Software Engineering

Measuring Software Engineering Productivity

Luke Hackett - 17324340

Introduction

In this report I will be outlining the various ways in which software engineering can be measured using quantifiable data and assessed. I will be going through the various platforms that offer services to perform these measurements and assessments as well as the approaches available and the ethics concerns with this type of work. There are many reasons that someone would want to measure productivity. Employers want to know if their employees are performing as well as they could, and employees could be interested in seeing how they are performing so that they can more accurately predict velocity and how much they can say they will get done to make sure employer and customer expectations are in line.

Measuring this productivity is difficult to do as there is no definitive metric for how much development you have completed. As well as this there are other aspects of software development such as refactoring, adding active/passive monitoring and unit testing that don't necessarily add features to the product but will have massive long-term gains for the project. Therefore, I would say that all these measurements are only a guide for what you can expect and not by any means an exact science.

Approaches for Measuring Software Development

A very straightforward approach to measuring software development productivity is looking at the numbers produced by each developer by a version control system such as git. These are very easy to misinterpret however and can lead to incorrect extrapolations.

The first of these types of metrics is the number of commits that a developer makes. This seems at first glance like it would be a good measure of how productive a developer is. However, this does not consider the amount of code changed in a given commit or what proportion of a given task is completed in this commit. This is an easily exploitable metric that by itself would reward clutter and encourage bad git practice by increasing drastically the number of commits.

Instead of this, say you measured lines of code(loc) as a measure of productivity. This could be more accurate however this could potentially reward inefficient code and take credit away from very clever and short solutions that could have taken a significant amount of time to think of. Another reason this is highly unreliable is as it depends highly on the language used to write code. A language like Haskell can implement certain methods in a few lines that could take a C like language 30 lines of code. Quicksort is a good example of this, you can see below quicksort in Haskell vs quicksort in C and why lines of code is ineffective as they are as much productive work as each other:

```
quicksort [] = []
quicksort (p:xs) = (quicksort lesser) ++ [p] ++ (quicksort greater)
    where
        lesser = filter (< p) xs
        greater = filter (>= p) xs
```

```
// To sort array a[] of size n: qsort(a,0,n-1)

void qsort(int a[], int lo, int hi)
{
  int h, l, p, t;

  if (lo < hi) {
    l = lo;
    h = hi;
    p = a[hi];

    do {
      while ((l < h) && (a[l] <= p))
          l = l+1;
      while ((h > l) && (a[h] >= p))
          h = h-1;
      if (l < h) {
          t = a[l];
          a[l] = a[h];
          a[h] = t;
      }
    } while (l < h);

    a[hi] = a[l];
    a[l] = p;

    qsort( a, lo, l-1 );
    qsort( a, l+1, hi );
  }
}
```

This type of measurement would also possibly discourage developers from building highly modular and reusable code as this would take away from their overall ‘productivity’. In my opinion both metrics outlined above are highly ineffective and need to be rethought.

Another approach to take is abstract away any one metric and ask the developer to estimate the time for a given tasks in an arbitrary scale and then see how close they get when development is completed in relation to a strict definition of done. A task can be defined as done when all the code for the required feature is completed, as well as comprehensive tests for any testable areas of code, and finally some monitoring and any documentation for the feature if required. This work clearly goes beyond what can be measured by number of

commits/lines of code and can encompass all tasks surrounding development that can increase the time required to complete a task.

As well as these tasks bugs in code are something that can pop up and take an indefinite amount of time to fix as the scope is often unknown and a lot of investigation work is needed. For this reason, bugs are not suitable for estimation. Tech dept is also something that may not be estimated but must be considered when measuring productivity. Tech debt can be composed of many things, such as untested bits of code or undocumented code. This can not be expressly estimated and can take up significant chunks of time.

The approach outlined above is essentially measuring speed which is, in my opinion, one of the best indicators of productivity. Trusting the developers to estimate a given task as a certain amount of time then measuring against their own expectations can help them understand their own productivity better and is more impervious then other methods against 'gaming' the system as it is not good to be constantly under estimating the amount of work you can get done. It is advantageous for both developers and employers to have accurate predictions on tasks. It will benefit the entire team/product.

Available Platforms

There have been several platforms that have emerged as people have tried to measure productivity. They can be totally manual or automatically scrape metrics from VCS depending on what you require. The tools that people/companies use heavily depends on the metrics they want to check and the amount of time and money they are willing to invest. Some of the top available platforms are:

- GitPrime
- Hackystat
- Jira

Git Prime

GitPrime collates all the data that can be taken from git into easy to pallet graphs and metrics for managers and engineers to optimize their coding practices. Gitprime is a very easy thing to use for a lot of companies as almost everyone uses one version of git or another as their VCS. The tool itself collects a lot of the data outlined in the previous section that individually may be misleading however when processed and combined it can be an interesting measure of productivity.

