Engineering Management
Field Project

Evaluating the Productivity of Software Engineers in Enterprise Development

By

Stephen J. Pack

Fall Semester, 2010

An EMGT Field Project report submitted to the Engineering Management Program and the Faculty of the Graduate School of The University of Kansas in partial fulfillment of the requirements for the degree of Master's of Science

_____
Tom Bowlin
Committee Chairperson

_____
Mike Kelly
Committee Member

_____
John Bricklemyer
Committee Member

Date accepted:_____

Acknowledgements

I would like to thank my many mentors, coworkers, and managers at Cerner, where I have been given the opportunity to grow and take on interesting challenges such as the question that fueled this research. In particular, thanks to Drew Clippard and David Edwards, who have helped shape my understanding of development practices and pushed improvement through example.

The faculty of the Engineering Management program has been central to formalizing the knowledge derived from my work experience, and I am grateful to my committee for their time on this project.

Finally, thanks to my wife and parents for their patience and encouragement throughout the degree program.

Executive Summary

Managing and evaluating the work of software engineers creating complex products at large corporations is particularly challenging with no standardized system to recognize productivity. Cerner Corporation, a leading supplier of healthcare information technology solutions, gives managers substantial latitude in tracking productivity, yielding high variance. The research reported here involves an examination of relevant background literature and interviews of Cerner associates with multiple roles in the organization as well as the author's own background.

By identifying the essential components of good software engineering and potential measurement systems, the research yields a design that the author will use to track the productivity of his direct report engineers in the next annual performance period. In it, the primary metric is the completion of story points, an Agile software development representation of the relative size and complexity of work to be done. Tracking the introduction of defects is an indicator of an engineer's code quality, although sufficient context must be captured. Finally, a peer feedback system helps ensure the manager recognizes performance from other perspectives.

Table of Contents

Introduction

Front-line managers of software engineers at large corporations are often responsible for writing annual reviews or otherwise evaluating the performance of their direct reports. Beyond providing feedback on skill and career development, evaluations often include discrete choices in allocation of compensation and promotions, effectively forcing a rank-ordering. Therefore, it is important to have a system that is clearly communicated and justly executed both within a team and across the organization, so managers have confidence they are making the right decisions and engineers recognize the fairness of the process.

A process in which a manager submits an evaluation entirely barren of recognized inputs other than the manager's thoughts could be seen by engineers as meaningless and arbitrary; with no controlling factors present, the output is potentially highly subjective, whether intentional or not. Additionally, such evaluations could be unduly influenced by factors such as time skew, with the successes and failures at the beginning and end of an evaluation period likely to have more impact in memory than those in the middle.

In comparison to other fields of engineering, software is notorious for late delivery and poor quality, so a desire to implement more scientific management to control teams and individuals is understandable, including correctly identifying both outstanding and poor performers. However, the system complexity, interdependencies, and long time-to-market of large enterprise development

preclude tracking back the outcomes of a software product to a single engineer. Additionally, the wide variance between types of projects different teams work on yield incomparable work packages. Daily coding practices would be expected to vary highly due to different technologies, architectural targets, team cultures, and experience levels. It is therefore necessary to develop an evaluation system around inputs and outputs that are more controlled and recognizable to a close observer.

Managers typically have a technical education and background, themselves having risen from an engineering position. Given this engineering history, it is perhaps not surprising that there have been systems devised to put numeric values on the output of engineers' work, based on the notion that irrefutable measurements lead to the most objective system possible. However, measurements are no panacea, as they can be misleading, manipulated, encourage undesired or unforeseen outcomes, and undermine morale depending on application.

Software engineers have a strong self-image of the highly-skilled professionals they are and an inherent suspicion that many metrics have little correlation to achieved value. Measurements on an individual level gathered and acted on by observers distant from the development team are particularly suspect, as normalization is attempted across different environments with highly relevant context missing. Yet software engineers will readily agree that abilities are far from equal, with orders-of-magnitude separating the best and worst

performers. Furthermore, engineers are well-positioned to quickly recognize into which category a colleague falls, since it is possible to feign competence to a manager or especially an executive far longer than a peer with whom one works daily. This holds particularly when there is a high degree of cross-visibility on project work.

