CS3012 – Software Engineering Essay

Liam Egan - 17340992

Introduction:

Over the course of this essay, the following will be discussed in-depth:

"How 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."

Finding a well-rounded method to measure the software engineering process is not an easy task.

What exactly is "measuring the software engineering process"?

We must first define a software metric:

A software metric is a measure of software characteristics which can be quantified or counted. They are important for many reasons, including, but not limited to: measuring software performance, planning work items and measuring productivity.

Within the software development process, there are many metrics that are all related to each other.

Why we use Software Metrics

The reason we track and analyse software metrics is to determine the quality of the current program or process, improve said quality and predict the quality once the software development project/program is complete. More specifically, those managing software development teams are trying to achieve the following:

- Financial reasons - Increase return on investment, reduce overtime and thusly reduce costs.
- Analytics - identify areas of improvement
- Management - manage workloads

These goals can be achieved by providing information and total transparency throughout the organization about complex software development projects. Metrics are key components of quality assurance, management, debugging, performance, and estimating costs, and are valuable to those on all levels of the software development process, from developers up to managers:

- Those on the management team can utilise software metrics to identify, prioritize, track and communicate any issues to the development team, thus increasing team productivity. This makes for effective management and allows assessment and prioritisation of problems within software development projects. The sooner the management team can detect software problems, the more straightforward and cost-effective the troubleshooting process.
- Software development teams can use software metrics to communicate the status of software development projects to the management team, pinpoint and address any issues they encounter, monitor, improve on, and better manage their tasks as a result of a better-informed management team.

Software metrics can display the impact of decisions made during software development projects. This helps the management team assess and prioritise objectives and performance goals and makes for a clearer stream of communication with the development team.

The nebulosity of Software Metrics and LOC

The exact meanings of terms used to describe software metrics can be ambiguous as they often have multiple definitions and ways to count or measure characteristics. One example would be the lines of code of a program (LOC). This is a common measure of software development, however, there are arguably two ways to count each line of code:

1. Each line that ends with a return. This counts comments and empty lines and is often refuted as programs can often technically be written on one line.

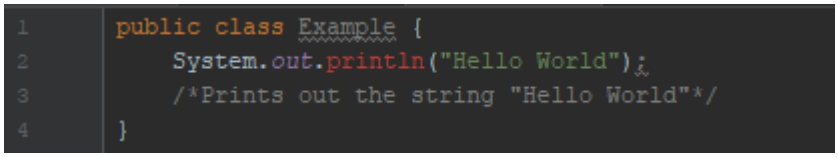

Fig 1

4 Lines

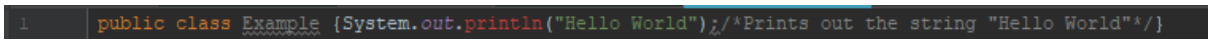

Fig 2

1 Line

Same program.

2. To circumvent these issues, each logical statement may be considered a line of code.

As a result, when gathering Software Metrics on the same program one could have two very different LOC counts depending on which counting method is used. That makes it difficult to compare software simply by LOC or any other metric without a standard definition agreed upon before the project's commencement, which is why establishing a measurement method and consistent units of measurement is crucial.

The usage of Software Metrics may also be an issue. If a team utilises metrics that measure productivity with emphasis on volume of code and errors, the development team could act deceptively and avoid tackling tricky problems to keep their LOC up and error counts down. Writing a large amount of simple code may result in great productivity numbers but show poor software development skills on behalf of the developer. Additionally, a lot of thought should go into establishing which Software Metrics are monitored, they shouldn't be monitored simply because they're easy to obtain and display – software metrics are there to assist you, not as some sort of

meaningless requirement – only metrics that can be used to improve the project/process should be tracked.

How to Track Software Metrics

Software metrics are great for management teams because they offer a quick way to track software development, set goals and measure performance. But oversimplifying software development can distract software developers from the experimentation and trial-and-error aspects of programming, and goals such as delivering genuinely useful software and increasing customer satisfaction.

However, obviously, none of this matters if the measurements that are used in software metrics are not collected or the data is not analysed. One major problem is that software development teams would most likely consider it more important to actually do the work than to measure it.

It becomes crucial for the management team to make measurement easy to collect otherwise developers may simply not collect it at all. As stated earlier, it is crucial to make the software metrics work for the software development team so that the whole team can work better. Measuring and analysing these metrics needn’t be burdensome or something that disrupts the natural progression and development of code by developers. Good Software Metrics are simple and computable; consistent and clear; use consistent units of measurement; are independent of programming languages; easy and cost-effective to obtain; able to be validated for accuracy and reliability; and finally, relevant to the development of high-quality software products.

