**The Measurement of the Software Engineering Process**

*3rd year - CS3012*

*Matthew Armstrong*

*15335849*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Since it’s inception, the profession of writing software has become most concerned with creating and efficiently maximising the quality of the software written. Quality refers to both how practical the software is; from its stability, speed, usability, testability, readability, size, cost, security, and number of flaws or "bugs", and how pleasant it is; including qualities such as conciseness, finesse, customer satisfaction and many others. The how of creating high quality software is a controversial problem in its own right. It covers many software design principles, commonly known as “best practices” for writing code. It also covers the broader occupational issues such as optimal team size, optimal process, most efficient delivery of software, and ensuring adequate work place relations in the form of culture, hiring practices and so on. All these fall under the broad topic of software engineering.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Measuring Software

*Software Metrics*

*Software Size*

Size is the primary measure of software. It is measured in function points. Another method of measure is the source lines of code (SLOC), which is the functional code excluding comments. However, this is more an indication of the effort put in rather than the codes functionality. Different developers may all solve a functional challenge using different techniques resulting in varying numbers of SLOC but the exact same functionality results. The most reliable method for measuring software size is code agnostic, from the user's point of view - in function points.

Software metrics are a method of measurement whereby metrics are analysed against the code itself. They include simple metrics such as counting the number of lines in a single file, the number of files in an application, the number of functions in a file, and so on. These measurements have become common in software development practice. More detailed metrics that can be used will measure things like Halstead, Software complexity, cohesion and coupling.

Measurable Data

There are many means of measuring data but from my insight into this topic I can gather that the main metrics are:

**Size**

The lines of code (LOC) of a project as well as the Source lines of code (SLOC) will always be numbered as it is a metric used to determine quantity of function. Along with these function points are used in conjunction to give a measure with a project

**Productivity**

Simply the amount of time and effort taken and from that time the measure of functionality that has been achieved

**Time**

The duration of the project in calendar months and within those months the level of productivity that has been achieved

**Effort**

The amount of work produced by members of an org/team to produce the effective/wanted product

**Reliability**

In my opinion this is the most important metric and by and by it itself can judge whether the amount of a aforementioned metrics justifies the end goal. The reliability is the measure of a defect rate/mean time to defect that occurs within the software.

Personal Software Process

The Personal Software Process (PSPSM) provides engineers with a disciplined personal framework for doing software work. The PSP process consists of a set of methods, forms, and scripts that show software engineers how to plan, measure, and manage their work. It is introduced with a textbook and a course that are designed for both industrial and academic use. The PSP is designed for use with any programming language or design methodology and it can be used for most aspects of software work, including writing requirements, running tests, defining processes, and repairing defects. When engineers use the PSP, the recommended process goal is to produce zero-defect products on schedule and within planned costs. When used with the Team Software Process (TSPSM), the PSP has been effective in helping engineers achieve these objectives.

In the PSP, engineers use data to monitor their work and to help them make better plans. To do this, they gather data on the time that they spend in each process phase, the sizes of the products they produce, and the quality of these products. These topics are discussed in the following sections. Time Measures In the PSP, engineers use the time recording log to measure the time spent in each process phase. In this log, they note the time they started working on a task, the time when they stopped the task, and any interruption time. For example, an interruption would be a phone call, a brief break, or someone interrupting to ask a question. By tracking time precisely, engineers track the effort spent on the project tasks. Since interruption time is essentially random, ignoring these times would add a large random error into the time data and reduce estimating accuracy. Size Measures Since the time it takes to develop a product is largely determined by the size of that product, when using the PSP, engineers first estimate the sizes of the products they plan to develop. Then, when they are done, they measure the sizes of the products they produced. This provides the engineers with the size data they need to make accurate size estimates. However, for these data to be useful, the size measure must correlate with the development time for the product. While lines of code (LOC) is the principal PSP size measure, any size measure can be used that provides a reasonable correlation between development time and product size. It should also permit automated measurement of actual product size. Lines of Code (LOC) The PSP uses the term “logical LOC” to refer to a logical construct of the programming language being used. Since there are many ways to define logical LOC, engineers must precisely define how they intend to measure LOC. When engineers work on a team or in a larger software organization, they should use the team’s or organization’s LOC standard. If there is no such standard, the PSP guides the engineers in defining their own. Since the PSP requires that engineers measure the sizes of the programs they produce, and since manually counting program size is both time consuming and inaccurate, the PSP also guides engineers in writing two automated LOC counters for use with the PSP course

Algorithmic Approaches

Factors which are involved in the algorithmic approach include:

Lines of code

Simply the LOC of a piece of software, which in my opinion is not the best way to judge or measure its viability, is an outdated method of judging the code in its point of production and usefulness

Source lines of code

Almost identical to the LOC, Using the source lines of code is in my opinion also outdated and is back by Gates as he says that it is “like trying to judge the progress of a plane by how much it weighs”

McCabe cyclomatic complexity

Another algorithmic approach is through determining the number of test cases that are necessary to achieve through test coverage of a model

It is useful because of two properties of the cyclomatic complexity for a specific module where the branch coverage is dictated by the upper and lower bounds

I.e. Branch coverage < Cyclomatic Complexity < number of paths.

Meta-Heuristic Algorithms are another well known method algorithmic approach. They are used for the optimisation of larger and more complex problems and provide optimisation and test the data repeatedly until the optimal solution is reached

Ethics

Software engineering involved the wider responsibilities then simply the application of technical skills. Software engineers must behave in an honest and ethically responsible way if they are to be respected as professionals. Ethical behaviours is more then simply upholding the law but involves the following set of principles that are morally correct:

*Confidentiality*

Engineers should normally respect the confidentiality of their employers or clients irrespective of whether a formal confidentiality agreement has been signed.

*Competence*

Engineers should not misrepresent their level of competence. They should not knowingly accept work which is outside of their competence

*Intellectual property rights*

Engineers should be aware of local laws governing the use of intellectual property such as patents, copyright, etc. They should be careful to ensure that the intellectual property of employers and clients is protected

*Computer misuse*

Software engineers should not use their technical skills to misuse other people’s computers. Computer misuse ranges from relatively trivial (gaming on a work PC) to extremely serious (Dissemination of viruses).

*ACM/IEEE Code of Ethics*

The professional societies in the US have cooperated to produce a code of ethical practice

Members of these organisations sign up to the code practice when they join

The code contains eight principles related to the behaviours of and decision made by professional software engineers, including practitioners, educators, managers, supervisors and policy makers, as well as trainees and students of the profession

Assessment of Measurement

Through measurement of the progress made over time the schedule of productivity is formed through milestones. From this schedule we gain an pre conceived model of work done over time and hence from the end result we can glean information as to whether that model was correct and or if new factors arose that inhibit productivity or whether fundamental working order was achieved. The measure of resources and cost is the measure of balance achieved between work performed/to be performed and the personnel resources assigned with it from within the project. Product size and stability are an important metric as the measure of stability is a basic necessity of the a software engineer as to judge work needed and time to be spent which effect the aforementioned assessments. Along with functionality the quality of the software provided must have a metric to be measured against (usually that it meets all criteria stated at the beginning of the project). Hence the ability to support the user’s needs without failure on a constant basis is a necessary metric. Assessing the technical effectiveness of the approach after the end of the project is useful as an insight using all previous mentioned metrics gives a sum total of the effectiveness of the model used throughout the software.