The evolution of software project management styles has significant interplay with evaluating individual and team performance. The "waterfall" approach in which each stage of a project -- for example, requirements, design, coding, and testing -- is completed before the next stage is started has been phased out at many companies in favor of more iterative approaches, in which the stages occur more simultaneously and are quickly cycled through. Business benefits of the latter are largely beyond the scope of this work, but include more rapid value delivery to customers and ability to respond easily to changing requirements. One consequence of this shift is that measurements that are only possible when substantial analysis has occurred early in the project lifecycle have no grounding in an iterative approach.

The purpose of this research project is to examine the existing scholarship in the evaluation of productivity of software engineers and research contemporary thoughts from practitioners in the field at many levels and with disparate points of view. Productivity evaluations at both the team and individual level are examined, since both are relevant, and there is often a management temptation to derive one if the other exists. For example, if individual metrics are

calculated, they tend to be rolled up to team scorecards, and conversely, team measurements may be broken down to individual attribution. At Cerner Corporation, the evaluation process is entrusted to the front-line managers. Despite some guidance on how to improve openness and objectiveness, it most often yields the aforementioned uncontrolled process with few recognized inputs. Through the analyses of previous works, existing processes, and original research, a system design will be developed, which the author intends to pilot test and refine with the team of engineers he manages at Cerner.

Literature Review

Attempts to measure the output of software engineers are as old as the profession. The first question examined is the broader one of whether to attempt measurement, or if it is inherently futile. Then, two of the most widely-known classes of software measurement are discussed, including their continued evolutions. Next, the full nature of a software engineer's work beyond creating code is explored. This has also led to the recent growth of Agile methodologies, a complicating factor in individual measurement. Finally, the question of how management should value output is offered.

Whether to Measure

Most literature related to the topic of measurement focuses on what to measure rather than whether to measure. However, the latter question is far from settled in the broader research, due to "the costs and potential for dysfunction associated with measurement in organizations" (Austin 1996, 4), with abundant examples of manipulative actions to get the numbers desired rather than achieve organizational goals. Particular caution is advised against metrics intended to motivate such as those that continue to rise over time, as workers take increasing shortcuts, and "measured performance trends upward; true performance declines sharply" (Austin 1996, 15).

Many advocate a middle path of balanced, contextual use, as "research indicates that indiscriminate use and undue confidence in [quantitative measures] result from insufficient knowledge of the full effects and consequences" (Ridgway 1956, 240). Measurement may be employed as a tool assisting internally-motivated employees to achieve organizational goals, but its presence can yield a threatening environment of external motivation. There is a tendency for managers to rely heavily on such measures when present, as "it is easier to defend ratings consistent with formal indicators of performance" rather than incorporating subjective corrections based on all available qualitative and quantitative information (Austin 1996, 71).

Classical Units of Measurement

Any form of measurement requires agreement on a standard unit to measure. Such a unit could serve in many calculations of developer productivity, including the number of units produced by an engineer in a given period and the number of defects found per unit. What unit this is -- and whether one exists at all -- is the center of much existing scholarship. "The difficulty with measuring productivity is that of measuring development output. Software development doesn't have a universal, perfect output measure, but some proxies do make sense in specific contexts and for specific purposes" (Erdogmus 2008, 4).

Norman Fenton indicated "that much published work in software metrics is theoretically flawed" (Fenton 1994, 199) as before any measurement is determined, "you need to know whether you want to measure for assessment or

for prediction" (Fenton 1994, 200). Yet in the effort to satisfy all needs with a single approach, the metrics discussed in the following paragraphs have been used by industry practitioners to both assess and predict the performance of both individuals and organizations. Existing scholarship has focused much more heavily on the latter, as organization-wide metrics are easier to quantify and analyze. Furthermore, "although external attributes like reliability of products, stability of processes, or productivity of resources tend to be the ones we are most interested in measuring, we cannot do so directly. We are generally forced to measure indirectly in terms of internal attributes" (Fenton 1994, 205), so much work is devoted to getting the internal to better correlate with the external.

The most primitive form of measuring a software engineer's output is counting the lines of code (LOC) written. This system began to emerge with computer programming itself in the 1950's (Jones 2008, 72). As Fenton points out, "even as simple a measure of length of programs as lines of code requires a well defined model of programs which enables us to identify unique lines unambiguously" (Fenton 1994, 199), though tooling could assist this model by imposing standard formatting and counting procedures.

