MEASURING ENGINEERING - A REPORT

Alexander Fields

15314665

Fieldsal

fieldsal@tcd.ie

In this report I intend to outline some of the different metrics involved in the measurement of software engineering. I will discuss the topics of measuring software, measuring data, algorithmic approaches to measuring software and the ethics of measuring software.

MEASURING SOFTWARE

There are many common ways in which to measure software engineering. Software is measured my using metrics, which are a standard measurement of the degree to which a process possesses a property. The main goal of measuring software is to reduce the latency and errors within software development. They are used for quality assurance, cost estimation, testing, debugging and performance optimisation.

Some common metrics for measuring software are as follows:

Code Test Coverage – this is used to show how much of a programs source code is run through a test suit, the higher the percentage code coverage the more of the processes code has been executed during testing. This is good for finding bugs in code, and allows developers to fix bugs before the process is put into circulation.

Code Cohesion – This refers to the degree to which the elements within a process belong together. It measures the relationship between the methods and data of a process.

Program Load Time – This simply is the amount of time it takes for the process to load all of the libraries and processes that it is dependant on. It is a fundamental metric in any process and by reducing this, both cost and latency can be reduced.

Program Execution Time – This is the runtime, the amount of time it takes for the program to execute though each of its phases, such as compile link and load. Similarly, to Load time, reducing this metric reduces both cost and latency.

Coupling – This is the degree of interdependence between processes. A Measure of how closely connected two processes are, and the strength of their relationship.

Number Of Lines Of Code – This metric is used to measure the size of a software process. It is not however the most reliable metric as the number of lines of code does not directly correlate to the quality of the code. For example, something that one developer could write 500 lines of code for could be written by a separate developer in 100 lines of code. The old adage quantity does not mean quality stands here.

Instruction Path Length – This is the number of machine code instructions that are needed to execute a process. As a metric the total path length of a process could be

used as a measure of a process's algorithmic performance. However, this may vary when using different hardware systems.

Maintainability Index – This is the ease with which a process can be maintained in order to

Correct Defects

Repair or replace faulty components of a system without having to replace the function components.

Design Structure Quality Index (DSQI) – this is a design metric which is used to evaluate the structure complexity and efficiency of a process.

McCabe's Complexity – Also know as Cyclomatic Complexity is a metric used to indicates the complexity of a program. It is a quantitative measurement of the number od independent paths through a programs source code.

There are some limitations to these measurement metrics, however. Due to the high variety of methods and processes it is often difficult to define and measure software qualities and quantities. Moreover, it is difficult to determine a coherent concurrent measurement metric, especially when trying to predict outcomes before design. It is also important to note that not everyone agrees on which metrics matter, and what their data entails.

MEASURABLE DATA

When discussing the measurement of software engineering it is important to note that not all measurements can be used in all cases. Depending on the way a software engineer or group of software engineers work, different metrics may need to be used. For agile and lean processes, the most reliable metrics to use would be:

Leadtime

Cycle time

Team Velocity

Open/Close rates

These metrics help with informed decisions about the improvement of processes and aid planning.

Leadtime is how long it takes you to go from idea to delivered software. If you want to be more responsive to the consumers using your software, reduce your overall Leadtime. This can be done by reducing wait time and simplifying decision making.

Cycle time is how long it takes you to make a change to your software and put that change into the main software for consumers to use. Engineers who use continuous delivery can have cycle times that can be anywhere between minutes and seconds, rather than months.

Team Velocity is how much software an engineer or team of engineers can typically complete in one iteration. This number should only be used in the planning process of future iterations. Comparing team velocities is seen to be a pseudoscience however as the metric is based on non-objective estimates. When reviewing velocity as a measurement of success, making a specific velocity a goal distorts its value for estimation and planning.

Open/Close rates are how many production issues are reported and fixed within a specific time period. For example, have all of the bugs that have been reported been fixed within a specific iteration. The specific numbers of this however matters less than the general trend of the open/close times.

Metrics however do not determine the root causes of problems, if these metrics are trending out of the expected range, it is important to get information from the software engineer or engineers as a different root cause may be the reason that the metrics are fluctuating.

With relation to production analysis there are two metrics that can be used:

Mean Time Between Failures (MTBF)

Mean Time To Repair/Recovery (MTTR)

Both of these metrics are overall measures a software system's performance within the confines of its current production environs.

Application Crash Rate is how many times an application fails divided by how many times it was used, this is used in relation to the MTFB and MTTR.

These three metrics however do not tell you much about individual features or the affected users. Still it stands to be said the smaller the numbers, the better. With modern operations-monitoring software makes gathering detailed metrics on individual programs and transactions fairly easy, however is time-costly and upscaling and downscaling may be difficult without the proper forward planning.

