Metrics & Measurement in Software Development

Index

i. Abstract

Research around software measurement can be thought about as fitting under 5 conceptual methodological issues: What is measurement, how to apply measurement theory to software, what tools do we have at our disposal to measure, how to algorithmically collect core measures, and lastly how to analyze these measures in terms of efficiency and ethics. Academic research is imperative in these areas for any company, organization or institution that wishes to understand and develop practices that produce the most efficient software processes to analyze successful products and services. In this essay the concept of measurement and what it means specifically for the software engineering process will be explored, the specific computational platforms and algorithmic approaches used to collect these metrics will be outlined and explained, and lastly the ethical concerns surrounding big data and cyber security in the 21st century in terms of measuring software will be analyzed. In a world where GDPR compliance and concerns around our privacy are challenging productivity and technical advancement for the first time it is important to understand the software we are using to best meet a trade-off between keeping our *private data* private while maintaining a stable and acceptable level of technological advancement.

1.1 The Concept of Measurement

Measurement is defined as the assignment of a number to a characteristic of an object or event, which can be compared with other objects or events. (Pedhazur, Schmelkin, 1991). Data is measured using a series of standardised steps in the hopes of using said data individually or by aggerating it to test hypotheses, plan further action and to better understand a process or the object being measured. We need to take measurements to better understand our work processes and products. Measurement provides us with the tools to compare trends in our data and to understand areas we may be able to improve. It tells us the objective quality of our work. Without measurement we are scientifically blind, only guessing about our aptitude and the calibre of our development processes.

1.2 Specifics to Software engineering

The software measurement process must aim to methodically process information to measure, evaluate, adjust and most importantly improve the software development process. There are many easily quantifiable elements of the software engineering process like lines of code, run times, number of errors and adherence to a coding standard. However, there are incredibly complex and seemingly immeasurable elements to the software development process like team productivity, the ability of an individual software engineer, the improvement of a program version to version at each iteration and team comprehension of both the product and its process. While measuring these complex variables is challenging and can seem daunting it is necessary to improve a product beyond current levels. In software there are 2 categories of metrics; software engineering metrics and software or application performance metrics. This report will focus on the former set of metrics.

1.3 Making the most of Measurement

It was the famous physicist Lord Kelvin who shared the sentiment that "If you cannot measure it, you cannot improve it." Which highlight the imperative nature of measurement and what it means in terms of our advancement. However, for out metrics to be useful we must understand how to measure each metric most effectively.

For Software metrics to be useful the must be easily understandable.
they must be:

- Simple and computable
- Consistent and unambiguous (objective)
- Applied with consistent units of measurement
- Independent of programming languages
- Easy to calibrate and adaptable

Measure what is necessary

You can measure almost anything but you simply cannot pay attention to everything. This is to say that if you give 5 metrics equal attention each you are relatively less attentive to each metric than if you only chose 3 to study. Choose only the metrics you feel are most important to you software development process to achieve the maximum result from your collected data.

Track the Trends of your data

Reaching a target is always good. However it is the trends in your data that will show you any true systematic progress within a company. (Bughin, Chui, Manyika, 2010) An individual or unique improvement in one area is not indicative of an overall improvement in your developmental environment. Keep an eye on the process overall.

Don't use any metrics for individual performances.

Resist the temptation to make any judgement on individual performances based on metrics. Metrics help you make inquiries to understand what really happened and therefore better understand the intricacies of the project and managing a team. (Kasurine, Taipale, Smolander, 09) They do not speak about the weaknesses or strengths of an individual team member.

Stop using software metrics that do not lead to change.

Comparing snowflakes is useless (Schneidewind, 02). Be attentive to the metrics you are tracking and make sure they can be used in the future to initiate positive change. If you discover that a metric you once thought valuable is in reality a useless statistic ask yourself; is that an inherent problem with the issue your having or is it the metric itself which holds the problem. Be mindful of how your metrics can be used.

2 The Implementation of Measurement

We have already established that for any company serious about improving their software development process that software metrics are a necessity not a choice. However, knowing *'to do'* something and knowing *'how to do'* that something are two completely different things. This report plans to describe, analyze and compare the different algorithmic approaches used to measure software metrics and the computational platforms that use them in an attempt to better understand what is useful and what isn't.