However, modern high-level languages allow programmers to write far more complex logic in fewer lines of code, as well as writing that logic using a variety of algorithms. At the most basic level, "this measure is easily distorted by code cloning, a discouraged practice that leads to poor design and difficult-to-change code" (Erdogmus 2008, 5). Such obvious manipulations could be

machine-detected, but automated analysis could not distinguish 100 lines of inefficient code from 10 lines of elegant code that may have taken refinement to write. Furthermore, no comparisons would be possible between one programming language and another due to inherent differences in how many lines of code it takes to create one logical statement. As teams and individuals are increasingly versatile in the language chosen for a given project, this would yield significant statistical incomparability. Therefore, the simplicity of LOC has generally been rejected by modern literature in favor of counting function points (FP) as a metric of relative system complexity and normalized unit on which to measure.

Allan Albrecht published the first paper on an FP method while a Program Manager at IBM in 1979 (Behrens 1983, 648), in which he explains the improvement FP presents. The "productivity measurement avoids a dependency on measures such as lines-of-code that can have widely differing values depending on the technology" (Albrecht 1979, 84). The primary motivation for such a system was not evaluating individual engineers but rather estimating time and effort at the management level, since "at least 85 percent of the software managers in the world jump into projects with hardly a clue as to how long they will take" (Jones 195). Additionally, Albrecht examined a single organization within IBM, warning that while there is a broad desire to improve productivity, "comparisons between organizations must be handled carefully" as there are likely appropriate variances in processes and definitions (Albrecht 1979, 84).

Initially, Albrecht's FP system was a formula of inputs, outputs, inquiries, and master files, each weighted based on a discovered proportion to application function delivering customer value (Albrecht 1979, 85). Many implementations of FP now exist, based on a variety of statistical implementation differences (Maxwell 2001, 23).

Recent literature asserts that "function point metrics have become the dominant measurement instrument" in much of the world (Jones 2008, 73) and further that well-trained, certified manual counters of the most common systems have high levels of accuracy (Jones 2008, 79). However, criticisms of FP methods abound as well. Behrens analyzed many projects over two years and found that the number of hours needed per FP was higher in projects with more FPs, that is, "as projects become larger, their unit costs become higher" (Behrens 1983, 649). This indicates that there are factors affecting development time unaccounted for in abstract measures, such as the complexity of growing enterprise systems.

As a compensation mechanism, Albrecht initially allowed manual adjustment to the formula (Albrecht 1979, 85). While *The Mythical Man-Month* is not primarily concerned with productivity measurement, it may partially explain the need for such adjustment, examining the declining efficiency experienced when adding resources to projects due to managing increased complexity and channels for communication (Brooks 1995). Yet attempts to crudely address this declining productivity with size have been shown to backfire. "As past research

had revealed large diseconomies of scale, the trend in the banks was to break large software-development projects into smaller projects. However, these smaller projects' proportionally larger overhead made them less productive" (Maxwell 2001, 83).

In discussing the International Software Benchmarking Standards Group's simple formula – project delivery rate equals work effort in hours divided by project size in FP – it is noted that "such a metric does not take account of the different tasks undertaken during a project, each impacting on other tasks" nor the impact of different costs across the phases of the project (Flitman 2003, 382). The types of operations managed in an enterprise "differ so greatly that the relative values of the different outputs may legitimately be different" (Flitman 2003, 383). From this analysis, an evolved approach with discretionary weighting is formed that "may be appropriate where units can properly value inputs or outputs differently, or where there is a high uncertainty or disagreement over the value of some input or outputs" (Flitman 2003, 390), although such weighting flexibility allows substantial manipulation. Flitman's proposed calculations are based on a centralized repository of software projects to use for comparison.

However, another analysis showed that company and sector were the two greatest factors in productivity variance (Maxwell 2000, 82), concluding that company-centric project repositories serve as the best benchmark for valid comparability. Correctly capturing the work effort for these systems is not trivial

due to differences in whose time is counted and the mechanism. For example, in "one organization ... the total effort data available for the same project from three different sources in the company differed in excess of 30 percent" (Maxwell 2001, 23).