Statistically, software is bound to fail, however when it does fail it is the objective of the software engineers never to lose critical data and to be able to have instantaneous recovery.

There are also a number of security metrics that are sometimes overlooked, sometimes left until it is too late. These tools can be used in conjunction with specialized stress and evaluation tests during the build process. The reason the security requirements are often overlooked is because the majority of them are common sense. No matter how simple they may seem it is important that the software developer/developers need to be mindful of them and the metrics that they yield.

A few production metrics that can be used to aid in the security of a software product and the consumers satisfaction are:

Endpoint Incidents

MTTR

Endpoint Incidents simply are how many endpoints, i.e. clients have experienced a virus over a given iteration.

MTTR in security terms is the time between the discovery of a breach in security and a deployed working remedy. As with the production metric MTTR the security metric MTTR should be tracked over specific time period, or iteration. If over time the MTTR become

smaller it could be said that the developer/ developers are becoming more effective in discovering and fixing security issues.

As with Most of the metrics listed above, the smaller the value of the metric grows, the better set you are.

PERSONAL SOFTWARE PROCESS

The Personal Software Process (PSP) provides engineers with a disciplined personal framework with which to measure their software development. This process shows software engineers how to plan, measure and manage their work. The PSP is designed to be used with any programming language at both industrial and academic levels. It can be used writing requirements, running tests, fixing bugs and defining processes. Colloquially this is when a developer, tries to measure data based on their own work.

This can be done by taking information and statistics gathered from previous projects that the developer/ developers. However, this may lead to some issues.

Results may be worse than expected

Results may be distorted

When measuring oneself developers may be prone to overestimating their abilities. This may be disheartening for the developer, and may lead to them completely disregarding the results and start again using a different method. However, the same can be said for developers who underestimate their own abilities. Developers may see the outcome of their PSP and think that they have made a mistake in their metrics and seek to find another was to measure the software engineering process.

In the same breath a developer could see poor results from their PSP and try to manipulate the data to make it seem like they are making more progress than they actually are. These distorted results may make them look better however this is futile as they are only prolonging the problem when they could be working towards a better solution by focusing on the outcome of the data and trying to rectify any mistakes that they are making.

When developers use the PSP, the end goal is to produce a fully functional product on schedule and within the planned budget and time constraints. When used in conjunction with other methods such as Team Software Process (TSP) PSP can be a very effective method of helping engineers reach their goals.

ALGORITHMIC APROACHES

Many software cost estimation metrics have been developed based on size measurements, such as lines of code, which I stated above is not the best way to measure software engineering. Companies have been moving more towards mathematical and algorithmic methods.

One of such methods is the COCOMO metric (Constructive Cost Model) is used as an algorithmic metric to calculate effort there are three types:

Basic COCOMO

Intermediate COCMO

Detailed COCOMO

The variables used by the COCOMO method can vary from developer to developer. These variables can be fine tuned to get optimised solutions.

Meta-Heuristic Algorithms are also used as an algorithmic approach to software measurement. They are used for more complex optimisation problems than the COCOMO model. The Algorithms work on optimisation and evaluate the data multiple times until an optimised result is reached.

ETHICS

Within software engineering, there are many different codes of ethics that can be employed. In general, they are all used as a systematic set of rules principles and regulations developed by a community of developers to exclude and punish any undermining behaviour. Software engineers have the code of ethics over which they base their design systems on. They are used to provide guidelines on what should and shouldn't be done.

The code of ethics that is followed by the majority of software engineers is the IEE/AMC joint Code of Ethics 1999. It focuses on eight principles:

PUBLIC - Software engineers shall act consistently with the public interest.

CLIENT AND EMPLOYER - Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.

PRODUCT - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.

JUDGMENT - Software engineers shall maintain integrity and independence in their professional judgment.

MANAGEMENT - Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.

PROFESSION - Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.

COLLEAGUES - Software engineers shall be fair to and supportive of their colleagues.

SELF - Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

In relation to the collection of data and measurement of software one of the main concerns is what data to keep and how to handle the data that is kept. For example, if you

are collecting log files for errors with users, is the user information that you are storing in your log files encrypted? If not, this raises many ethical concerns.

The measurement of software engineering is very important, for quality and efficiency. There are many different approaches to the measurement of engineering and it is up to the developer or group of developers to decide which metrics best suit them, depending on the requirements of the software that they are developing and the information that they perceive as being most important. Software Engineers also need to take ethics into consideration to ensure a safe platform for their consumers and to allow everyone to work in a better environment.