2.1 Types of Metrics

Agile Metrics

Project or Sprint Burndown

This metric is about project status. It is related to team output. Some teams only consider the number of tasks to be done when thinking of project size, but that would assume that all tasks are equal, which is simply not the case. Instead, some teams consider story points. This gives each task a weight in terms of its importance/complexity. The best way to assign these weights is to look at your team's history and compare the progress on the current release, to previous ones.

Ticket Close Rates

Ticket close rate are the amount of stories or story points your team or any contributor solved during a certain period of time. You should never use this metric to evaluate the individual performance of developers. A developer could fix one bug that nobody managed to solve, one that is impacting every aspect of your product's performance, and it could take him or her a full week. In the meantime, another developer could fix 20 small not impactful bugs. It is a relativistic measurement

Size-Oriented Measurements

Lines of Code (LoC) Produced.

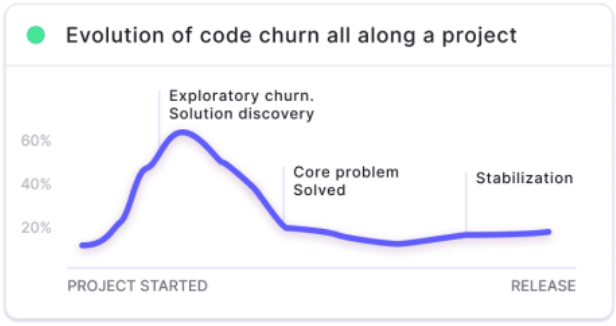
LoC simply refers to the amount of lines physically in your code. You can use LoC in the same way as Ticket Close Rate to understand when your team is having issues, like importing too many libraries i.e. quantity over quality. The issue is that knowing how many lines of code were added does not tell you anything about the code itself, 100 line of code could contain many bugs vs 1 line of code which fixes an issue.

Production Metrics

These metrics are interested in how much work is being produced by a team over a given time period. There are a number of ways to measure this, such as:

Code Churn

Code churn is typically measured as the percentage of a developer's own code representing an edit to their own recent work. To compute it, we measure the lines of code (LoC) that were modified, added and deleted over a short period of time such as a few weeks, divided by the total number of lines of code added.

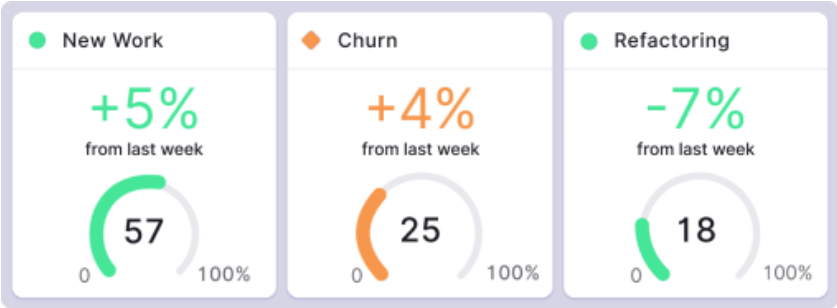


Refactoring Rate

Refactoring is the amount of time spent changing legacy code. You could measure this by the amount of commits needed to replace old code. In effect the percentage ration between lines of code replacing old code vs total number of code changed. This is more an indicator for managers.

New Work

New work would just be defined as the lines of code added to the base, not replacing any existing code. This metric could be computed as the percentage of new code contributed on the total number of lines of code changed. In that sense, it would be complementary to code churn and refactoring.



Function-Oriented Methods

These metrics are function orientated. They consider things like user inputs, error reports and messages, user inquiries etc. They care more about how the software functions when used than the actual code itself.

Security Responses Metrics

This type of measurement is used to understand how security responsive software is. In an age where data protection is of the utmost importance security metrics are essential to detect how much time is needed to fix and overcome security breaches. These are difficult to establish as the metrics are context sensitive and environment dependent. We can measure : Size and complexity, Defects / LoC, Defects (severity, type) over time, Cost per defect, Attack surface (# of interfaces), Layers of security and Design Flaws .To try and understand our vulnerability and risk of security failures within a piece of code.