Engineers Make More than Code

Other authors step back from both LOC and FP systems, attacking the assumption underlying these systems that a software engineer's role is solely creating code that fulfills requirements. "While some people may be responsible for implementing features, others may play a supporting role -- helping others to implement their features. Their contribution is that they are raising the whole team's productivity -- but it's very hard to get a sense of their individual output" (Fowler 2003). For example, designing, testing, documentation, knowledge sharing, managing interdependencies, and learning new technologies consume increasing proportions of time, lessening that spent purely on code creation. Going further, software engineers could be considered "mostly in the human communication business" (DeMarco 1999, 5) due to the amount of coordination with project teams. "The entire focus of project management ought to be the *dynamics* of the development effort," but evaluation of people "is often based on their steady-state characteristics: how much code they can write" rather than how they truly contribute to the complete body of work (DeMarco 1999, 10). Jones also points out this evolution of engineer activities in his criticism of LOC, but sees it supporting rather than detracting from an FP system (Jones 2008, 72).

However, a rationale against measuring at all is the temptation to standardize procedures solely for the purpose of measurement, which could make the work less fulfilling (DeMarco 1999, 17), because "in such processes [as software development], non-repetitiveness is an essential property of the task" (Austin 1996, 106). Observing that "measurement schemes tend to become threatening and burdensome," DeMarco goes so far as to say management should not have any visibility to measurements, but instead that individuals should be empowered to self-improve (DeMarco 1999, 60-61).

In examining an individual coding competition, it was found that speed to completion of the best outperformed the average by over a factor of two (DeMarco 1999, 46). Nevertheless, evaluating engineer productivity requires assessing more than just quantity, whether in units of LOC, FP, or in this case, comparative speed. "Do [work-standards] take account of quality, or only numbers?" (Deming 1982, 21). In focusing on previous scholarship around software complexity, which could serve as an input for either of these models as a broad indicator of quality, Fenton asserts that "the search for a general-purpose real-valued complexity measure is doomed to failure" (Fenton 1994, 201), although "specific attributes of complexity, such as the maximum depth of nesting ... and the number of paths of various types, can all be measured rigorously and automatically" (Fenton 1994, 202). That is, while many meaningful measurements can be produced that can inform an intelligent understanding, there is no ordinal of quality into which all can be synthesized.

Fenton is frustrated that measurements are tweaked and correlated in an attempt to drive closer toward a comprehensive metric rather than accepting piecemeal measurements as intrinsically useful, writing that "an analogy would be to reject the usefulness of measuring a person's height on the grounds that it tells us nothing about that person's intelligence". He goes on to criticize the more complex systems derived from FP at its most basic as "analogous to redefining measures of height of people in such a way that the measures correlate more closely with intelligence" (Fenton 1994, 205).

Engineers are increasingly responsible for testing their own code. These tests may be automated, in which an engineer writes code that tests code, or manual tests in which a component is executed as a user would consume it. The decision on type of testing to use is impacted by tool availability, tradeoffs of time constraints, and place in the development cycle. Although such testing makes development take longer, it pays off in any system that must be supported, since finding a defect early is an order of magnitude less costly to fix (Vegas 2003, 3).

The amount of time required for testing could be accounted for in FP estimation, and tools are available to ascertain whether some form of testing is complete, but none sufficiently account for the type and quality of testing. This is one distinction between the more measurable short-term process and the "real productivity" it impacts (Erdogmus 2008, 6). A corollary argument would be for defects found after developer testing to be counted against the developer's productivity. However, other than a raw count, no clear scaled, comparable

measurement exists that can be understood by anyone other than a close observer familiar with the project. "In the case of an attribute like ‘criticality’ of software failures an empirical relation system would at best only identify different classes of failures and a binary relation ‘is more critical than’" (Fenton 1994, 201), providing no additional context.

Measurement in Agile Methodologies

The growth in popularity of Agile methods in software development in the last decade further complicates the use of measurement systems. One of the key principles in Agile is the notion that management trust "self-organizing teams" (Fowler 2001), in which the team commits to delivering functionality but the individual contributors choose what they can best do to fulfill that commitment. In a separate personal writing, the lead author of the first Agile thesis recognizes that it may be possible to measure a team's productivity in this environment, getting "a rough sense of a team's output by looking at how many features they deliver per iteration" (Fowler 2003) and the complexity of those features.

However, beyond the variation in how the individual engineers contribute, there is also the fact that Agile approaches focus on iteratively improving the system over time with dynamic planning. For instance, in the Scrum framework of Agile development, development teams estimate complexity in units of story points, a relative measure of how long each story will take (Schwaber 2004). While having high correlation to true delivered value, "coarse-grained

measures—such as those based on function points, user stories, story points, use cases, scenarios, and features—tend to be less uniform and more prone to low-value instability" (Erdogmus 2008, 5). While conceptually similar to FP, story points are expressly intended to be a rough, iterative estimate of how long it will take that development team to accomplish, rather than an objective measure. For example, if a team evaluating a new story has just completed a similar project using the same technologies, they would likely assign a lower relative story point value than a team for which it would be new territory. FP would not vary in this situation.

