MEASURING SOFTWARE ENGINEERING

Measuring, Evaluating, Calculating & Considering Software Engineering Performance

James Cowan

SF Computer Science & Business | Software Engineering CS33012 |
Student No. 19309917

Introduction

Software engineering is a new, broad, and dynamic field. Unlike physics, biology, or chemistry, which can trace their origins to legends of history like Pythagoras or Hippocrates, software engineering as a field of practice is arguably around half a century old. As a result, the styles and applications of the field are rather loose with a diversity of approaches and theories. Some even question whether software engineering is a form of engineering at all, with notable figures in computer science like Edsger Dijkstra describing the field as “How to program if you cannot” [1]. Despite this, in the practical world of software, software engineering has been embraced as a field of practice taken seriously by governments, international organizations, and the largest and most valuable companies in the world.

Many approaches and methodologies to organize and further the practice of software engineering have been developed to handle the ever-increasing scale and complexity of modern software projects. According to Reisenwitz, Agile, Waterfall, Rapid, and DevOps have all been introduced as practices to best handle the development, maintenance, and deployment of projects from small-scale applications to enterprise products [2]. Around these methodologies, many tools have been developed to implement and improve these practices. Notably the Atlassian suite and GitHub Enterprise. Many of these tools have been used to improve the measurement and organization of software projects.

All of these tools, practices, and methodologies seek to accomplish a few simple goals. That being the creation of a measurable and systematic approach to software development,

applying the precepts of engineering to software development. Maintenance, business, timelines, and other concerns factor into the day-to-day practical considerations of the software engineering world.

Measuring Engineering Activity

Of the various methodologies involved in the process of software development, all require some form of quantification to determine when a stage of the process is complete. Waterfall is only fully complete on the release of the project, with maintenance following suit. Rapid application development works in cycles of defining, prototyping, and demonstrating, with completion at client approval and deployment. Agile cycles are considered complete at the end of their sprint finishing one feature or aspect to the project, leading to the next one. DevOps takes this another step by focusing on continuous integration and deployment, releasing small implementations often, and automating both builds and releases as much as possible. When it comes down to the core of each of these strategies it is clear that the main issue that they focus on is quantification of scope and progress [2].

When is a feature truly done? Is it when you have a working implementation that passes tests? Or is it only when the project is released in its entirety? And how can you know if someone is contributing their fair share? It's obviously difficult to compare the contributions of a seasoned developer to the new hire. Settling on simple solutions like commits or passing automated test suites can lead to a moral hazard where developers optimize their work to just look good, rather than getting the job done. Thus, we must consider more holistic methods to measure

engineering progress, and the extent to which the various methodologies appropriately quantify progress. Additionally, the class of project at issue will consequently become a major consideration to the software engineering process.

Some ways that we can think about the types of projects that affect how we approach the measurement of software engineering start with timelines. A triple-A game will have very different measurement considerations than, for example, a simple CRM platform. Where your triple-A developer is looking towards the release date, and thus the final product at that date, the CRM developer is more concerned about the singular feature and will accept a slower speed of development if it creates a better end product. Naturally, different development methodologies better fit each of these strategies. Perhaps Waterfall for the triple-A game studio and Agile or DevOps for the CRM development team. Thus, a one-size-fits-all approach is unwise for the issue of progress measurement.

Segmenting performance quantification and finding appropriate scale for tasks in a project can be important factors for effectively measuring software engineering performance. Tasks with too wide of a scope lack direction. This leads to overlapping roles and stymied progress. Too narrow of a scope can create a false sense of achievement. Which wastes time by bogging the team down in design considerations, without making considerable progress on the project as a whole. Finding appropriate scope requires project directors to set defined and achievable goals that fall within the scope.

For example, a team may be building a calendar as part of a larger system. When determining the scope, may want to set the calendar, with all of the integrations and extra features as a single Agile iteration. Another director may set the focus too small, suggesting that the first iteration focus on the event modal, then the next on the dropdown, then the next on a settings pane, and so on and so on. In that way wasting time on small features that can be enveloped in an achievable yet focused goal. This team would be better off by focusing on getting a working calendar first, then focusing on the extra features at the next iteration or set of iterations. This would let them make progress where they can and avoid getting stymied or lost in too broad of a scope. This approach is a functional and more easily measurable form of segmentation appropriate for development teams rather than a single developer.