Deciding how to use software metrics should take priority over how software metrics are collected, calculated and reported. Four important guidelines for an appropriate use of software metrics outlined by successful software architect, and former CTO of N26 Bank Patrick Kua are as follows:

1. Stop using software metrics that do not lead to change.

Repeating the same work without adjustments that do not achieve goals is the definition of managing by metrics – also, oddly enough, the definition of insanity.

A developer's inability to focus on goal that lead to better software experiences can be attributed to trying to reach the wrong goals. Priorities are key.

Some software metrics are useless when it comes to indicating software quality or how well the team is working together. Both management and software development teams need to work on software metrics that drive progress towards goals and provide verifiable, consistent indicators of progress.

2. Track trends, not numbers.

It is important that the management team is aware of the fact that the trends of which certain datapoints are a part of are far more significant than the datapoint itself. Ignoring trends could lead to the downfall of a project due to an inability to see how changes implemented affect the rate at which goals are met/not met. 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 than any single datapoint will.

For example, size-based software metrics often measure LOC to indicate coding complexity or software efficiency. In an effort to reduce the code's complexity, the management team may set targets on how many lines of code are to be written to complete a function. In trying to simplify functions, software developers could write more functions that have fewer lines of code to reach their target but do not reduce overall code complexity or improve software efficiency and could lead to the omittance of certain error cases being accounted for or some other vulnerabilities as a result.

Before the commencement of the project it is crucial that the management team involves the software development teams in

establishing goals, choosing software metrics that measure progress toward those goals and align metrics with those goals.

3. Set shorter measurement periods.

Software development teams have a tendency to spend their time getting the work done, not checking if they are reaching management established targets – it is important that this “flow” is not disturbed, as it could inadvertently lead to a lower quality product. A good approach would be a hands-off approach whereby a target is set some time in the future that, until then, the software team are not told if they have met.

By breaking the measurement periods into smaller time frames, the software development team can check the software metrics — and the trend line — to determine themselves how well they are progressing, rather than being told by management. This relieves the pressure of reaching one massive final goal and will keep up productivity consistently rather than big spikes at each evaluation period.

While it is an interruption to the aforementioned “flow”, giving software development teams more time to analyse their progress and change tactics when something is not working is very productive. The shorter periods of measurement offer more data points that can be useful in reaching goals other than exclusively software metric goals, which leads nicely onto the next point of Patrick Kua’s.

4. The linkage of software metrics to goals.

In order to shift focus to that desired by the management team, software metrics will be presented to the development team as goals such as the following: Reducing the lines of codes; reducing the number of bugs reported; increasing the number of software iterations; and speeding up the completion of tasks.

Focusing on these metrics as goals help developers reach the more important goals such as improving software usefulness and user experience.

