Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Tom BOURRET - Measuring software engineering

Table of Contents

Introduction **3**

Measuring the software engineering process **3**

Available computational platforms **5**

Algorithmic approaches **6**

Ethics concerns **7**

Conclusion **8**

Introduction

The objectives of measuring the software engineering process are often business goals, and it is used by managers to help them evaluate some aspects of a project or a system, discover some weakness and confirm or not some hypothesis concerning some evolutions.

Thoses objectives then require objective quantitative values to interpret and compare.

1. Measuring the software engineering process

Measuring the software engineering process consist of describing a software system with a software metric. Using software metrics allow to get some quantitative informations, more or less relevant depending on the situation, about a software system, and use those measable datas to evaluate the system and its evolutions.

https://techbeacon.com/9-metrics-can-make-difference-todays-software-development-teams

There are plenties of possible software measurements, which can be grouped by type :

- **Agile process**
  - *Leadtime* : this metric represent the time between the initial idea and the final software. Reducing the value of this metric means be more responsive on customers needs.
  - *Team velocity* : the number of units a team is able to do in a base of time. It obviously depends on the team experience on the project or in general, but helps to estimate what is feasible in a certain period.
  - *Cycle time* : the time between the changement or the improvement made on the software and the deployment of it. It depends on how the team decide to deliver the versions.
- **Code quality**
  - *Bugs per line of code* : this metric focused on the code quality. It is easy to understand that it has to be as small as possible, and it will probably increase each time a new feature is deployed.

    - *Code coverage* : this metric is regularly used in many companies, and focused on the code testing quality. A well-tested code is less-likely to have unexpected behaviour that end as a bug, but a 100% code coverage doesn't mean the software can't bug.
    - *Comment density* : this metric simply gives the ratio of comment lines in the software. In my opinion, this metric should not be used as a quality measure, but more as a warning if it becomes too small.
- **Modules dependency**
    - *Connascent software components* : this metric quantify the dependency relationships in object oriented design. Two components are connascent if a change in one would require the other to be modified.
    - *Cohesion* : not really a quantitative measure, but it still allows to evaluate the quality of a software. Cohesion refers to "the degree to which the elements inside a module belong together". Basically, a high level of cohesion means less complexity and a more reusable code.
    - *Coupling* : it is the degree of interdependence between software modules. Low coupling is usually a good point for a software, and is often a sign of a well-structured system.
- **Complexity**
    - *Cyclomatic complexity (McCabe's complexity)* : this metric is linked to testing. The cyclomatic complexity of a software correspond to the number of independent tests necessary to test each line of code of this software. Effectively, a software with a high complexity will be more difficult to read, to test and to maintain too.
    - *Halstead Complexity* : this metric is also a complexity, but based on the code himself. Its value depend on the amount of operators and operands in the code.
    - *Maintainability index* : mix of lines of code, cyclomatic and Halstead complexity.
- **Performance**
    - *Instruction path length* : this metric evaluate the number of machine code instructions are needed to execute a program. It might be used notably to improve performance of some algorithms, but doesn't look to be a relevant data for a whole software.
    - *Program load time* : this metric gives a time measure of the loading of the software and what he needs in memory.
    - *Program execution time* : this metric gives a time measure of the execution of the program. It is use notably to evaluate the performance of a program, in benchmarks.

- **Software size**
    - *Program size (binary)* : this metric gives the size of a software. It is not really relevant as an important size doesn't imply a lot of features, as said in the "number of lines of code".
    - *Number of lines of code* : this metric looks very simple in the way it just consists of counting the number of lines of a software (physical, logical, and or/comments). This metric is not often used because it is not relevant, in fact an important number of lines of code doesn't imply more functionalities, it depends on the abilities of the developers.
- *DSQI (design structure quality index)* : DSQI is an architectural design metric used to evaluate a computer program's design structure and the efficiency of its modules. It is recommended to use it on a comparaison basis.
- *Function Points* : this metric is a standard used to evaluate the functionalities provided by a system to a user, in terms of business. It is calculated from users requirements and what the software provide to them.

This list is obviously not exhaustive, but gives some examples of how it is possible to evaluate and measure software engineering on a project.

2. Available computational platforms

The reasons why companies are trying to evaluate their software and its development are mainly businesses reasons : many metrics can be used to measure performance or evaluate an average time to develop which can then give an average cost of development. This explains why companies are so interested in measuring software engineering.

I searched for existing platforms available to perform this work, and found that many companies are making their business on it. Measuring, analysing and counseling.

