1. Introduction

A software engineering task can be roughly divided into a few components,

- Design
- Coding
- Testing
- Release

Rather than being a sequence, this list is more of a connected graph as a software project is never really complete until the project is abandoned.
The process consists of a collection of goals and a collection of resources that will be used to achieve that goal, these goals and resources are subject to change over time.

When seeking to optimize the development of the project it is beneficial to collect data at various stages of the project as well as on the different levels of the project and in order to do such a collection of metrics must be chosen in order to record the data. This is a difficult task as, in general, there isn't a canonical set of metrics that will best capture the ideal data to optimize the project. Section two of this report will survey a few of the metrics commonly used in the industry when surveilling a project.

Once a collection of metrics have been chosen and data is being collected the next step is to seek ways to improve the performance of the development process, part three of this report will go over some of the computations that can be performed on the data.

Finally, part four will go over some of the ethical issues that arise when taking part in a software engineering project.

2. How one can measure engineering activity

In order to measure software engineering progress, it is necessary to use metrics at every stage of the development lifecycle. The goal of using these metrics is to obtain objective, reproducible and quantifiable measurements which can be used for planning, return-on-investment analysis, quality assurance, and testing, code complexity optimization, and optimal task assignments.
The following are some examples of metrics that can be used to measure a software engineering project at various stages of the development lifecycle [1].

- Formal code metrics: metrics on the code produced so far such as the number of lines of code per individual, time and space complexity, etc.
- Developer productivity metrics: metrics on the developers and engineers such as the amount of time working on a project, frequency of uploads and modifications, etc.
- Agile process metrics: metrics on the team as a whole such as Lead time: how long it takes you to go from idea to delivered software, Cycle time: how long it takes you to make a change to your software system and deliver that change into production, Team velocity: how many “units” of software the team typically completes in an iteration, and Open/Close rates: how many production issues are reported and closed within a specific period. The general trend matters more than the specific numbers.
- Operational metrics: metrics that measure how effective the team are at handling errors and issues that occur in the project such as Mean Time Between Failures (MTBF) and Mean Time to Recover (MTTR).
- Test metrics: metrics that measure the degree to which the project has been tested such as the number of automated tests, the amount of the code that has been tested, etc. This metrics is associated with the quality of the final product.
- Customer satisfaction: Once the project has been released to the public this metric measures the customer feedback such as Net Promoter Score (NPS), Customer Effort Score (CES) and Customer Satisfaction Score (CSAT).
- Security: these metrics measure how secure the app is to exploitation and penetration, as well as how efficient the team is at responding to these problems.

Some of these metrics are outdated such as the formal code metrics as due to the increase in computing power over the past few decades resources such as space have become abundant, and depending on the style of development some of the metrics will not be used. The Wikipedia page on Software Metrics has an extensive list of metrics developed over the years [2].

3. Performance Analytics

Below we list and discuss several platforms for analyzing and recording data from the metrics mentioned in the last section.

- Formal code analytics: This aspect of measuring software development is difficult as in order to measure any aspect of the code the software analysis tool will have to determine whether the program halts or not which is impossible in the general case. The tools for this form of analysis can be divided into two categories, static analysis, and dynamic analysis. Static analysis is the process of measuring features of the code by analyzing the source code of the program. This usually takes place before testing. Each language has its own static analysis tools but for python specifically, there is PyCharm, PyDev, Pylint, and Semgrep. Dynamic analysis is the collection of techniques used to test running applications. Examples of tools used for python are CrossHair, icontract, Scalene, and typo.
- Developer productivity analytics: The tools used for developer productivity are usually under the header of project management tools. There are quite a few tools for project management and some well-developed version control systems have project management tools that track pull and push requests, branching, number of contributors, frequency of contribution, etc.
- Agile process analytics: Tools used for agile process analytics are pieces of software that usually come equipped with, boards, backlogs, roadmaps, multiple type of report charts and many more features. Some examples are Trillo, JIRA, and Zepel.
- Operational analytics: I was unable to find ant automated tools for operational analytics although several companies offer services to do this for you for example sleek.io.
- Test analytics: There are several options for automated test analytics such as Avo Assure, Kobiton, TestProject, Subject7 etc. Some of these tools are code free and can be used by people who are part of the project and have no coding experience. Unit testing could also be considered as a medium for testing analytics.
- Customer satisfaction analytics: This topic is more traditionally related to business analytics as the usual media can be used such as surveys, ratings, interviews.
- Security analytics: Some companies specializing in information security offer audits with feedback in the form of security analysis, such as McAfee.