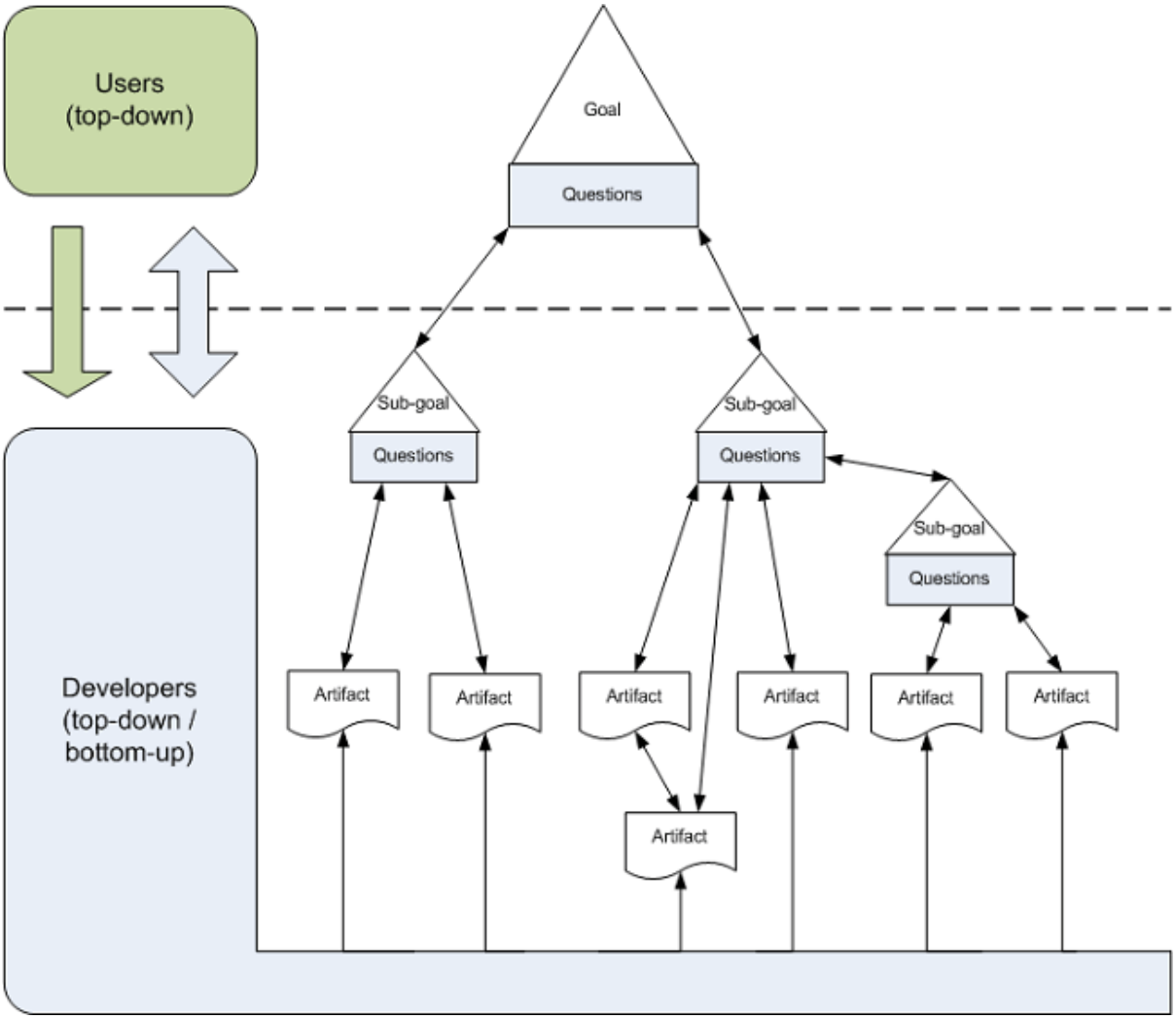

Fig 3 - via Wikipedia

Examples of Software Metrics

Agile process metrics

A process we studied in the course, agile process metrics assess the agility of the team in its decision making and planning.

Lead time

Lead time is how long it takes for ideas to be developed and delivered by the development team in the form of software, or for features to be

implemented. The higher this time, the less responsive developers are to user's feedback.

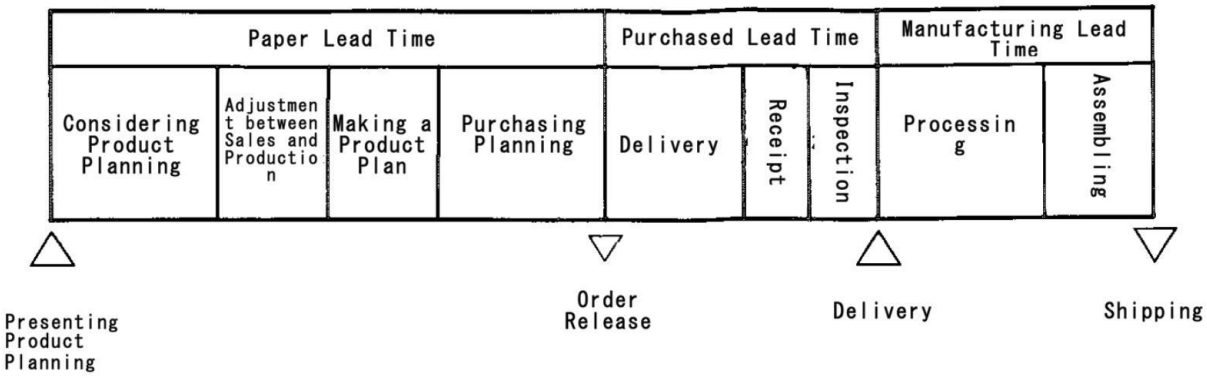

Fig 4 - via https://www.asprova.jp/mrp/glossary/en/cat247/post-561.html

Cycle time

Cycle time describes how long it takes to change the software system and implement that change in production. This time is often confused with Lead Time, the below diagram should help clarify the differences.