More fundamentally, Agile methods reject heavy upfront analysis, whether in requirements formalization or deliberate counting based thereupon. "Classical estimation methods need well defined requirements. Agile methodologies don't support this behaviour" (Schmietendorf 2008, 113). Agile practitioners typically use a non-linear set of values when assigning story points, such as 1, 2, 3, 5, 8, 13, 20, 40, and 100, "to avoid a false sense of accuracy for large time estimates" (Kniberg 2007, 34), and the process of assigning points is done quickly in an open team discussion. Indeed, in rejecting another metric model that requires difficult estimation of size, Fenton assents that FP solves this ambiguity by having size "computed directly from the specification" (Fenton 1994, 204), but Agile repudiates the need for such a detailed specification with the central tenet of "Working software over comprehensive documentation" (Fowler 2001).

An approach to ensuring efficiency and excellence can be found in practitioners of one Agile framework, Extreme Programming (XP). XP emphasizes rapid development cycles to respond to changing requirements, often recommend Pair Programming, in which two engineers develop code jointly on one computer. This technique is a linchpin of XP, as "it is dangerous to do XP without pair programming. One primary reason is that the pairs keep each other honest. XP is a minimalist approach, so it is essential that many of the practices actually get done" (Williams 2003, 177). Thus, completeness and correctness of work is enforced by professional pride, knowing the partner will call out deficiencies.

While practitioners of modern development frameworks may reject classical counting techniques like LOC and FP, it does not necessarily follow that an individual engineer's development activities must be entirely opaque, free from management control, or that one cannot be judged against another. As software metric supporter Tom Gilb said, "Anything you need to quantify can be measured in some way that is superior to not measuring it at all" (DeMarco 1999, 59). Multiple measurements, including counting techniques, may be integrated and normalized to assist forming a complete picture. While rejecting "measurement acquiescence", Erdogmus recognizes "context-dependent and proximate measures can still be very valuable" (Erdogmus 2008, 6) "provided we understand why we’re doing it, and provided we’re aware of limitations" (Erdogmus 2008, 4).

Defining Valuable Output

Whether it is desirable or possible to measure the business or other external impact of the output of individual developers is another matter of exploration. If one engineer implements fewer FPs in a period than another engineer but his result in higher profit, perhaps he could be considered most productive (Fowler 2003). However, engineers in large enterprises often have limited control over what products they work on, and many layers stand between them and the customer, so sales may not be a fair metric. More importantly, when large numbers of engineers contribute to massive systems sold for millions of dollars, examining a single person's business impact would be impossible. These attributes compound the fact that "employees true output (such as value to the organization) is often intangible and difficult to measure; in its place, organizations choose to measure inputs" (Austin 1996, 18) such as those discussed in the preceding paragraphs.

Research Procedures

In order to propose a system to evaluate software engineers in an enterprise setting, the author first analyzed contemporary industry views and implementation approaches by conducting interviews across a representative set of specialists. All interviews were conducted with associates of Cerner Corporation ("Cerner"), a major supplier of healthcare information technology solutions. Management at Cerner is decentralized, giving front-line managers substantial latitude to create and implement their own policies and practices. Therefore, while all interviewees were employees of the same company, it was expected that a wide variety of attitudes, information, and experience on the topic of engineer evaluation would be encountered.

Potential interviewees were solicited from internal corporate online communities of software engineers, software architects, technical project management, and senior management of development. Additional individuals were specifically targeted based on work history and responsibilities. Of respondents, interviewees were chosen who represented a wide sampling of roles and organizations. The interviews were conducted using the following questions:

1. What are the characteristics of a good software engineer, and is it really an "engineering" profession?

2. How can the characteristics of a good software engineer be best judged objectively?

3. What should the aspects of “productivity” be as applied to this profession? Is defect accountability part of that? (Defect accountability is a formal, enterprise-wide system Cerner has introduced and refined over the past two years to track on the person responsible for the presence of a defect, with aggregate reporting to senior management.)