4. Computation on SE Data

Once measurements have been obtained through realized applications of the various software metrics considered the next question is how to use these measurements to make decisions about how to optimize the software engineering endeavor on all levels of the process. A scientific approach can be applied in this context as various hypotheses on how to improve an organization can be created, tested, improved, and reiterated.
On the human level, the supervisor of the data gathered can clearly see which persons are performing most of the work on the project in terms of contributions and effort spent, so by reducing the number of resources given to the less effective contributors and the project can optimize its cost. The team could also offer incentives for increased productivity and innovation but since there is a limit to the innovation caused by profit incentives it is up to the supervisor to calculate the correct costs. With the given data is might actually be a realistic idea to consult an economist.
It’s possible given the large body of data collected that a statistical approach could be utilized. For instance, a correlation of the value of several variables with the level of activity could be discovered and accounted for.
You can identify where the team’s strengths and weaknesses are, and try to create methods to improve the overall performance.
You can use the data to see who works well together within the team and make changes accordingly.
If you have access to the data of other software projects of similar goals to your own you could use machine learning to try and approximate various metrics such as the amount fo time needed and the cost.

5. Ethics

There are many ethical issues when it comes to measuring and analyzing a software engineering project.[5]

- Privacy: The most obvious one is about developer privacy, the developers involved in the project will, during the lifecycle of the project and possibly indefinitely, have to consent to their surveillance in order to avoid invasion of privacy. The recording of the developer's information is a concern as the supervisor of the project could (either internationally or unintentionally) record personal information such as name, date of birth, location, etc.
- Accuracy: When it comes to fixing errors in a project the developer becomes a part of an ethical dilemma as every fix to a part of a system comes with a probability of introducing new errors so they are faced with the decision to leave the error in or cost the software a faster lifecycle of updates, patches, and disclaimers. Another issue is the accuracy of the data used on the developers themselves such as socioeconomic and biological indicators, as incorrect data can cause biased points of view to be developed and consequently have a negative impact on the members of the team. Another issue is team communication, specifically differences in native language as this can cause miscommunication costing the project’s efficiency.
- Property: A major concern in the current economic environment is authorship and ownership of the project being developed. The traditional view of property rights in software development is that the software base is owned by the organization mostly involved in its development, but has the potential to change depending on several factors such as if the project is open source, or if the project has the potential to generate a large profit.
- Accessibility: Another ethical issue is access to the software systems being developed. If the project is connected to the internet, a reliable internet-connected hosting part must be present. A body of people must be available to make sure the hosting of the system is well maintained. Regional policies must also be taken into account as the content provided by the software system might violate certain laws in different regions. Another consideration is the usability of the system. If the system is very complex not a lot of people will have access to it due to its steep learning curve, so good well-maintained documentation should be provided.
- Maintenance: Finally as both software and hardware march forward into the future the system will need to be updated and potentially made compatible make compatible with systems in order to provide ongoing support.

6. References

[1] “Top 5 Software Metrics to Manage Development Projects Effectively.” *SeaLights*, https://www.sealights.io/software-development-metrics/top-5-software-metrics-to-manage-development-projects-effectively/. Accessed 27 December 2021.
[2] Software Metrics, https://en.wikipedia.org/wiki/Software_metric
[3] Simplify your Python Code: Automating Code Complexity Analysis with Wily, https://towardsdatascience.com/simplify-your-python-code-automating-code-complexity-analysis-with-wily-5c1e90c9a485
[4] static analysis (static code analysis), https://whatis.techtarget.com/definition/static-analysis-static-code-analysis
[5] https://www.srs.fs.usda.gov/pubs/VT_Publications/01t7.pdf