QA Metrics

These encompass the quality metrics that must be taken when taking measurements. Testing is an integral part of any development process. There are numerous metrics for software equality testing. The basics include test cases executed and test cases written. This means that you determine an issue to be tested write code to test that issue and see if at any point the code fails. JUnit testing in Java is a great example of this. The amount of test past per tests written is a good indication of how reliable one's code will be.

2.2 Algorithmic approaches

While the current metrics available to managers may be useful, they are inherently flawed by the human condition. There is only so much a manager can manage themselves. As Steve McConnel notes in his book 'Diseconomies of Scale in Software Development', there is an exponential relationship between the complexity of a project and time needed to solve it. That is to say a problem twice as complex will take exponentially longer than twice the time to solve. So, it is no surprise that academics have sought to dive deeper into the data around the software development process and as computer scientists do, have produced algorithms for maximum efficiency in measurement.

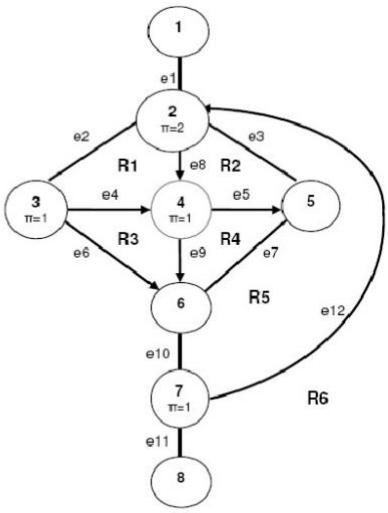
Cyclomatic Complexity

Cyclomatic Complexity is a quantifiable measurement of complexity of a system. A Techrepublic study recently found that the level of complexity in a program has been empirically correlated to coding errors within that program. This increased error propagation in turn impacts the cost of development of a project.

Edges and Nodes Method:
v = e – n + 2
e = 12, n = 8
v = 12 – 8 + 2 = 6

Predicate Method:
v = Σπ + 1
Σπ = 5, sum of predicates
v = 5 + 1 = 6

Region (Topological) Method:
v = ΣR, sum of regions R
ΣR = 6
v = 6



Cyclomatic complexity formula:

$M = E - N + 2P$,

where

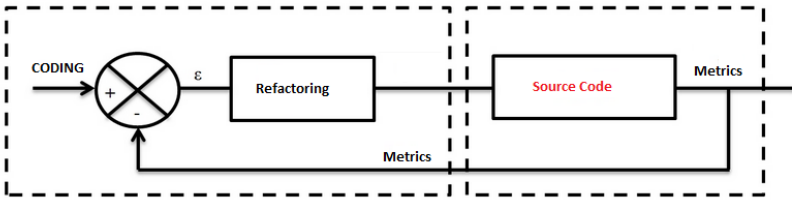
E = the number of edges of the graph.
N = the number of nodes of the graph.
P = the number of connected components.

However as the complexity of a problem increases you can only compare projects with similar complexities, think of a simple calculator program that is incomparable with software for a banking system.

Halstead's Software Science

Halstead distinguishes software science from computer science. The premise of software science is that any programming task consists of selecting and arranging a finite number of program "tokens," which are basic syntactic units distinguishable by a compiler.(Halstead, 1977). A computer program, according to software science, is a collection of tokens that can be classified as either operators or operands. The primitive measures of Halstead's software science are:

n1 = no. of distinct operators
n2 = no. of distinct operands
N1 = no. of operator occurrences
N2 = no. of operand occurrences



Program Length: N = N1 + N2
Program volume: V = N log 2 (n1 + n2)
(represents the volume of information (in bits) necessary to specify a program.)
Specification abstraction level: L = (2 * n2) / (n1 * N2)
Program Effort: E = (n1 + N2 * (N1 + N2) * log2 (n1 + n2)) / (2 * n2)
(interpreted as number of mental discrimination required to implement the program.)

