John Davis Carbeck
CS3012

Measuring Software Engineering: *A look into methods and problems of measuring and assessing data from the software development process*

Introduction

The development of software is both a mechanical and creative process. It requires that developers are knowledgeable about the tools they are using but also creative in the way they use them. This sets software development apart from most other fields. How can data be collected and interpreted to measure and analysis this two pronged process?

Software development is becoming increasingly integrated in how the world operates. It is forecasted that in 2022 the global software market will exceed $500 billion. Without real data and information, software development decisions are made based on gut feeling or experience. Inaccuracies in decision making lead to wasted resources and increased cost of building and maintaining large software systems. This is why the measurement software development has been researched heavily. Many tools and utilities have been developed to measure the software development process in hopes of providing information that will improve the decision making by senior software engineers and the evaluation of performance of developers. A correct implementation of software development metrics, offers the saving of millions in development costs as well as the production of more secure and efficient software that is easier to maintain.

Large pieces of software are created by teams of developers, this collaboration means that code must be shared and managed by repositories and version control systems. These tools have historically been the main focus for the collection of data on software development. In recent years the focus has shifted into all aspects of the development process, from the clicks and keystrokes during the editing of code in IDE to the discussion and meetings of software developers in a team to even the analysing of personality profiles to build teams with better chemistry for agile development. There is an incredible amount of data being collected on all aspects of software development process, but there remains practical and ethical problems with how this data is collected and interpreted.

This paper will discuss the measurement of software development, determining what impacts software thus what need to be measured, the methods of collecting information from repositories of data, the software tools available to provide measurement and analytics on software, and the pitfalls of measuring software development from improper implementations of measuring software development.

Quantifying software development

The methods for analysing software development are based on the ability to collect data on specific parts of the development process. Different utilities and tools focus on

different areas of development, and thus the methods used for the collection and interpretation of theses areas can be vastly different. Each technique or tool provides some insight into software development process.

Factors that affect software engineering and how to quantify them

There has been significant research done on the practice of measuring development. The field of software development is quite young, only being around for 60 odd years. Since the beginning, there has been great effort made to understand how the development process is affected by external factors and then how the process can be improved through identifying trends in these measurements. These studies provide an overview of the factors that need to be addressed.

1970-1979

Walston and Felix analysed in 1977 in one of the first larger studies to determine the factors correlated to programming productivity[1] (measured in effort per SLOC, source line of code). Many other studies of the time focused on lines that of code. Since the 70s a lot has changed with regard to how software is developed, such as team structure and program structure, and significant amount of what was measured has less impact on software development today, but analysing lines of code has remained a cornerstone in measuring coder productivity.

1980-1989

W. D. Brooks[2] used factors from Walston and Felix as the basis for his study at IBM. In his paper, *Software technology payoff: Some statistical evidence*, he identified program complexity and structured programming to be important to how maintainability of software. DeMarco and Lister changed what many were measuring in software development with their paper *Peopleware. Productive Projects and Teams*, switching from lines of code and function point centered researched to looking at the sociological soft factors that impact software development. They identified turnover being a central factor influencing software engineering. They also mentioned the workplace environment having an impact on productivity, calling for a need for proper windows, natural light, quietness, etc.[3]

1990-1999

The 90s show a stronger interest in soft factors that impact software development. Brodbeck discussed in his paper, *Communication and performance in software development projects*[4], the measuring the communication and how that impacted the end product from the studied development team. He showed that projects with a higher communication effort were more successful. He also showed that the intensity of internal communication had a strong positive correlation with project success. In Chatzoglou and Macaulay's study consisted of interviewing over a hundred software project to determine factors that affect productivity.

---

[1] S.Wagner and M.Ruhe. A structured review of productivity factors in software development. Technical Report TUM-I0832, TU Mu ¨nchen, 2008.

[2] W. D. Brooks. Software technology payoff: Some statistical evidence. *J. Syst. Software*, 2(1):3–9, 1981.

[3] T. DeMarco and T. Lister. *Peopleware. Productive Projects and Teams*. Dorset House Publishing, 1987.

