MEASURING SOFTWARE ENGINEERING

Theodor Barbu
Student no.: 17334387

CONTENTS

1. Introduction
2. Measurement and Assessment in terms of Measurable Data
    a. What measurable data is
        i. Software metrics
    b. How it is measured and assessed
        i. Data collection
        ii. Data analysis
    c. Examples
        i. User happiness
3. Computational Platforms that perform this work
    a. Examples
        i. Some of the most prominent tools
        ii. PSP (Personal Software Process)
4. Algorithmic Approaches
    a. Halstead's Software Physics or Software Science
    b. McCabe's Cyclomatic Complexity
    c. Bug Counting using Dynamic Measurement
    d. Estimating Failure Rates
    e. Reliability Growth Models
5. Ethics around this type of analytics
    a. What ethical concerns mean in this field
    b. Reason why they should be taken seriously
    c. Examples
6. Conclusion
7. Bibliography

INTRODUCTION

The process of software engineering has been vital to the development of the world as we know it right now. It consists of breaking software development work into several parts to improve design, product management and project management. There's quite a few distinct types of such processes, the main one arguably being the agile method currently. There's multiple other methods, such as waterfall,

prototyping, iterative and incremental development, spiral development, rapid application development, and extreme programming.

However, the main question this report aims to cover is **how** do people actually measure a software engineering process. The answer might not be as straightforward as it is with measuring other aspects from different sciences. That might be the case since there are so many facets which can assess the degree to which the process was successful, ranging for example from the sales record of a certain application, and how it measured against the target set before being put into production, to something as simple as getting the most optimal runtime for processes within a certain piece of software.

1. MEASUREMENT AND ASSESSMENT IN TERMS OF MEASURABLE DATA

First of all, it should be clarified what measurable data represents in a process of this sort. One widely used name for it is **software metrics**. A metric of this type is a measure of software characteristics which are quantifiable or countable. They are important for many reasons, including measuring software performance, planning work items, measuring productivity, as well as many other utilisations.

Software metrics are divided into the following main categories and what those categories consist of:

A. Software Requirements Metrics
B. Programmer Productivity Metrics
    - ★ Common measures:
        - Lines of source code written per programmer month.
        - Object instructions produced per programmer month.
        - Pages of documentation written per programmer month.
        - Test cases written and executed per programmer month.
C. Software Design Metrics
    - ★ Number of parameters
    - ★ Number of modules
    - ★ Number of modules called
    - ★ Data bindings
    - ★ Cohesion metrics
D. Management Metrics
    - ★ Techniques for software cost estimation
        - Algorithmic cost modeling
        - Expert judgement
        - Estimation by analogy: useful when other projects in the same domain have been completed.
        - Pricing to win: estimated effort based on customer's budget

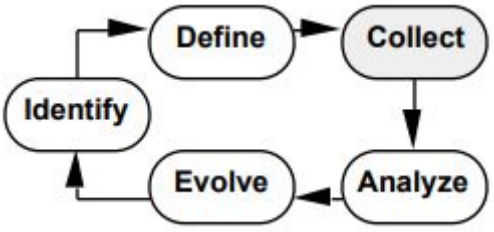
- Top-down estimation: cost estimate made by considering overall function and how functionality provided by interacting sub-functions. Made on the basis of logical function rather than the components implementing that function.

On the one hand, software metrics are beneficial to both companies and software engineers, because it helps managers keep constant track of any potential issue with the software under development or which is already in production, which enables the company to avoid and/or diminish any potential losses, while engineers can use them to increase team productivity by means of efficiently communicating with their coworkers and resolving any issue regarding code.

On the other hand, the main disadvantage regarding these metrics is the current lack of a concrete layout. What is meant by this is that software engineering process measurement is still under development, which means there is not a satisfyingly objective way of interpreting data yet. Thus, one very common way of measuring the quality of code is to count the lines of code written. Even though it is no actual proof of the actual quality of code, it is the main way to quantify code. This, as well, has its own shortcomings, since software developers could be put off tackling tricky problems which would actually require lots of lines of code, while another developer who handles simple pieces of code by writing more lines than their peers might not display the best software development skills (even though productivity would still be considered high).

Subsequently, the next topic represents how the data can be measured or assessed.

The process of data measuring is divided in two main steps: **collecting** and **analyzing** it. For the former, data is collected, recorded and stored. Afterwards, it is validated and the procedures are reviewed for adequacy. Procedures may need to be tailored or adapted to suit each project that implements the measurement procedure. The diagram below better explains the process in question:



Data analysis has the aim of preparing the reports, presenting them to a certain audience and reviewing procedures for accuracy and adequacy. The procedures may need to be updated if the report is not providing insight into the issues, or the report is not understood. In either case, feedback is collected to update the measurement procedures to analyze data.