Algorithmic Cost modelling

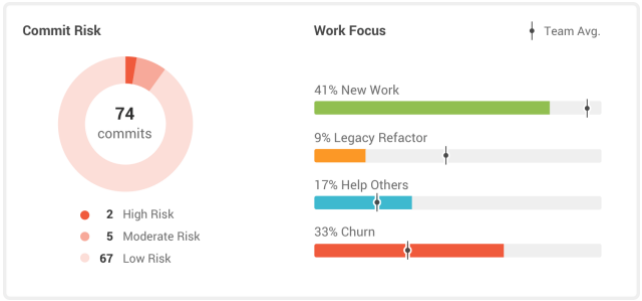
Build model by analysing the costs and attributes of completed projects. Dozens of these around -- most well-known is COCOMO. Assumes software requirements relatively stable and project will be well managed. Basic COCOMO uses estimated size of project (primarily in terms of estimated lines of code) and type of project (organic, semi-detached, or embedded).

Effort = A * KDSI b

where A and b are constants that vary with type of project.

2.3 Computational platforms

Gitprime



Gitprime is a paid professional service that can be used to measure some of the aforementioned metrics. Gitprime allows a user to see commit levels in a repository and the amount of new work within each commit, it also gives data on legacy refactoring and code churn. It helps to give an oversight of actual work being done in each commit. Gitprime also provides services which allows managers to determine if the agile sprint cycles for a given week had a net positive or negative effect. Moreover, they also claim that they can determine the *riskiest* commits in the scheme of the overall process which raises red flags and pinpoint commits which may need to be reviewed.

Hackystat

Hackystat is an open source framework for collection, analysis, visualization, interpretation, annotation, and dissemination of software development process and product data. It unobtrusively collects and sends data about development to a Hackystat server which can be queried for information that can then be used to visual and communicate data about a project. It is used by both researches and the wider industry to organise, plan and control processes using an automated software suite. It is easily integrated into any git derivate.

Humanyze

Collecting raw data about a development process is useful however it completely disregards the idea that the humans and individuals working on the code have any influence on the process and the code itself. Humanyze challenges this. Humanyze is a High-Tech startup which tracks everything a company's employees do during the day. The company's Sociometric badges contain microphones, an accelerator and Bluetooth to track your position, movement and speech. Humanyze currently works with over 50 of the fortune 500 companies across a range of industries like IT, Big Pharma and Energy. The company understands possible privacy concerns and stresses that it uses and anonymizer to keep your data private. It also stresses that its service is used only to measure overall productivity and not the productivity of an individual.

3. Measuring the Ethics of Software Measurement

3.1 Big Data

Big Data Ethics also known as simply Data Ethics refers to systemising, defending, and recommending concepts of right and wrong conduct in relation to data, in particular personal data. Last year Europe saw the introduction of the now infamous, General Data Protection Regulation (GDPR), a constitution for ownership of data within the EU. In layman's terms the EU recognised the value of an individual's data in the modern age and saw it fit to pass legislation to regulate data and its ownership.

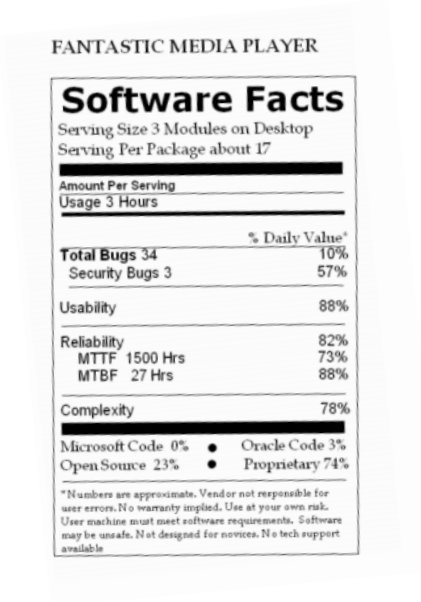
As the collection, organisation and retention of data has become common place in modern business the ethical implications behind collecting this data have also grown. We must ask ourselves what is a reasonable level of surveillance on an individual and their data and more importantly what is the reason for this surveillance. Authors Kord Davis and Doug Patterson address all the issues surrounding bid data in their aptly titled work "Ethics of Big Data." Davis remarks "Big data itself, like all technology, is ethically neutral. The use of big data, however, is not. " Unfortunately there is no 'Bill of Rights' for big data, so at the minute legislation is hazy around what is and isn't ethical. It can be difficult to define when a breach of personal security has been committed.

