Measuring Software Engineering

CS3012 Report | campbel2@tcd.ie | Lee Campbell
November 2018.

Introduction:

In this report, we will explore the ways in which the software engineering process can be measured and assessed in terms of measurable data. We'll then examine various computational platforms available to perform this work, the algorithmic approaches available, and the ethics concerns surrounding this kind of analytics. In order to provide context and understanding for these metrics we'll also take a look at the different ways the software engineering process can be modelled.

Why measure software engineering? :

Software engineering typically operates within the context of business, as such the desire to limit missed deadlines, overspend and flawed products are some of the many obvious reasons people seek to measure the software engineering process. Of course, anyone that has even been remotely involved in some aspect of the software development process will know how notoriously difficult it is to measure. This is in large part due to the fact that "Software entities are more complex for their size than perhaps any other human construct because no two parts are alike" [Brooks, 1987]. Software development is analogous to manufacturing, except that we don't make the same identical widgets over and over. We can't just measure for defects, reject some products, and ship the others. In this respect, software systems differ profoundly from computers, buildings, or automobiles, where repeated elements abound. In fact, repetition within source code is often a sign of lazy or bad code structure. [Brooks, 1987].

Modelling the software engineering process

The Software Development Life Cycle (SDLC) is an ISO/IEC 12207 accredited process used by the software industry to design, develop, test and maintain high-quality software. The SDLC is designed to produce high-quality software that meets or exceeds customer expectations and that reaches completion within time and cost estimations [Tutorialspoint, 2018]. A typical Software Development Life Cycle consists of the following stages:

1. Planning and Requirement Analysis
2. Defining Requirements
3. Designing the Product Architecture
4. Building or Developing the Product
5. Testing the Product
6. Deployment in the Market and Maintenance

Within the Software Development Cycle methodology there are various types of models used, which follow a series of unique steps to ensure the success of the software development project. Some of the most popular models in the industry include:

1. Waterfall model
2. Agile model
3. Spiral model
4. V model
5. Big Bang model

The chosen model has a significant impact on the metrics and means by which the process can be measured. The variations in these models are indeed one of the many factors which make having a unified measurement approach difficult. To give us a broad sense of what's out there, we'll focus on the Waterfall and Agile models below, two very different but popular approaches.

Waterfall model

The waterfall model emphasizes that a logical progression of steps be taken throughout the Software Development Life Cycle (SDLC), much like the cascading steps down an incremental waterfall. It is based on a linear six-step sequential flow, meaning that each phase must be completed before the next phase can begin and there is no overlapping in the phases [Powell-Morse, 2016]. It was the earliest and first popular SDLC approach to software engineering. The six sequential phases in the Waterfall model are:

1. **Requirement gathering and analysis** − in this first step all possible requirements of the system are gathered and specified in a Software Requirement Specification (SRS).
2. **System design** - The next step in the process is to design the overall system architecture. In turn, figuring out the necessary hardware requirements to support the software.
3. **Implementation** − Once the framework is defined, the system is broken out into small unit programs. These units are then developed and tested using unit testing.
4. **Integration and Testing** − All the units developed and tested in the implementation phase are then integrated together to build the entire system. Once this integration has occurred the system as a whole is tested.
5. **System deployment** − Once the functional and non-functional testing is completed; the product is either deployed into the clients existing environment or released into the market as a standalone product.
6. **Maintenance** − The final stage in the process is to carry out maintenance on the deployed software to resolve bugs or improvements discovered when used within the client's environment.

The waterfall method is simple and easy to manage due to the rigidity of the model. Its simplicity allows for a strict check against the original requirements and so it works particularly well for smaller projects where the requirements are well understood. However, in spite of going having an explicit testing phase built into the model, as discussed above, this testing is often too little, too late. If a test in stage five reveals a fundamental flaw in the design of the software it often requires a huge leap backwards in terms of stages, which can be devastating to the project. It also ignores user and

client feedback during the development stages which can prevent costly re-developments at the end of the project. Non-adaptive design constraints and lack of client feedback are some of the many reasons the Waterfall approach is less suitable for long-term projects where requirements are at a even a moderate risk of changing [Tutorialspoint, 2018].