When it comes to measuring the progress of a single developer, focusing on the human issue is paramount. Many statisticians will tell you that a univariate analysis is always missing critical information. Likewise, focusing on singular or simple factors like the number of code commits, lines written, or total characters written will be absent of any real meaning and can be effectively exploited by the people they are supposed to evaluate. People naturally want to look productive and accomplished, and of course, will leverage simple factors that contribute to this. If they are evaluated on, for example, the number of commits they push, then they will increase volume, without improving output or quality, as this is in their best interest [3]. Thus, simple approaches like this are best avoided for critically evaluating output and performance which absent other factors are made largely meaningless. Beyond this, there are many other reasons why simple analysis may fall short. A developer may be exceptional at cleaning up an

optimizing code, but below par in its initial creation. These human factors emphasize the need for holistic and flexible personal evaluation approaches to software engineering.

Where a business manager may seek to build a formulaic approach tied to revenue and sales, seasoned developers know that there are more considerations beyond simply the superficial monetary value of what they are currently working on. Other firms may be focused on the number of lines of code a single developer produces, but again, this is a flawed conception. A significantly better developer would produce a much more concise and maintainable codebase [1].

With this in mind, determining a quantifiable measure of software engineering output and progress is difficult, but has the opportunity to be much more representative of the true nature of robust software development. Thus, using a combination of team and personal factors and moving beyond surface-level measures of progress is the most constructive approach to better manage developers, their teams, and their projects.

Gathering Code Data

In recent years, many new tools and platforms have emerged to handle the complexity involved in software engineering management. Multi-billion-dollar firms like Atlassian have become leaders in this space with many established firms like GitHub, JetBrains, and Salesforce rolling out software engineering-focused tools to follow suit. These suites of software form the backbone for modern software evaluation methodology. Many different classes of tools have

emerged to address the forms of evaluation prescient to these issues, providing immense value when teams are managed and delegated effectively. These tool suites can help gain better insight into programming output and developer productivity.

One of the most common methods of evaluating programming output comes in the form of code review platforms. Some notable examples of this include JetBrains' Upsource, which integrates directly with JetBrains' suite of enterprise-grade IDEs [4], and Atlassian Crucible, which works with Atlassian's Bitbucket service for Git as well as their issue tracker service Jira [5]. These integrated tools help team managers and senior developers improve output by assuring more consistent quality and reducing maintenance requirements into the future. Code review tools like those mentioned can provide project leaders with a more qualitative assessment of the output that their team produces. This may affect their evaluation in ways for which quantitative analyses cannot account. Metrics obtained through these code review tools like percentage of code approved, time to approval, percentage modified, and other data can contribute to a quantitative analysis that may prove useful in evaluating a developer's output.

Once projects or iterations reach the alpha, beta, or maintenance stages of development, issue trackers are generally employed in order to improve quality assurance. Additionally, issue trackers can be useful for evaluating both team and individual performance. Issue tracking services can be linked to code review tools and progress measurement systems. Of this class of tools, Atlassian's Jira is the most popular, feature-rich, and noteworthy. Jira, like many of its ilk, allows clients, end-users, and fellow developers to create issue tickets, adding a bug fix to the

project teams' agenda [5]. Generally, issue resolution can be linked to segments of code, which may provide insight into which developers may have caused the issue, who approved it, and how long it existed before resolution. These can be exported for evaluation purposes.

Dynamic insight tools for software engineering have also become popular in recent years, with the most recent headliner being GitHub Copilot, which while not providing raw data to teams or individuals, uses machine learning and other artificial intelligence strategies to suggest implementations and improve code completion [6]. Other services like the popular integration for GitHub and Bitbucket, Snyk, can identify vulnerabilities, gather data, and attempt to resolve security issues through artificial intelligence [7]. As these intelligent tools improve, teams and individual developers can gain better insights and operate on data obtained.

Many tools beyond those mentioned are employed in order to improve quality and gather data and insights for enterprise projects, often in the form of adopting overall suites of enterprise software like those offered by Atlassian or GitLab. By leveraging the breadth of these tools and the insights they provide, and with the proper project and scope management, software development tools, issue trackers, and intelligent systems provide a solid basis for improving and evaluating software engineering performance.

Evaluating and Profiling Software Engineering Performance

When employing the tools common to software engineering, project managers and developers try to gain insight into their group and individual performance. Many strategies, both

quantitative and qualitative are adopted to effectively evaluate performance. The breadth of available tools and methodologies to evaluate performance have improved project members' ability to constructively improve performance and code quality. Some forms of evaluation, like Gantt charts and Kanban boards, provide a more easily communicated visual representation of performance, while other tools provide a more numerical demonstration of performance.

When it comes to progress tracking, there are a plethora of applications and platform integrations of note. In this case, Atlassian, and their Trello platform again take the lead in this measure, especially in the context of the integrations available through Atlassian's broader suite of services [5]. Other similar services generally called "Kanban boards" are a common method of tracking progress. This is especially common as a form of scrum management in agile methodologies. Many teams compare their current progress to timelines established in Gantt charts, which are timeline-oriented bar charts that indicate task length, dependencies, and milestones, a leading example of which is developed by Monday.com [8]. These platforms can generally be automated in response to pull requests, code reviews, issue resolution, and the like in order to provide a visual representation of the project's current progress.