[4] Felix C. Brodbeck (2001). Communication and performance in software development projects. *European Journal of Work and Organizational Psychology, 10*, 73-94.

They found the experience, knowledge and persistence of the team members, as well as communication, resources available, tool and techniques, and the management style are important factors in the development time and quality of software[5].

2000-2018

Recent research identifies that technological and sociological factors impact software development. Jones shows in his paper *Software Assessments, Benchmarks, and Best Practices* that software projects are impacted by about 250 different factors, and that individual projects are impacted by about 20 different factors. These factors were identified by case studies with some containing quantitative results[6]. In Berntsson-Svensson and Aurum's paper *Successful software project and products: An empirical investigation*[7] surveys were conducted to determine the factors influencing projects and product success. They determined that different industries define success in different ways, but found that influencing factors for reaching those successes are similar, such as: well defined project scope, complete and accurate requirements, good schedule estimations, adequate personal.

Utilities and Tools

In order to address the factors that are affecting software development, they must be measured in close to real time and reported effectively. By quantitatively analysing these factors, senior developers don't not have to speculate and instead point to real information, much like other disciplines to make decisions.

Due to computers being the platform that software development is conducted on, much of the process can be recorded in real time. Software data that is recorded can be broken down into 3 main types of repositories of information:

- Historical repositories: source control repos, bug repos, communications about the evolution and progress of project.
- Run-time repositories: deployment logs that contain info about the execution and usage of software systems.
- Code Repositories: sourceforge.net, google code, and github cobtain source code of various software systems developed by a team of developers

Runtime repos can be used to pinpoint execution anomalies by identifying dominant execution or usage patterns across deployments, determining deviations in the patterns. Code repos can identify dominant and correct framework or library usage patterns. Historical repositories have the largest amount of measurable information. The information contained in them is used to determine team efficiency, health of the software product, and individual performance and contribution.

---

[5] P. D. Chatzoglou and L. A. Macaulay. The importance of human factors in planning the requirements capture stage of a project. *Int. J. Proj. Manag.*, 15(1):39–53, 1997.

[6] C. Jones. *Software Assessments, Benchmarks, and Best Practices*. Addison-Wesley, 2000.

[7] R. Berntsson-Svensson and A. Aurum. Successful software project and products: An empirical investigation. In*Proc. ISESE '06*, pages 144–153. ACM Press, 2006.

How data is Mined

The information that is collected from repositories are passed through mining algorithms to determine patterns and possibly linear correlations that allow for the creation of metrics that are then used to analyse development .

Software Engineering Data	Mining Algorithms	Software Engineering Tasks
Sequences: execution/static traces, co-changes etc.	Association rule mining, frequent itemset/subseq/partial-order mining, seq matching/clustering/classification, etc	programming, maintenance, bug detection, debugging, etc.
Graphs: dynamic/static call graphs, program dependence graphs, etc	Frequent subgraph mining, graph matching. Clustering/classification. etc	Bug detection debugging, etc
Text: bug reports, emails, code comments, documentations, etc	Text matching/clustering/classification etc.	Maintenance bug, detection, debugging etc

Association rule mining[8]--a procedure which is meant to find patterns, correlations, associations, or casual structures from data sets found in relational databases, transactional databases, and other data repositories.

- AIS: targets the discovering of qualitative rules and generates significant association rules between items in the database. The candidate items are generated on passes of examined databases.
- SETM: allows for the use of SQL to compute large itemsets. The STEM algorithm generates the candidate itemsets by scanning database. The STEM algorithm separates candidate generation from counting by using the SQL join operation for candidate generation.

Association rule mining helps with determining bugs, programming contributions, and points of maintenance. Association rule mining is a general data mining method used in other fields like business, and consumer goods.

Frequent subgraph mining[9]--a way of discovery of graph structures that occur over a significant number of times across a set of graphs

- gSpan: complete frequent subgraph mining, improves performance over extensions to graphs through DFS representation and aggressive candidate pruning
- SUBDUE: an approximate frequent subgraph mining, uses graph compression as metric for determining a “frequently occuring” subgraphs

[8] Pallavi V. Nikam, Deepa S. Deshpande, "New Approach in Big Data Mining for Frequent Itemset Using Mapreduce in HDFS", *Convergence in Technology (I2CT) 2018 3rd International Conference for*, pp. 1-5, 2018.