Agile model

In recent years, the Agile model has continued to grow in popularity and take over the more traditional Waterfall model. Agile is based on adaptive software development methods, whereas traditional SDLC models are based on a predictive approach [Tutorialspoint, 2018]. Agile is a combination of the iterative and incremental process models; it involves Planning, Requirements Analysis, Design, Coding, Unit Testing and Acceptance testing like in the Waterfall model except instead these steps take place for every increment. Agile delivers a working product at a much earlier stage while allowing for greater adaptability and in turn greater customer satisfaction. The model is based on a number of core methodologies:

1. **Individuals and interactions** − Agile encourages self-organisation and motivation in conjunction with co-location and pair programming.
2. **Demos** − In an Agile environment, working software demos are key to getting feedback and understanding the customer's requirements further.
3. **Customer collaboration** − As the requirements cannot be compiled in their entirety at the beginning of the project due to various different factors, continuous customer interaction is very important to get the correct product requirements.
4. **Responding to change** − Agile development is adaptive and should be able to make quick responses to change and support continuous development.

Where the Waterfall model lacks, Agile steps up. The model gives a very realistic approach to software development in which functionality can be developed and demonstrated rapidly. It enables concurrent development and delivery within a structured process that has the customer at the centre. While the model delivers for the customer through constant engagement this can be problematic and drive the team in the wrong direction if the customer is not clear what they want [Tutorialspoint, 2018].

Nonetheless Agile is increasingly becoming the go-to model in the modern software industry.

Metrics for models

“You can’t control what you can’t measure.” This is how Tom De-Marco describes the importance of software metrics in his book ‘Controlling Software Projects’ [DeMarco, 1982]. Proper measurement of software metrics is essential to an effective software development lifecycle (SDLC). Control is the extent to which a project manager can ensure software project stays on track in terms of time, costs and requirements. Measurement is a recording of past effects to quantitatively predict future effects. [SoftwareTestingGenius, 2018].

While the metrics deployed depend greatly on their suitability for the particular SDLC model in use; at a high level, a metric is just a measurable indication of some quantitative aspect of a system. Of course, there are also a number of qualitative metrics which are equally important such as customer satisfaction and employee motivation.

Metrics within Software Engineering can be grouped into three closely interdependent categories:

1. **Product metrics** - analyse the project in terms of size, compatibility, design features, performance and quality characteristics. E.g. Kilo lines of code (KLOC).
2. **Project metrics** - these type of metrics look at the team's ability to deliver a project on time and within resource constraints. E.g. Number of developers.
3. **Process metrics** - are used to improve the efficiency of an existing process used in software development and maintenance. E.g. Defect density.

In order to get a broad sense of the different metrics which can be deployed from each of these categories, we’ll look at them in terms of the Waterfall and Agile models. The diversity in metrics between these models is mostly because of the different processes in these models. For e.g., while Waterfall emphasizes on documentation, Agile focuses on lightweight documents. [Javdani, et al., 2013]

Waterfall model metrics

1. **Thousand Lines per Code**

   A traditional metric that was commonly deployed in models such as Waterfall was Thousand Lines per Code (KLOC). This looked at the size of the code as a metric for work done. In more recent times the number of commits to a repository is another example of a similar metric for measuring work done. However, the obvious problem that arises here is that these metrics can be easily manipulated by the developer without any meaningful additions to the project. In this sense lines of code etc. need to be individually evaluated for quality if they are to be meaningful metrics today.

2. **Number of pages of documentation**

   Good documentation is key to the maintainability of the code but it also provides insights into precision of the developer. To capture another traditional metric that can be deployed is the number of pages of documentation per KLOC.

3. **Defect density**

   The quality of a software project is often determined not only by how it meets the requirements of the client on face value but also the number of bugs present. One of the ways that quality can be measured is by the defect density i.e. the number of bugs divided by the total number of lines of code.

4. **Function point (effort analysis)**

   A prominent metric for measuring effort due to its sophistication and independence of human factors is function point analysis. Function points measure the functionality that a unit of code provides a software project as a whole.