A question which may arise is what can be done in order to improve the current process. A possible answer lies in the way the audience perceives the analysis process presented to them. They may see area for improvement in certain aspects. Thus, certain decisions which may involve replanning, corrective action, or simply moving on without change would be consequently made. This activity also assesses the adequacy of the measurement process itself. As decisions are made, the audience provides feedback on the adequacy of the data available. They may raise additional issues, or indicate that certain data are no longer available. This feedback, in addition to the feedback from the other activities, is addressed during the next reporting cycle, by reidentifying and redefining the measurement process. This activity overlaps the entire process, and it culminates in the review of the current reporting cycle.

With regard to the whole process, software metrics, as described beforehand, come into place. One relevant example would be represented by agile process metrics. Even though they do not describe the software itself, they are often used to improve the process of software development.

This divides into seven main categories:

1) Software Size - typically represented in story points when agile methods are used. This approach is supported by the decomposition of functionality from a user's perspective--into user stories.

2) Effort and Staffing - must be tracked because they tend to be the primary cost drivers in knowledge-intensive work. Use of agile methods will not change this fundamental fact, nor will it be necessary to make major changes to the mechanisms used to monitor progress.

3) Schedule - traditionally viewed as a consequence of the pace of work performed. In agile development, the intent is to fix this variable, and work to maximize performance of the development team within well-defined time boxes.

4) Quality and Customer Satisfaction - an area where agile methods provide greater opportunity for insight than traditional development approaches tend to allow. The focus on frequent delivery of working software engages the customer in looking at the product itself, rather than the intermediate work products like requirements specifications and design documents.

5) Cost and Funding - structures can be tailored to leverage the iterative nature of agile methods. Using optional contract funding lines or indefinite delivery indefinite quantity (IDIQ) contract structures can add flexibility in planning and managing the work of the development organization.

6) Requirements - often expressed very differently in the context of agile development--in contrast to traditional large-scale waterfall development approaches. A detailed and complete requirements specification document (as defined in DoD parlance) is not typically viewed as a prerequisite to the start of development activities when agile methods are employed.

7) Delivery and Progress - area where perhaps the greatest difference is seen in agile development, compared to traditional approaches. The frequent delivery of working (potentially shippable) software products renders a more direct view of progress than is typically apparent through examination of intermediate work products.

Out of the mentioned categories, one aspect which shouldn't be ignored is user happiness. This is the driving factor behind the success of software ranging from mobile applications to operating systems. It is also the main aim of these pieces of software. Is there a way to objectively quantify it? If so, how can we know whether the filters used accurately measure it? The answer might lie in several different facets:

1. User retention - The percentage of users that open an app on the nth day after the installation.
2. User activity - measured using different metrics:
    - ★ DAU/WAU/MAU - used to quantify how many unique users have opened a certain mobile app during a day/week/month
    - ★ Sessions - overall number of a run app within a certain period of time
    - ★ Average session length - overall length of all sessions for a certain period of time divided by their number
    - ★ Lifetime - defines how many days a person uses an app (from first day used to last one)
3. Audience gain
    - ★ Downloads
    - ★ New users
    - ★ Total users

However you put it, this aspect is one of the hardest to predict, since users themselves aren't aware what would suit their lifestyle. Thus, the small details from conversations or the complaints people make about their daily lives still serve as some of the best quantifiers when it comes to answering what the next big thing in tech will be. Needless to say, it is quite difficult to gather data of this sort on a global scale, but this is more of a marketing issue, rather than a software one.

2. COMPUTATIONAL PLATFORMS WHICH PERFORM THIS WORK

When it comes to software metrics, there is a wide variety of computational tools used to perform certain tasks, some of the most prominent of them including:

- ➔ **Analyst4j** - is based on the Eclipse platform and available as

stand-alone Rich Client Application or as an Eclipse IDE plug-in. It features search, metrics, analyzing quality, and report generation for Java programs.

➔ **CCCC** - an open source command-line tool. It analyzes C++ and Java files and generates reports on various metrics, including Lines Of Code and metrics proposed by Chidamber & Kemerer and Henry & Kafura.

➔ **Chidamber & Kemerer Java Metrics** - an open source command-line tool. It calculates the C&K object-oriented metrics by processing the byte-code of compiled Java files.

➔ **Dependency Finder** - open source. It is a suite of tools for analyzing compiled Java code. Its core is a dependency analysis application that extracts dependency graphs and mines them for useful information. This application comes as a command-line tool, a Swing-based application, a web application, and a set of Ant tasks.

➔ **Eclipse Metrics Plug-in 1.3.6** - an open source metrics calculation and dependency analyzer plugin by Frank Sauer for the Eclipse IDE. It measures various metrics and detects cycles in package and type dependencies.

➔ **Eclipse Metrics Plug-in 3.4** - open source, developed by Lance Walton. It calculates various metrics during build cycles and warns, via the Problems View, of metrics 'range violations'.

➔ **OOMeter** - an experimental software metrics tool developed by Alghamdi et al. It accepts Java/C# source code and UML models in XMI and calculates various metrics.

➔ **Semmle** - an Eclipse plug-in. It provides an SQL like querying language for object-oriented code, which allows to search for bugs, measure code metrics, etc.