[9] "T. Ramraj & R. Prabhakar" ("2015"). "Frequent Subgraph Mining Algorithms – A Survey". *"Procedia Computer Science", "47"*, "197 - 204".

- SLEUTH: complete frequent subgraph mining, built specifically for tree data structures

Frequent subgraph mining is used for determining bugs or faulty part of code. The graphs are constructed by looking at program flow. From the graphs the algorithms can determine flows of execution that identify faulty code.

Clustering[10]: divides the population of data points into a number of groups such that data points are in groups are more similar to those in the group to those in other groups

- Connectivity models: models based on the notion that data points closer in data space are more similarity to each other than data points lying farther away. They start with classifying all data points into separate clusters and then aggregating them as the distance decreases. Second approach, all data points are classified as a single cluster and then partitioned as the distance increases.
- Centroid models: These are iterative clustering algorithms in which the notion of similarity is derived by the closeness of data points to the centroid of the clusters. K-Means clustering algorithm is often used. The no. of clusters must be specified before hand.
- Distribution models: thest clustering models are based on the notion of how probable is it that all data points in the cluster belong to the same distribution, Normal, Gaussian. Sometime suffer from overfitting

Clustering is used in almost all aspects of measuring software, it allows for the parsing of large sets of data to determine metrics.

Interpretation and Metrics

Inorder for conclusions from data mining to be drawn the metrics that are being used need to follow strict guidelines to make sure that the conclusions are valid. The IEEE Standard 1061 gives a methodology for developing metrics for software quality attributes, where an attribute is a "measurable physical or abstract property of an entity" and a quality factor is a "a management-oriented attribute of software that contributes to its quality". A software metric is "a function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software posses a given attribute that affects its quality". The standard lists validation criteria for software metrics:

- Correlations: there should be a linear correlation between the metric and the quality factor
- Consistency: That the quality factor will result in the same output of the metrics function
- Tracking: The output of the metrics function should change promptly with the change of input data

[10] Kaushik, Saurav, and Saurav. "An Introduction to Clustering & Different Methods of Clustering." *Analytics Vidhya*, 10 Dec. 2016, www.analyticsvidhya.com/blog/2016/11/an-introduction-to-clustering-and-different-methods-of-clustering/.

- Discriminative: The metrics can discriminate between high-quality software components and low quality components, thus the metric values associated with higher quality software components must be higher than that of lower quality software components.

Overview of Metrics

- *Connascent software components* - two software components are connascent if a change in one component would require the other to be be changed to keep the overall correctness of the system. Connascence is a way to characterise and reason about certain types of complexity in software systems.

- *Constructive cost model - (COCOMO)* is a cost estimate model for software projects that is used to determine project costs. Based on evaluating lines of code, product attributes, hardware attributes, and personal attributes

- *DSQI* - is a software architecture metric used to evaluate a program's structure and efficiency of its modules, the result of the metric is a value between 0 and 1, the closer to 1 the higher the quality of the architecture.

- *Halstead Complexity* - measures a program complexity directly from source code, emphasis on computational complexity. This metric is calculated from the number of distinct operators, distinct operands, operator instances, and operand instances.

Overview of software

- *Personal Software Process* is a framework that helps developers with software work. PSP uses methods, forms, and automated scripts to discern how software engineers should approach their work. The goal of using PSP is to develop a zero defect product on schedule with planned costs.

- *Hackystat* is an open source framework for the collection, analysis, visualisation, and dissemination of the software development process. Hackystat uses sensors that attach to development tools which collect data about the development to a data repository. The repository then can be queried to provide high level abstraction on the data that has been collected.

- *SonarQube (sonar)* is a platform for realtime inspection of code quality. SonarQube detects bugs, code smells, and security vulnerabilities on a set of programming languages. This is done by integrating SonarQube with IDEs such as Eclipse, Visual Studios, and IntelliJ IDEA. SonarQube also integrates into external tooks like LDAP, Active Directory, and github. SonarQube is open source and is expandable with plugins

Implementation of Methods and Tools to gain insight

Here we will look at an example demonstrating the use of software to provide metrics to ultimately draw conclusions on a impacting factor to software development.