Agile model metrics

Agile development methods were introduced as a reaction to traditional software development methods. Agile as we have seen focuses on characteristics previously not valued as highly by traditional methods and hence a number of metrics we'll see have been newly developed specifically to support Agile development measurement. In fact, up until 2011, when COSMIC introduced a guideline, there was no official publication regarding measurement in Agile methods [Javdani, et al., 2013].

1. **Effort estimation**

   Product owners capture requirements from the client, but they don't always understand the details of implementation. So good estimation can give the product manager new insight into the level of effort for each work item, which then feeds back into their assessment of each item's relative priority. User requirements in agile methods are defined as user stories and are collected in backlog. The most popular and common approach for effort estimation in an Agile environment is a subjective estimation. Teams often collectively work out a story point estimation for each task. Story points rate the relative effort of work in a Fibonacci-like format: 0, 0.5, 1, 2, 3, 5, 8, 13, 20, 40, 100. The abstraction here pushes the team to make tougher decisions around the difficulty of work and hopefully make it more accurate [Radigan, 2018].

2. **Velocity**

   Velocity is the average amount of work a scrum team completes during a sprint, measured in either story points or hours, and is very useful for forecasting. The product owner can use velocity to predict how quickly a team can work through the backlog if the metric tracks the forecasted and completed work over several iterations; the more iterations, the more accurate the forecast [Radigan, 2018].

3. **Sprint burndown and epic burndown charts**

   Delivering a software product on time is often essential in order to satisfy the client. One metric we can use to keep track of the progress of a project is a burndown chart of which there are two types. Scrum teams organize development into time-boxed sprints which may contain work from several epics. At the outset of the sprint, the team forecasts how much work they can complete during a sprint. A sprint burndown report then tracks the completion of work throughout the sprint. The x-axis represents time, and the y-axis refers to the amount of work left to complete, measured in either story points or hours. The goal is to have all the forecasted work completed by the end of the sprint. It's important however to track both the progress of individual sprints as well as epics so we can also take a look at the epic and release burndown charts to track the progress of team over a larger body of work [Radigan, 2018].

4. **Responding to change**

   Adaptability is one of the key agile principles and so a useful metric many teams use is 'Re-work' which can be seen an indicator of the ability of a team to hand over quality product i.e. free from bugs etc. Re-work can be defined as the man-hours spent fixing flaws and defects. This can be compared to the number of man-hours spent on developing new tasks. As this is done within each iteration it is also a useful metric for discovering bottlenecks and delays within a project [Javdani, et al., 2013].

Computational programs

1. **Hackystat**

   Many of the metrics we explored above form part of The Personal Software Process (PSP); a popular process for software development that was designed to help engineers reduce defects in their software and increase the accuracy of their estimations of the development duration. The PSP process consists of a set of methods, forms, and scripts that show software engineers how to plan, measure, and manage their work. Although the concept of the PSP is widely praised, there are two major issues with the process; one is the pressure on developers to manipulate numbers to satisfy management, an issue we touched on within individual metrics such as KLOC and the other is the big manual overhead in collecting much of the data required to complete the analysis using the PSP.

   As a response to these issues, computational tools such as Hackystat, Jasmine and PROM were developed to support automated data collection, data tracking and data analysis. In particular we're going to take a look at Hackystat, an open source framework developed by the Collaborative Software Development Laboratory (CSDL) at the University of Hawaii at Manoa. Hackystat allows users to attach software 'sensors' to their development tools, which unobtrusively collect and send "raw" data about development to a web service called the Hackystat SensorBase which can be later queried by other web services to form higher level abstractions of this raw data [Hackystat, 2018]. Aside from removing the manual overhead of PSP, one of the main benefits of Hackystat is that it collects data on both the client side and the server side. This gives great insights into

individual engineers as well as how the team co-operates. Another benefit was the level of detailed analysis provided by Hackystat because it collects data consistently throughout the day on whether the developer is calling methods, testing as they go etc. Hackystat or the likes is now integrated into many IDE's because of it's popularity, however, this collection of data has raised ethical concerns particularly among developers who are uncomfortable with the level of personal data being collected. We'll explore this in the next section.