➔ **Understand for Java** - a reverse engineering, code exploration and metrics tool for Java source code

➔ **VizzAnalyzer** - a quality analysis tool. It reads software code and other design specifications as well as documentation and performs a number of quality analyses.

One widely known method is PSP (Personal Software Process), developed by Watts Humphrey. It can be described as a "framework" or "guide" on how software engineers can comprehend and improve on their performance.

Some of the ideas proposed by this method include:

- Due to the uniqueness in style of work each software developer possesses, there is a need for a thorough evaluation (including by themselves) of one's strengths and weaknesses and how they can be integrated in the engineering process.
- It is significantly easier to prevent issues than to identify them after they become a problem.
- Also, preventing this sort of issues is significantly cheaper.

- There is a vital need for a sense of responsibility within the community engineers when it comes to the quality of their work, since it ensures the quality of the product.
- Result analysis is required in order to improve on performance.

3. ALGORITHMIC APPROACHES

There are a few algorithmic approaches which are used when it comes to measuring data regarding the software development process. Some of the most important ones are listed and explained below:

1. Halstead's Software Physics or Software Science

   n1 = no. of distinct operators in program
   n2 = no. of distinct operands in program

   N1 = total number of operator occurrences
   N2 = total number 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.)

2. McCabe's Cyclomatic Complexity

   Hypothesis: Difficulty of understanding a program is largely determined by the complexity of control flow graph.

   I. Cyclomatic number V of a connected graph G is the number of linearly independent paths in the graph or number of regions in a planar graph.
   II. Claimed to be a measure of testing difficulty and reliability of modules.
   III. McCabe recommends maximum V(G) of 10.

3. Bug Counting using Dynamic Measurement

- Estimate number remaining from number found.
    - Failure count models
    - Error seeding models
- Assumptions:
    - Seeded faults equivalent to inherent faults in difficulty of detection.
    - A direct relationship between characteristics and number of exposed and undiscovered faults.
    - Unreliability of system will be directly proportional to the number of faults that remain.
    - A constant rate of fault detection.
- What does an estimate of remaining errors mean?
- Most requirements written in terms of operational reliability, not number of bugs.
- Alternative is to estimate failure rates or future interfailure times.

4. Estimating Failure Rates

    - Input-Domain Models: Estimate program reliability using test cases sampled from input domain.
        - Partition input domain into equivalence classes, each of which usually associated with a program path.
        - Estimate conditional probability that program correct for all possible inputs given it is correct for a specified set of inputs.
        - Assumes outcome of test case given information about behavior for other points close to test point.
    - Reliability Growth Models: Try to determine future time between failures.

5. Reliability Growth Models

    Software Reliability: The probability that a program will perform its specified function for a stated amount of time under specified conditions.

    - Execute program until "failure" occurs, the underlying error found and removed (in zero time), and resume execution.

- Use a probability distribution function for the interfailure time (assumed to be a random variable) to predict future times to failure.
- Examining the nature of the sequence of elapsed times from one failure to the next.
- Assumes occurrence of software failures is a stochastic process.

4. ETHICS AROUND THIS TYPE OF ANALYTICS

The topic of ethical concerns covers, on the one hand, the potential for error when it comes to software analytics, in which case the data provided would be biased. Also, due to the current inaccuracy of metrics when it comes to software engineering, there is a higher possibility that such ethical issues would arise. On the other hand, another issue covered would be the diminishing privacy of software engineers and how they conduct their work. This is mainly due to managers having access to these type of software metrics. Even though it is necessary that they have a means of measuring the engineers' work, there is always at least a slight chance that the information might be used in an inappropriate way.

One reason why it is important to keep track of ethical concerns in this respect is that even the slightest bias to pieces of data would result in an inaccuracy of evaluation, thus preventing the development of the best version of software. Also, it is very important to take people's personal values into consideration, since something like a lack of privacy can come in direct contradiction with what is valued by both engineers and users. This would result in straying away from the ideology at the core of software development, which is to make people's lives better.

For example, the selection of data or samples in a way that does not represent the true parameters or circumstances of a process could be an issue. This could be caused either by ignorance or by a certain personal interest. The effect this would have could be either the delivery of an underwhelming piece of software, which would affect both the company's credibility and trustworthiness of the engineer/s, or gain/s for certain individuals, which would just cause the reduction in terms of opportunities for the software world to grow, given the potential public reaction to an event of the sort.

CONCLUSION

In conclusion, the measurement and assessment of the software engineering process, albeit still undergoing development, is of great importance not only to the tech world, but also to the general public. It bears the role of ensuring a correct, unbiased evaluation of the quality of a certain piece of software, while sticking to the values which have helped shape the world of software engineering as one which is

dedicated to making life better for everyone and, thus, overseeing the delivery of the best possible product.

BIBLIOGRAPHY

- http://arisa.se/files/LLL-08.pdf
- http://sunnyday.mit.edu/16.355/metrics.pdf
- https://medium.com/swlh/how-to-measure-the-effectiveness-of-a-mobile-application-23c29c6722cd