One implementation for quantifying software development is measuring the use of development tools and libraries. This is researched in the paper, *Towards recognizing and rewarding efficient developer work patterns*[11]. This paper focused on measuring the use of software development techniques that provided more efficient development and determining a way of sharing that information to increase the use of good techniques with other developers.

The information was collected by a system such as Mylyn Monitor, HackStat, or PROM. The systems collected command and clock stream data from developers, this was necessary to record the use of commands issued while editing the code and outside of the environment. Mylyn Monitor captured the commands issued in the IDE, HackyStat captures the command events in multiple tools using plug-ins and PROM collects commands performed in the IDE. For the analyzing of patterns of developers, Markov chains were used to track steps taken during a development session. Using JUBL the researchers were able to describe the usage model and to calculate the occupancy and frequencies of the model.

The researchers highlighted the differences between the patterns of development by selecting two developers, one who used structured navigation to find the bug and one that used unstructured navigation and browsing more. The researchers were able to determine the reduction of steps and thus higher productivity, here are the following results:

Developer	Searcher	Browser
Navigation nodes	17	27
Navigation arcs	37	89
Average sequence length	6.3	15.5
Sequence length standard deviation	2.5	8.7

The information collected demonstrates how the use of structured navigation is more effective in the isolation of bugs in fewer steps. The Browser had more than twice the average number of steps per session, but with also less consistency with the number of steps taken for each navigation session. By using this model of collection and evaluation other aspects of the development process can be measured and improved by recognizing more efficient uses of tools. Once these efficient patterns are determined they can then be implemented into the workflow of developers and it can be tracked and developers can be given a relative mark based on other developers.

[11] W. Snipes, V. Augustine, A. R. Nair and E. Murphy-Hill, "Towards recognizing and rewarding efficient developer work patterns," 2013 35th International Conference on Software Engineering (ICSE), San Francisco, CA, 2013, pp. 1277-1280.

Problems with quantifying development

The practice of measuring software does have flaws. When measurement and interpretation is implemented incorrectly it not only provides false information but it impacts the evaluation of developers. The pitfalls of measuring software development lie in the tradeoffs of software, data used to make measurements, and the ethics of conducting measurements

Design problems

In the paper *Searching under the streetlight for useful software analytics*[12], Philip Johnson discusses the inherent design tradeoffs involved in the collection of information on software development. He argues that these tradeoff are between automation, adoption barriers, and the generality of the analytics that the measurement can provide. By looking at existing ways of measuring software Philip develop the following graph.