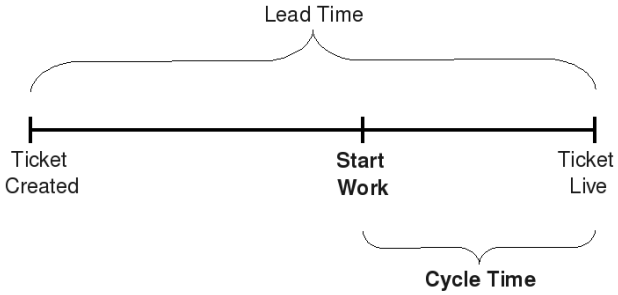

Fig 5

Team velocity

This refers to the turn out of one development team in a pre-defined time-frame – this will differ per team as the definition of deliverable differs per team due to the difference in expectations etc.

Production

This metric essentially quantifies how much work is done and determine the efficiency of different software development teams.

Different expectations will alter this metric – some management teams will prioritise speed over quality and vice versa.

Active days

Rather than assessing a developer's productivity in a broad, non-specific way like how many days on which the project was worked on, this metric assesses how many actual days of work are gotten out of a developer by summing together all active time – it highlights the hidden wasted time throughout the day.

Assignment scope

How much code one programmer can maintain in a year. This metric can be used to establish how many developers are required post-development phase to keep a project running.

Code churn

Code churn represents the number of lines of code that were modified, added or deleted in a specified period of time. If code churn increases, then it could be a sign that the software development project needs attention.

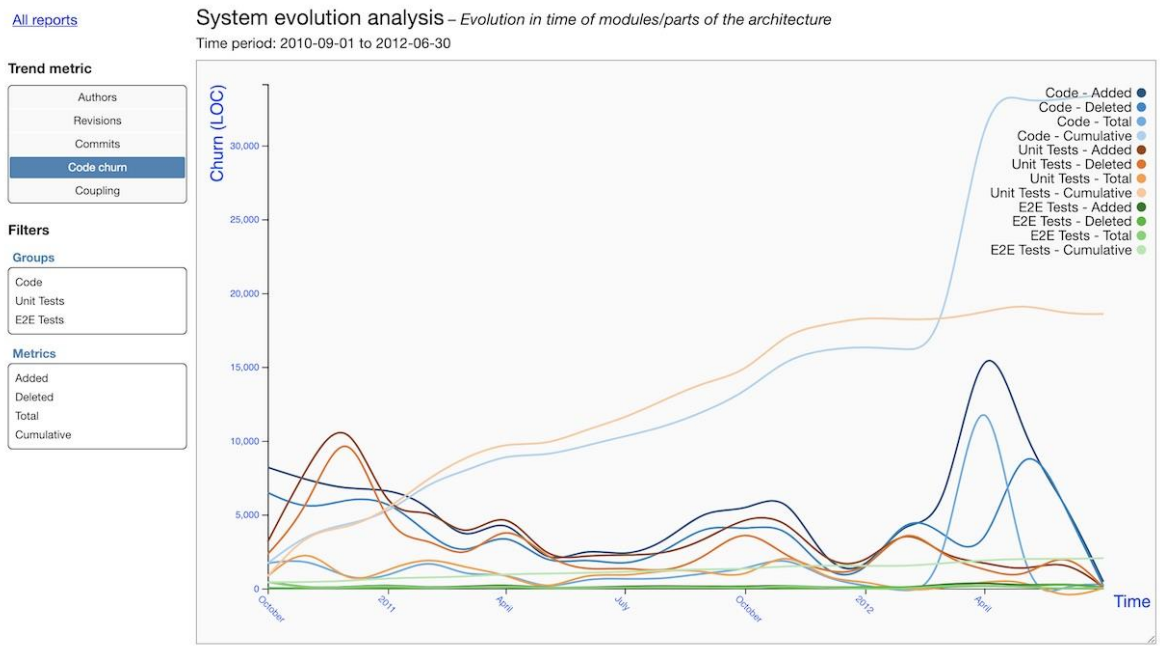

Fig 6 – via https://github.com/smontanari/code-forensics/blob/master/README.md

Impact

This measures how important changes are.

Security metrics

Security metrics reflect software quality. These metrics are very important and need to be tracked over time to show how software development teams are developing security responses to vulnerabilities discovered internally or externally. These metrics can carry very severe repercussions if user data is at risk.