3.2 Trade-off between privacy and Advancement

However vigorously we shout that we want our data to be private, in practice these sentiments do not hold true. Think of Apple's Geolocation which is on every iPhone. We happily give away our exact location to apple without a second thought despite the fact if they were to install CCTV cameras outside your house the people would revolt. We only seem to care about our data when it is convenient.

It is hard to say for sure what the future holds for ethics around measurement and how that data is used, like Humanyze for now its data is anonymous but if a sinister organisation got hold of that information irreparable damage could be done to countless careers. It is without doubt we need legislation and protection of our personal information however, fear of data collection and adequate online surveillance may in turn be causing a sub optimal level of advancement, with companies and individuals alike facing red tape when trying to achieve technological innovation. We live in an exciting albeit scary time for technological advancement and ethical maturity surrounding this advancement.

4. Conclusion


FANTASTIC MEDIA PLAYER

Software Facts

Serving Size 3 Modules on Desktop
Serving Per Package about 17

Amount Per Serving
Usage 3 Hours

	% Daily Value*
Total Bugs 34	10%
Security Bugs 3	57%
Usability	88%
Reliability	82%
MTTF 1500 Hrs	73%
MTBF 27 Hrs	88%
Complexity	78%

Microsoft Code 0% • Oracle Code 3%
Open Source 23% • Proprietary 74%

*Numbers are approximate. Vendor not responsible for user errors. No warranty implied. Use at your own risk. User machine must meet software requirements. Software may be unsafe. Not designed for novices. No tech support available

Today the conscious developer is as aware of the statistical information they produce as they the products they put into their body. Using software metrics may seem cumbersome and, at times, useless however, as outlined above, metrics can be used to highlight areas of improvement and possible failures within the software development life cycle of a company. Without measurement we are blindly chasing advancement with no evidence that our actions will result in a positive change. While there are ethical surrounding the ethics of collecting this data and who it is shared with, it is imperative for the sake of our technological advancement as a species that we embrace software metrics into our software development processes even more so than we already do. Measurement can be thought of as an '*informational label*' of sorts for your program allowing developers and users alike to decide which software product they are most likely to consume. They tell you all you need to know about a given piece of software

Bibliography

Pedhazur, Elazar J.; Schmelkin, Liora Pedhazur (1991). Measurement, Design, and Analysis: An Integrated Approach(1st ed.). Hillsdale, NJ: Lawrence Erlbaum Associates. pp. 15–29. ISBN 978-0-8058-1063-9.

N. Schneidewind, "Methodology for Validating Software Metrics" in *IEEE Transactions on Software Engineering*, vol. 18, no. 05, pp. 410-422, 1992. doi: 10.1109/32.135774

ussi Kasurinen, Ossi Taipale, Kari Smolander, "Analysis of Problems in Testing Practices", *Software Engineering Conference 2009. APSEC '09. Asia-Pacific*, pp. 309-315, 2009.

Calikli, Gul and Bener, Ayse (2013). An Algorithmic Approach to Missing Data Problem in Modeling Human Aspects in Software Development. In: PROMISE ’13: 9th International Conference on Predictive Models in Software Engineering, ACM, New York, USA, article no. 10.

Organizational citizenship behaviors: A critical review of the theoretical and empirical literature and suggestions for future research – Philip M. Podsakoff, Scott B. Mackenzie, Julie Beth Paine, Daniel G. Bachrach - 2000

https://www.researchgate.net/publication/220344622_Measurement_in_software_engineering_From_the_roadmap_to_the_crossroads

https://www.researchgate.net/publication/260480820_Software_measurement_and_software_metrics_in_software_quality

https://anaxi.com/software-engineering-metrics-an-advanced-guide/
http://www.forevueinternational.com/Content/sites/forevue/pages/1392/5_3_6_Clouds_big_data_and_smart_assets_McKinsey.PDF

url: https://doi.ieeecomputersociety.org/10.1109/32.135774

https://www.careeraddict.com/the-highest-paid-jobs-in-the-world

http://sunnyday.mit.edu/16.355/metrics.pdf