2. **Jira Software**

As we've seen the Agile model is often so different than traditional models, the metrics and tools used to support it are usually bespoke; when it comes to computational platforms this is no different. The most widely used platform which supports the Agile metrics we've looked at is Jira Software, an Agile project management tool. It provides everything from Agile boards to reports, in which teams can plan, track and manage all their Agile based software development projects with a single tool [Atlassian, 2018].

Jira allows teams to estimate stories, adjust sprint scope, check velocity, and re-prioritize issues in real-time. This is extremely useful for teams in sprint planning meetings to help more accurately plan what needs to be done from the backlog to progress the project. During the sprint then, custom issue types can be developed for bugs, stories etc and a custom workflow configured to update issues automatically based on events in other systems, or define criteria that must be met before an issue can progress. Additionally, a feature called release hub allows developers to see what's shipping in the next release. All of these features help measure and keep track of all the work being tackled in each sprint and allow developers to stay focused on the task at hand [Atlassian, 2018].

Finally, then Jira provides teams with retrospectives through various metric reports such as Burndown charts, Velocity charts etc. which we explored in the metrics section of this report [Atlassian, 2018].

Of course, Jira as with Hackystat faces issues in regard to manipulating the system for better reports and concerns from developers over the level of data

being tracked. Jira, however, does tackle some of these issues and comes with a feature called sprint permissions which allows a team to define what each developer can see or do in a given project.

Ethical concerns and conclusion

Whether it's measuring your own personal performance, that of your teams or the quality of the product you produce; the measurement of the software engineering process is key.

However, especially today, we often look at metrics as the be-all and end-all to the success of a project without considering the possible negative effect these metrics can have. As I alluded to at various points throughout this report, many metrics come with possible ethical concerns around the privacy of developers. In fact, when metrics from the software engineering process are used to; assess the individual developer or intimidate the developer with observations this can not only cause developers to manipulate the numbers or refuse to participate but also reduce morale and in turn productivity. A study by the Hitachi found that "Compared to people who are unhappy, it has been found that people who are happy have 37% higher work productivity and 300% higher creativity." [Yano, et al., 2015]. Some companies are even starting to use the latest game theory techniques to motivate and increase the productivity of developers [Silverman, 2011]. So as Brooks put it all those years ago, "there is no silver bullet" when it comes to measuring software engineering; not only is the process itself complicated to measure but indeed the people involved in it need to be considered to. Companies and managers must try strike the balance when it comes to metrics, their accuracy, and employee happiness as they are all ultimately interlocked in the overall result of the project.

Bibliography

Radigan (2018) “Five agile metrics you won't hate,” *Atlassian*. Available at: https://www.atlassian.com/agile/project-management/metrics.

Atlassian (2018) “Jira | Project Tracking Software,” *Atlassian*. Available at: https://www.atlassian.com/software/jira.

Hackystat (2018) *Hackystat*. Available at: https://hackystat.github.io/.

“No Silver Bullet: Essence and Accidents of Software Engineering” (1987) *No Silver Bullet:* Available at: http://www.cs.nott.ac.uk/~pszcah/G51ISS/Documents/NoSilverBullet.html.

Silverman, R. E. (2011) “Latest Game Theory: Mixing Work and Play,” *The Wall Street Journal*, 10 October. Available at: https://www.wsj.com/articles/SB10001424052970204294504576615371783795248.

Tutorialspoint (2018) “SDLC Agile Model,” *www.tutorialspoint.com*. Available at: https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm.

Tutorialspoint (2018) “SDLC Waterfall Model,” *www.tutorialspoint.com*. Available at: https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm.

SoftwareTestingGenius (2018) “An Introduction to Metrics used during Software Development Life Cyclel,” *www.softwaretestinggenius.com/*. Available at: https://www.softwaretestinggenius.com/download/aitmuds.pdf.

Taghi Javdani , Hazura Zulzalil, Abd. Azim Abd. Ghani, Abubakar Md. Sultan, On the current measurement practices in agile software development, International Journal of Computer Science Issues, 2012, Vol. 9, Issue 4, No. 3, pp. 127-133.