Often organizations develop an internal "scorecard" to quantitatively assess their developer's performance, with author Steve McConnell in a video presentation recommending this approach, but caveats this by mentioning that it is not verifiable or a hard measurement. Instead, McConnell emphasizes that the scorecard's structure and alignment with organizational goals. McConnell notes that with this approach you can use the scorecard as a

tool to focus on output. McConnell, however, does not recommend this approach when evaluating process, though improvements in process can be gleaned from a post-change evaluation using this method [3]. This style of scorecard evaluation is concurred by Reisenwitz, who emphasizes considerations of the actual metrics used, rather than employing all possible metrics available [2].

Despite much of the seasoned advice from experts like McConnell, many firms are quick to subscribe to the notion that more data is better data, and the mistaken conception that machine learning simply improves these analyses. This embrace of buzzwords, however, is shortsighted and flawed. While improved data gathering and evaluation can be useful to discover meaningful metrics, many firms, unfortunately, consider too many of these factors, or weight them incorrectly, drawing meaningless conclusions from otherwise meaningful data. Most often, factors that should be considered for individual performance evaluation can be project-specific, role-specific, and team-specific, and metrics for evaluating a security maintainer are almost certainly incongruent with metrics for evaluating an app developer.

Considerations of the Use of Personal Data

Like with many of the other topics evaluated, the human factor of personal data use reaches the forefront. Companies, governments, and services have become ever wearier of personal data collection and processing, with sweeping legislation like the European Union's General Data Protection Regulation, the California Data Protection Act, and many other pieces of legislation setting out guidelines for personal data processing and use. In many organizations,

quantitative methods of evaluation can be the difference between a raise for a star developer, and a person's unemployment. Keeping this in mind, we must be ever critical and reduce our reliance on quantitative evaluation of a complicated and often more nuanced area of practice like software engineering. Instead, to evaluate the process, performance, and output of a software engineer more constructively it would be wiser and more ethical to strike a balance between both individual and team, and quantitative and qualitative factors.

Conclusion

As project managers, executives, peers, and individual developers seek to critically evaluate their processes, performance, and output, there are a variety of methodologies, tools, and data sources that prove useful to this pursuit. As the world of software engineering moves to more collaborative platforms with greater integrations, data generation, and analysis many opportunities to improve quality and output have emerged. Despite these helpful tools, when evaluating performance, human factors are of course key. Too often, as described by Reisenwitz, software engineering is evaluated on flawed metrics that are unconnected with revenue, organizational goals, and code quality [2]. It is clear that pure evaluation of software engineering tends to be subjective, unverifiable, and ignorant of human and organizational factors. Rather than being seen as a process of gradual improvement, learning, and collaboration, common methods of software engineering evaluation are at risk to be seen as an absolute verdict on performance. Perhaps rather than viewing software engineering as a "pure" engineering discipline which can be conclusively evaluated, we should call back to Edsger

Dijkstra's adage that "If we wish to count lines of code, we should not regard them as 'lines produced' but as 'lines spent'" [1].

Bibliography

[1] E. Dijkstra, "On the cruelty of really teaching computing science," The University of Texas at Austin, Austin, 1988.

[2] C. Reisenwitz, "The ecstasy and agony of measuring productivity for Software Engineering & Development," Clockwise, 3 June 2021. [Online]. Available: https://www.getclockwise.com/blog/measure-productivity-development.

[3] S. McConnell and C. Software, "Measuring Software Development Productivity | Steve McConnell," 26 January 2016. [Online]. Available: https://www.youtube.com/watch?v=x4IboMnTdSA.

[4] JetBrains, "Upsource: Code Review and Project Analytics by JetBrains," 2021. [Online]. Available: https://www.jetbrains.com/upsource/.

[5] Atlassian, "Atlassian | Software Development and Collaboration Tools," 2021. [Online]. Available: https://www.atlassian.com/.

[6] GitHub, "GitHub Copilot - Your AI pair programmer," 2021. [Online]. Available: https://copilot.github.com/.

[7] GitLab, "Iterate faster, innovate together - GitLab," 2021. [Online]. Available: https://about.gitlab.com/.

[8] Snyk, "Snyk | Developer security | Develop fast. Stay secure.," 2021. [Online]. Available: https://snyk.io/.

[9] Monday.com, "Project Management Software | monday.com," 2021. [Online]. Available: https://monday.com/project-management/features.