Size-oriented metrics

Size-oriented metrics focus on the size of the software (as touched on earlier) often presented as kilo lines of code (KLOC). One of the simpler metrics in terms of collection – however in definition it is not so straightforward (as described earlier). Unfortunately, it is not useful for comparing software projects written in different languages. (Fig 7). Examples include: Errors per KLOC; Defects per KLOC; Cost per KLOC.

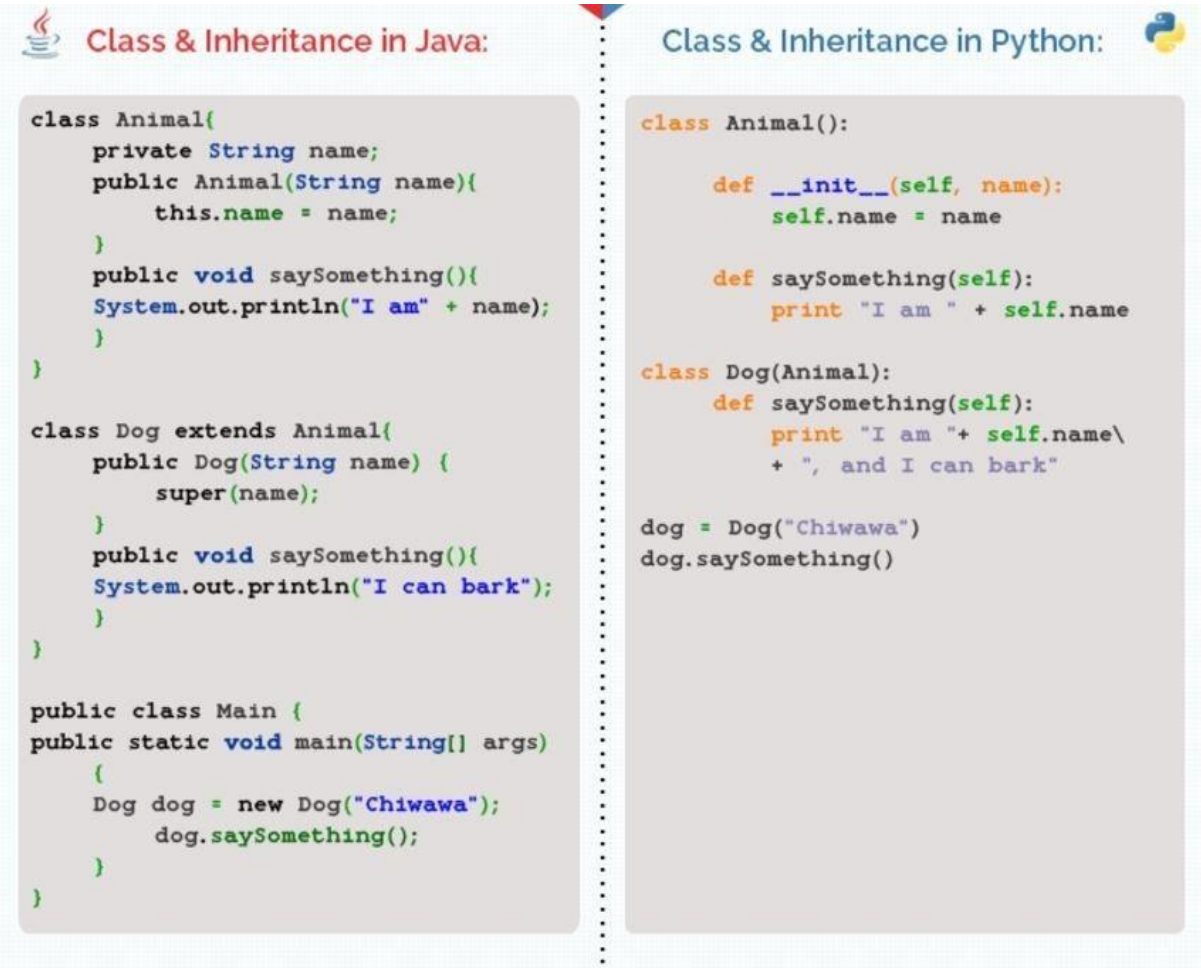

Fig 7 – via https://jaxenter.com/java-vs-python-in-fintech-140955.html

Platforms to compute Software Metrics

Luckily, at the moment there are many open-source toolsets available to compute many of the metrics outlined in this essay. There are toolsets for almost all languages out there.