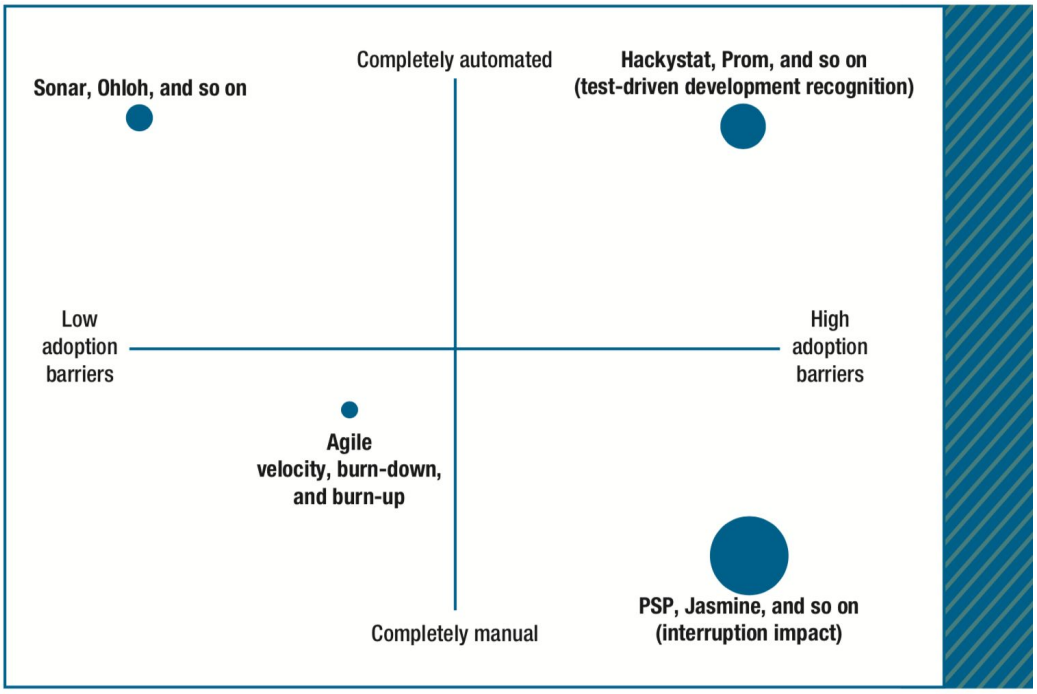

FIGURE 6. A classification for software analytics approaches, including automation, adoption barriers, and the breadth of possible analytics the approach supports (indicated by the circles' size). In the parentheses are analytics that would be difficult to implement with techniques or technologies in the other quadrants.

Due to the tradeoffs of measurement Johnson believes there is no single best approach, or that newer approach improve upon techniques that have previously been developed. Therefore in order to effectively measure development the tradeoffs between tools must be addressed and the ones that address the specific information that need to be measured must take priority. Thus measuring software requires the use of many different tools.

---

[12] Johnson, P. (2013). Searching under the Streetlight for Useful Software Analytics. *IEEE Software*, 30(4), pp.57-63.

Data problems

With the measurement and extrapolation of anything the quality of the data that is used is very important. Problems lie in how the data is collected and what it is collect from. Self reporting of data is problematic as the information can be falsely reported to make the developer reporting it look better than his true contributions. Also depending on what has been collected, the information might not identify a underlying issue or problem, thus leading those measuring software to overlook something or not focus on it.

Each source of information brings with its own problems. In the paper *The Promises and Perils of Mining GitHub*[13] the researchers identified problems with using github repositories as a source of creating software metrics and doing general software management.

I. A repository is not necessarily a project
II. Most projects have very few commits
III. Most project are inactive
IV. A large portion of repositories are not for software development
V. Two thirds of projects (71.6%) are personal and have no development team
VI. Only a fraction of projects use pull requests. And of those that use them, their use is very skewed
VII. If the commits in a pull-request are reworked GitHub records only the commits that are the result of the peer-review, not the original commits.
VIII. Most pull requests appear as non-merged even if they are actually merged
IX. Many active projects do not conduct all software development in GitHub

These issues exist in similar form in other repositories. It is essential that these problems are taken into consideration in order to no use bias data. It is important to question the quality of data that is being used to measure software as not all metrics and methods will work out of the box.

Ethics

With the collection of any data comes ethical problems. It is important that the developers using the software that is measuring them know exactly was is being collected and how it might be interpreted. The use of measurement causes effects to how a team might operate. In the case where developers are being evaluated on their delivery time of software products, they might begin to overestimate the amount of time need for a project so the they don't end up with a poor measurement. Another ethical problem is when certain aspects of development are focused for measurement the development team will spend more time on those components and could possibly neglect other aspects of the project leading to a net negative impact from measuring development. The practice of measuring software does provide great insight and often can allow for increased development and happier customers,

[13] Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. and Damian, D. (2015). An in-depth study of the promises and perils of mining GitHub. *Empirical Software Engineering*, 21(5), pp.2035-2071.

but with improper use it will be more harmful to the development team and the health of the project than advantageous.

Conclusion

Software development allows for automatic collection of data and thus can provide real time information for development teams. By collecting information from repositories of data associated with development, software tools and measurement methods can parse the data into metrics to give feedback to the developers and management. When ethical, data, and design problems are addressed, the information can provide insight to the numerous factors that impact software development. These factors when mitigated allow for faster more secure and stable products.