Furthermore, it will try extract meaning from the data which is interesting to see and saves time. It can tell you a lot about how a developer is performing on commits on a project, the tickets these are linked to and Pull requests that are currently out for a developer. These work logs are convenient for managers for a quick view of developer productivity. They also boast that it can correlate events to increases or decreases in productivity. For instance, they can say if a new Continuous deployment/Continuous integration tool is added to the pipeline then productivity increased for all developers. This is very useful as it is understanding that productivity is not just lines of code but all extraneous technical work surrounding getting a product to market.

An important part of measuring productivity is trying to increase productivity in the future. Gitprime provides retrospective tools so you can see when you were least productive and try to understand why. Overall, I believe that this tool is very useful for measuring productivity and takes a lot of the manual work out of the measurement process that you would have to do in some of the other tools.

Hackystat

Hackystat is a tool that collects swathes of data on developers during their development process and analyses it. It provides visualisations as well as interpretations of the data with annotations included. They claim to ‘unobtrusively’ collect data by attaching sensors to the developer’s tools (such as their ide), which in turn sends that raw data to a server to interpret.

Hackystats data collection is very fine grained which separates it from other platforms. It collects data down to the complexity of methods. This highly granular data allows whoever may be measuring to see on a second to second basis how productive a software developer is being.

While this is clearly a very effective way of measuring every task the developer does there are clearly ethical concerns and developers will most likely object to being micro managed in such a blatant way.

Jira

Jira is one of the platforms that is mostly manual. It collects data and has it all contained in one place however it does not make deep extrapolations from said data. Jira is a Kanban system built to work with Agile workflow. You can have a well-defined board of tasks that are all estimated and planned out. Jira can track these tasks through their various stages, from start to in progress to in test to done. This follows the stages of software development and when paired with the estimated velocity before starting the task can be a powerful tool for measuring productivity.

While this system is not as detailed as a system such as Hackystat it provides solid rough data to evaluate productivity and is significantly less intrusive than Hackystat while performing all necessary functions. It can graph expected velocity and how it differs from estimated, as well as the total amount of stories left on the project as well as a few other interesting metrics. I personally would be substantially happier to work with a measurement tool such as this rather than Hackystat.

All these platforms for measuring efficiency have their own advantages and disadvantages. Some companies may prefer some of the more manual systems or the intense automation depending on their needs. It is highly likely that developers would be against some of the more intrusive tools such as Hackystat, even Gitprime seems to reduce a lot of what we do to numbers and some developers may be against the oversimplification.

Algorithmic Approaches

There have been several differed algorithms for software development dreamed up in the last few years by people trying to accurately measure productivity. Some of these new algorithms are ingenious and take advantage of the latest technological trends such as Machine Learning. I will outline some of the more interesting approaches below.

Analogy and Analogy-X

Analogous estimation is an estimation technique that compares a previous smaller activity to current activity and draws proportional comparisons. This method can be used to measure complexity, size and weight of tasks. This method is most effective when the estimation is done by an expert, or even more specifically a member of the team that has performed these previous tasks. In a lot of companies analogous estimation is the backbone of what developers are doing when they estimate a given task in story points. They are evaluating how long a given piece of work will take based on past experiences

Analogy-X is an algorithm for software cost estimation that estimates software development. Analogy based estimation assumes that projects that are similar with respect to features are similar in effort, and Analogy-X provides a way of challenging these assumptions. It can do this by using Mantel's correlation and randomization tests for comparing two distance matrices. This formalization and statistical basis make analogy estimation more robust.