SemanticDesigns provide the JavaMetrics and C# Metrics to measure metrics of Java and C# Programs these programs provide very useful and in-depth data which I have seen computed first-hand. Below is an example of some sample output from JavaMetrics:

```
Summary:
    4389 lines of source.
    2204 lines of Java code.
    1782 lines of comment.
    2 packages.
    4 files.
    11 types.
    234 methods.
    Cyclomatic complexity: 623
    Conditional statements: 320
    Decision density: 2.68
    Max loop depth: 2
    Max loop depth position: <file d:/users/hzheng/JavaMetricsTests/MultiClassInOneFile.java>::MultiClassInOneFile @ line 8
    Max conditional nesting depth: 5
    Max conditional nesting depth position: <file d:/users/hzheng/JavaMetricsTests/JTree.java>::javax.swing.JTree @ line 2491
    Halstead unique operators: 523
    Halstead unique operands: 443
    Halstead operator occurrence: 3982
    Halstead operand occurrence: 2472
    Halstead program length: 6454
    Halstead program vocabulary: 966
    Halstead program volume: 63997.09
    Halstead program difficulty: 1459.21
    Halstead program effort: 93384893.89
    Halstead bug prediction: 21.33
    SEI maintainability index: 155.15

    PACKAGE %default_package%
        159 lines of source.
        134 lines of Java code.
        9 lines of comment.
        3 files.
        4 types.
        5 methods.
        Cyclomatic complexity: 27
        Conditional statements: 20
        Decision density: 0.87
        Max loop depth: 2
        Max loop depth position: <file d:/users/hzheng/JavaMetricsTests/MultiClassInOneFile.java>::MultiClassInOneFile @ line 8
        Max conditional nesting depth: 4
        Max conditional nesting depth position: <file d:/users/hzheng/JavaMetricsTests/MultiClassInOneFile.java>::MultiClassInOneFile @ line 8
        Halstead unique operators: 83
        Halstead unique operands: 120
        Halstead operator occurrence: 513
        Halstead operand occurrence: 359
        Halstead program length: 872
        Halstead program vocabulary: 203
        Halstead program volume: 6684.17
        Halstead program difficulty: 124.15
        Halstead program effort: 829867.92
        Halstead bug prediction: 2.23
        SEI maintainability index: 100.19
```

Fig 8 – via https://www.semanticdesigns.com/Products/Metrics/JavaMetrics.txt

OWASP provide a range of SAST (Static Application Security Testing) tools to collect security metrics and weed-out any vulnerabilities in a program including but not limited to:
the SonarQube Orizon Projects.

As touched on in this essay, software metrics extend beyond the actual software and code itself – for example Humanyze is a workspace analytics solution that doesn't solely focus on the code aspect of the software development process, but rather focusses on the developers as people.

Algorithmic approaches to computing Software Metrics

The following are useful algorithms for computing metrics

Application crash rate (ACR)

Application crash rate is calculated by dividing how many times an application fails (F) by how many times it is used (U).

ACR = F/U

Defect Removal Efficiency (DRE)

The Defect Removal Efficiency is used to quantify how many defects were found by the end user after product delivery (D) in relation to the errors found before product delivery (E). The formula is:

DRE = E / (E+D)

The closer to 1 DRE is, the fewer defects found after product delivery.

Ethics surrounding Software Metrics

At the centre of this entire discourse is one thing – data. Data can range from anything from a 0 or 1 to a full-scale program. This of course includes personal data. When an employer seeks to monitor a developer's performance, they will most-likely record the data input into their computer. This is not strictly limited to the tech sector, this practice occurs in most workplaces – including retail, where managers may record the time spent by employees doing certain tasks. I don't believe this to be an issue – once you're doing what you should be then there are no issues, where it does become an issue is when private information is illegally gathered.

In a case in Germany a web developer was fired after his employer used a key-logger to monitor what he was doing at his workstation. They had been notified that the company property was not to be used for personal use, however said employee used it for just that. The fired employee successfully sued the company after a judge decided the utilisation of said key-logger was illegal. In this instance, the employer would have been able to see what the employee was doing on his computer, logging everything he typed – including but not limited to passwords, private messages, usernames and potentially view what was on his monitor. I feel this is a step too far – especially considering the fact that they were not notified that their keystrokes were being logged.

The ethics of these metrics extends beyond standard cases, in the instance where say someone has diabetes, or some sort of illness that requires time away from one's workstation throughout the day – their active days metric score would decline, however in the time they are at their workstation they could be outputting much more than other employees.

Much like the metrics themselves, the ethics surrounding them are just as nebulous. There is no one perfect metric collection method from a data standpoint or an ethical one – each has its own upsides and downsides and applicability to certain instances.