS is often criticized because of numerous weaknesses the system has yet it accounts for 72.98 percent share of the desktop, tablet, and console OS market. Customer satisfaction at the end of the day is what makes or breaks software so we need to give attention to this side of software engineering.

Collating metrics :

Why should we collate metrics

The goal of tracking and analyzing software metrics is to determine the quality of the current product or process, improve that quality and predict the quality once the software development project is complete. On a more granular level, software development managers are trying to: Increase return on investment (ROI), Identify areas of improvement, Manage workloads, Reduce overtime, Reduce costs. These goals can be achieved by providing information and clarity throughout the organization about complex software development projects. Metrics are an important component of quality assurance, management, debugging, performance, and estimating costs, and they’re valuable for both developers and development team leaders. Often sets of software metrics are communicated to software development teams as goals. So the focus becomes: Reducing the lines of codes. Reducing the number of bugs reported. Increasing the number of software iterations Speeding up the completion of tasks.

How to collate metrics

Make the software metrics work for the software development team so that it can work better. Measuring and analyzing doesn't have to be burdensome or something that gets in the way of creating code. Software metrics should have several important characteristics. They should be: Simple and computable. Consistent and unambiguous (objective). Use consistent units of measurement. Independent of programming languages. Easy to calibrate and adaptable. Easy and cost-effective to obtain. Able to be validated for accuracy and reliability. Relevant to the development of high-quality software products But software development teams and management run the risk of having too much data and not enough emphasis on the software metrics that help deliver useful software to customers**.**

Platforms we can use mainly consist of a time tracking system, chart task completion (often optimized for either agile, waterfall or whatever method of development they are choosing), digital activity monitoring software, productivity quantifying software and group meetings or check-ins. Of course due to covid-19 the ways in which we gather information has been transformed due to new priorities. These softwares analyse during working hours to predict trends and prevent problems. Github is also a popular yet limited platform for tracking efficiency. Instead it tracks general activity. We can extend our view on what is going on by extending the base of metrics we collect. If we want to collect health data, companies have been shown to give fitbits and apple watches that stream data back to algorithms that can detect stress levels and unhealthy habits. We can also collate market metrics to detect sways in the market so that we can get an early jump on trends we can analyse the markets. The platforms you use are derived from the kind of problems you want to solve.

When to collate metrics

A popular saying is to track trends not numbers so we need to make an educated selection of metrics to use. The simple answer when should we collate metrics is always. The ethical question will be posed below, but the benefits of collating metrics is too strong not to measure them. The question should be where should we collate metrics. The simple also Software metrics are very seductive to management because complex processes are represented as simple numbers. And those numbers are easy to compare to other numbers. So when a software metric target is met, it is easy to declare success. Not reaching that number lets software development teams know they need to work more on reaching that target. These simple targets do not offer as much information on how the software metrics are trending. Any single data point is not as significant as the trend it is part of. Analysis of why the trend line is moving in a certain direction or at what rate it is moving will say more about the process. Trends also will show what effect any process changes have on progress. The psychological effects of observing a trend – similar to the Hawthorne Effect, or changes in behavior resulting from awareness of being observed – can be greater than focusing on a single measurement. If the target is not met, that, unfortunately, can be seen as a failure. But a trend line showing progress toward a target offers incentive and insight into how to reach that target.

Algorithmic analysis of human work & Using AI is interesting and can boost the performance of our teams. Aspects of software quality that we take into consideration and analyse are reliability, maintainability, testability, portability, reusability and security. We can set up functional, unit and system tests that will automatically assess any changes made to the code base. A simple formula we can use is productivity = function points implemented / person months where. The function point is a "unit of measurement" to express the amount of functionality provided by a team/developer. This can be an attractive measurement of productivity because we are measuring the return on time invested into development. It is also a realistic base of measurement because it keeps the end user / customer in mind too. We can all feed this kind of measurement data into an AI machine to significantly reduce the amount of time engineers spend on debugging, while also speeding up the process of rolling out new software. An example of this is facebook's SapFix AI hybrid tool. SapFix automatically generates fixes for specific bugs, and then proposes them to engineers for approval and deployment to production. SapFix has been used to accelerate the process of shipping robust, stable code updates to millions of devices using the Facebook Android app. It’s the first such use of AI-powered testing and debugging tools in production at this scale. The tool generates multiple potential fixes per bug and then evaluates their quality by checking for three issues: Are there compilation errors, does the crash persist, and does the fix introduce new crashes? We can see how this kind of technology can make software engineering assessment and measurement much more efficient and its applications will make software faster and more responsive. We can also find out subtle things through AI like measuring someone’s job satisfaction through facial recognition, or parse a Facebook post to detect signs of depression, or even determine someone’s heart rate through a webcam.

Using metrics

Ethics :

Good vs Bad ethics is the final point we are going to discuss and is something that has a strong variance in opinions. Personally I don't see anything wrong with considerate and cognisant measurement of work. If we didn't keep an eye on the tasks being done we would have a weaker control on development. An interesting analogy is that monitoring is for symptom base alerting. There is, however, a line that must not be crossed. If we are to measure the efficiency of software engineering we must do so with respect. Consent is a fundamental pillar to measuring software engineers. You can expect software engineers to be productive if they feel like they are constantly being watched, and their every move is being fed into an algorithm. That will instead foster an environment that hinders progression as we become afraid of proposing radical changes.

We need to pay attention to the fundamentals and get them right first. I personally find it hilarious that struggling companies choose meaningless metrics to measure their software engineers effectiveness. Instead of measuring their test metrics, agile process metrics and organizational metrics, they measure the screen time of the developer and persecute developers who don't spend X amount of hours coding, even if they have already gotten their jobs finished. In short, once the developer is aware of what is being assessed about them,how these metrics will be used and aren't upset about the collating of their data, there shouldn't be any problems. Metrics should also be used to improve and not punish our resources.

Modern view on ethics is the last point we will discuss. We are seeing a rapid expanse in the type of matrices people are considering. This can be both good and bad. For example some employers are now tracking the health of software engineers so that HR can take this information into account when making decisions. Just as HR professionals must always think about people first in the face of metrics, they must think of all their employees when creating wellness programs and initiatives. What is best for the entire workforce? Although metrics can tell us a lot, they can't capture the complete picture of your employees as people. Wellness initiatives themselves can raise compliance issues as well. The big question is whether employers can require participation in certain wellness programs. I understand tracking health is something that is very easy and useful in 2020, but we still need to be clear on the extent that we use this tracking. We can see the australian government making this issue a serious offence by extending the maximum jail term which will increase from two to five years, the maximum fine for individuals will jump from AU$126,000 to AU$315,000, and private health insurers will not be able to access health or de-identified data. This means that in australia companies that choose to track health data can't fire, bully, or treat workers differently based on the health data they collect. Its good to see this type of proactive defense against discrimination, and if we are to see the extent of measurement on software engineers grow we need to see them protected by these kinds of laws

References :

https://www.lexology.com/library/detail.aspx?g=8c1d8682-2492-41db-9468-45e1ec09f491
https://www.tsheets.com/gps-survey
https://techbeacon.com/app-dev-testing/9-metrics-can-make-difference-todays-software-development-teams
https://www.sealights.io/software-development-metrics/top-5-software-metrics-to-manage-development-projects-effectively/
https://stackify.com/track-software-metrics/