Here are some examples of companies making software measurement, mostly using some of the previously listed metrics :

- *CAST* is a technology corporation, category and market leader in Software intelligence, founded in 1990. This company provides an expertise in software industries and use some code quality metrics as commitments in projects they are working on. In 2015, they launched CAST HIGHLIGHT, a Software as a service platform to track software health, cloud readiness, complexity and cost of

software portfolio using predictive pattern analysis. This company also works in research to improve their methods and develop new metrics or ways to use it.

- The *CISQ (Consortium for IT Software Quality)* is not a company, but is linked to CAST in that Bill Curtis, the CISQ's first Executive Director is also the CAST's Scientist Director. The goal of CISQ is to develop international standards for automating measure of software size and quality and helps to develop a market based on it. It has been launched in 2009.
- Many companies use the "Function Points" metric, it is the case of Premios. Premios' teams work with teams of developers to evaluate their projects and working to improve this value.

3. Algorithmic approaches

There are many algorithmic approaches to measure software engineering, with both strong and weak points. Here are some of the algorithms I found.

The first algorithm I found is *COCOMO (Constructive Cost Model)*. This model is use to estimate effort, cost, and schedule for software projects, based on previous projects. Basic evaluation uses software size, with source lines of codes metric, and takes the team's size and experience into account. This result only gives a quick estimate of software costs.
A more complete version of Cocomo ("intermediate") also computes four "cost drivers" which are product attributes, hardware attributes, personnel attributes and project attributes. Each attribute is subdivided and each subdivision get a rating, and those ratings are used to compute the effort value.
A final version of Cocomo, called "detailed", does the same than the intermediate one, but evaluating each attribute in each steps of the software process to be as precise and complete as possible.

One of the successor of Cocomo is the *Weighted Micro Function Points* (WMFP), which measure different or updated elements than Cocomo. For example, it uses flow complexity and object vocabulary, which are respectively more complete versions of cyclomatic complexity and Halstead vocabulary. It aso measure others metrics like data transfer, code structure (the time passed to separate code into classes and functions), comments, etc… And it finally calculate a time with those parameter and a formula, and this resulting time is then multiplied by an average developer cost per hour to produce an average software cost.

4. Ethics concerns

Finally, there are a lot of ways to measure software engineering through different projects, and many reasons to do it. In most cases, it is business reasons : the objective is to improve a software, make it better and more profitable, easier to maintain and develop, etc…

But measuring performances during the development of a project has many counterparts. Developing a software is different than building a house or a car. It's a complex task, some parts of a project can be much more time-consuming than expected because of some unexpected problems to face. Some software failures can also be critical for the people who will use it once completed, and it could represent a lot of pressure for developers, like softwares used for constructions (bridges, houses, etc…). Some mistakes in those kind of software can end with many people injured or die. In such cases, spending more time seems to be normal, but it implies less productivity, and more pressure on the developer.

There is also an important impact which must to be taken into account : if some metrics are used to evaluate some developer's work and that impacts his working conditions (rewards, manager's discussion, or else), the developer will try to work with those metrics in mind. It could firstly seem to be normal, and even a good thing, because those metrics exist for some reasons. But knowing the metrics, a developer could be more focused on "how to improve the metrics value" than the work they are doing. The result of such a choice might reveal better measures about the project, but it is more likely to let some errors or not complete code, because of a lost of focus on the real important points.

And this is, according to me, the most important ethical question software engineering measurement raised : what should be measured, and how should we use software metrics to really improve the way we create softwares ?

In any projects with software engineering measures, metrics should only be used wisely, and only to answer specific questions raised earlier in the process. Metrics should be used to understand, evaluate and find weaknesses. It might be pretty obvious that any metrics is not relevant in every situation, and it is a priority to choose those metrics depending on the objectives of the project.

Also, using metrics to evaluate a project must not be just a basic comparison of numbers. If some metrics vary much more than expected, either up or down, this should be study to understand what happened : why does it happened ? Is it normal ? Is it

good or bad ? Is is just punctual or will it occur again ? What impact could it have on the business ? And probably a lot more questions… A variation on any metrics could come from many reasons, and even if it is bad for the project, team members must not be fear of those result. The metrics results must be used to understand, evolve and correct potential mistakes, not punish a worker.

Conclusion

With the omnipresence of applications and softwares in every jobs and moments of our life, it is normal to develop some new ways to improve smartly the projects we are working on. But those methods must be used with the benefit of hindsight. The measures which can be made through the evolution of a project must serve as guides to have a better understanding of the environment of this project, to evolve in the best direction and be as reactive as possible with any kind of problems the team could have to face.