Agile and Kanban

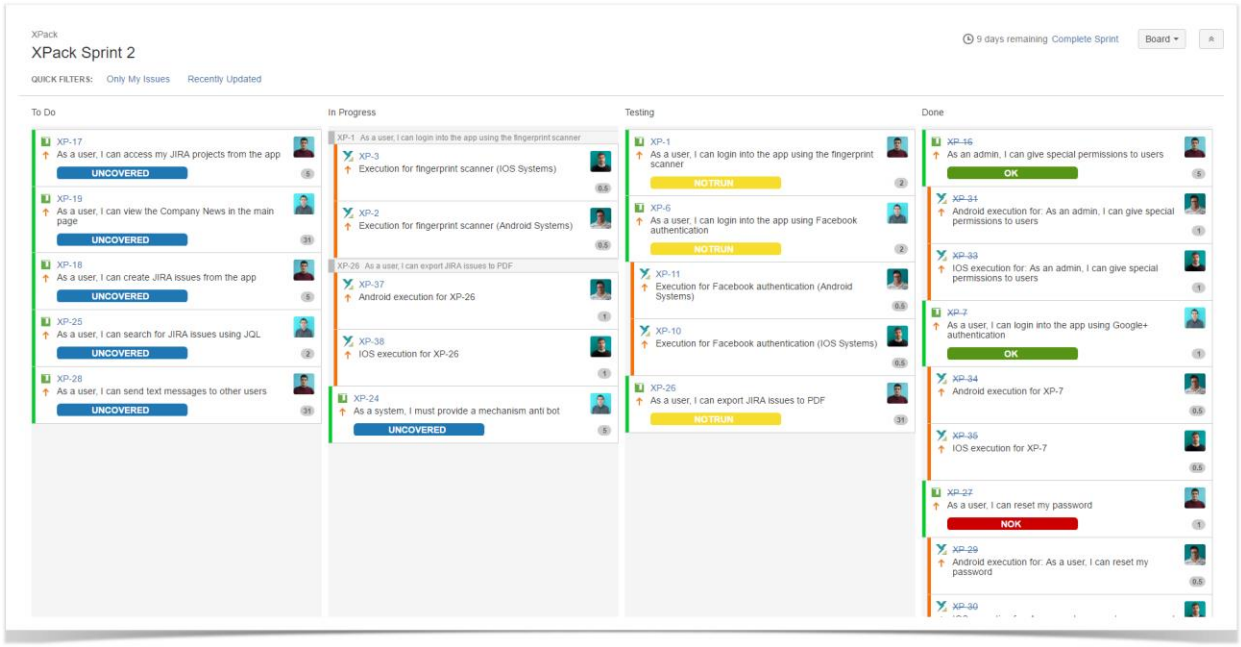
Agile is an approach to software development that understands that the process is fluid and evolves over time due to customer needs, revision of goals and feedback from managers. Work is broken up into short periods, where there are two week "sprints" where the goal is for a team to complete a given feature or deploy a certain product. During this two-week sprint developers work on stories that they have estimated previously. The idea is that at the end of the sprint all work for an overarching task will be done and the developers will be free to work on a completely new thing.

During the sprint the team will meet daily to talk about their progress and make any changes necessary to allow them to complete their work. Maybe a developer would have a

blocking bug from someone on another team or work that would have to be done before they could begin, and the PM would have to go talk to other teams and let them know. This is a massively powerful part of the agile workflow and increases productivity tenfold.

At the end of a sprint there is a retrospective where the team meets and will talk about the positives and negatives of what had been done during the sprint. This helps people more accurately estimate their work in the future and identify any bottlenecks in their current process.

As an extension to this workflow a lot of companies use a Kanban system to track how work is being done easily and visually. There can be a table with different columns for each stage of the development process:



Ethics Concerns

Whenever dealing with large swathes of data that has been collected from people there are always ethical concerns to do with what can be collected and what can ethically be used to calculate productivity. A lot of ideas go too far in my opinion and most developers would agree. Certain companies have started measuring employees' posture or taking screenshots of

employees screens every few minutes to ensure they are working. This type of micro-management in a high skill field is not ethical in my opinion as it goes well beyond what you would be expected and need to know.

The Hawthorne effect (also referred to as the observer effect) is a type of reactivity in which individuals modify an aspect of their behaviour in response to their awareness of being observed. So, this also means that the data collected by these extremely intrusive methods may not even be as effective as the employees would not be as productive when they are being watched.

Conclusion

From all that has been discussed above it is clear that there is no holy grail of productivity measurement. I believe that the best way to measure and estimate productivity is to leave it to the people who know what they are talking, the developers. What computational platforms can be used should be what they feel most comfortable to work with. A platform like Jira or Azure Devops that has a Kanban board to help track work and aid with retrospectives gives the developers all the power that they need to get their work done while not intruding. Gitprime definitely seems useful for managers when considering things like promotions during employee reviews.

References

[1] S.Malathi, & Seshadri, Dr.Sridhar. (2011). An algorithmic approach for the implementation of analogy–X for software cost estimation. 10.13140/RG.2.1.3899.4004.

[2] Hélie, Jean, Ian Wright, and Albert Ziegler. "Measuring software development productivity: A machine learning approach." Proceedings of the Conference on Machine Learning for Programming Workshop, affiliated with FLoC. Vol. 18. 2018.

[3] https://www.gitprime.com/platform/code/

[4] J. W. Keung, B. A. Kitchenham and D. R. Jeffery, "Analogy-X: Providing Statistical Inference to Analogy-Based Software Cost Estimation," in IEEE Transactions on Software Engineering, vol. 34, no. 4, pp. 471-484, July-Aug. 2008.

[5] https://code.google.com/archive/p/hackystat/

[6] https://dev.to/nickhodges/can-developer-productivity-be-measured-1npo

[7] https://hackernoon.com/measure-a-developers-impact-e2e18593ac79