4. How could we identify, track, and react to the vastly varying quality and quantity of engineers’ output?

5. Can any part of the output of an engineer be measured numerically by an outside observer?

6. How should evaluations be done? Should some form of peer feedback be a consideration in evaluating engineers?

The interviews and literature review are taken together with the author's experiences and reflections to form the basis of the new evaluation design, which the author plans to implement in the next review cycle.

Results

Interviews

Fulfilling the need for a sampling of roles and organizations, opinions on measurement were collected through interviews with each of the Cerner associates listed in Table 1 in October 2009.

Table 1. Interviewees

Name	Role	Organization
Brandon Heck	Software Engineer	Millennium Services
Steve Giboney	Technical Project Manager	Healthe
Katie Carter	Technical Project Manager	Foundations
Scott Schroering	Lead Architect	Millennium Services
Dan Plubell	Director & Knowledge Architect	Acute Care
Katie Lofton	Business Analyst	Development Operations

Brandon Heck

Brandon Heck's response to the question #1 focused on quality work, including proactively finding defects. He mentioned the work always needs to be "accomplished within a reasonable timeframe," although establishing such a timeframe is difficult when only starting with use cases or requirements. He also noted the many collaborative aspects of software engineering that make it a mix of art and science, taking it beyond “just pumping code.” For question #2, Heck

said evaluation is difficult since the aspects he identified of quality and speed can be traded off against each other. He saw some value in setting a timeframe for completion as long as it was reached with true buy-in from the engineer rather than being imposed. However, every miss should not be treated as a failure, as he noted “you can’t know everything about a project until you’re done with it.” The difficulty of counting defects was a further complicating factor.

Heck's response to question #3 was that an engineer’s productivity can be evaluated by a close observer, but he believes a metric to be impossible, primarily due to the mix of art in the profession he discussed in the first question. As individuals practice their craft differently, while there may be best practice guidelines, strictly defining various aspects of project completion such as the amount of testing would “adversely affect the culture” because “self-value would decline” in an environment of complying to minimums. He believed defect accountability could be useful but necessitates significant context into the nature and situation of each defect, as mere counts are meaningless.

For question #4, Heck recognized that "healthcare is complicated" and engineers often develop primarily in either breadth of functionality or depth of understanding. Such varying approaches to the problem domain yield different types of familiarity and output, which interact valuably in a team context while difficult to isolate to the individual contributions. Therefore, for question #5, he believed "a metric is difficult if not impossible, because too many things change, and engineers practice their craft differently." If useful measurements could be

created, Heck recognized trends would help indicate good and poor performance, but would need to be interpreted in context.

Responding to question #6, while generally saying, "I don't think performance can be measured objectively" due to its complexity, Heck nevertheless advocated for progress evaluation. Such evaluation should be done by someone close to the engineer throughout, rather than someone looking primarily at the finished product or a metric by-product. He thought an executive or other individual more removed from the daily work could be swayed by personal characteristics. Given the amount of collaboration and interaction between team members, he believes peer feedback using a system of structured questions as well as open-ended comments should also be integrated in a system.

Steve Giboney

Giboney's response to question #1 gave primary weight to engineers having the drive to solve problems, saying he is "not satisfied until they're innovative." He believes that software engineering is distinct from other engineering disciplines, as the field is not as "empirical" as classical engineering professions and there can be no single defined process. Instead, he prefers a framework to identify and respond to changes as quickly as possible. The main characteristic he looks for in question #2 is an engineer making use of teammates without being reliant on them, so that all can attain maximum productivity.

In question #3, Giboney immediately rejected counting LOC or FP. Instead, he believes that transparency -- for example, from using an Agile approach -- can help understanding of an engineer's productivity and highlight underlying issues or barriers, but one "can't estimate with any degree of accuracy" due to the unknowns and interruptions. Although his team uses Agile story points, they are only to help forecast and used only collectively, recognizing particularly that the larger the project, the less accurate the forecast likely is. His team does track individual defect accountability, but he expressed doubt that is the optimal mechanism to drive quality or motivation. Instead, he would like to experiment with approaches such as pair programming to identify issues earlier in a fundamentally different way, but has not fully explored the institutional implications and levels of support or barriers. He would prefer a metric that tracks not the mere presence of a defect but its implications, such as troubleshooting and support engineering effort required, wasted resource consumption, and client outages.

On question #4, Giboney indicated that "solving problems more elegantly than requirements points to a higher quality or more productive engineer." However, it is not possible to "predict or set out as measurement" what makes a solution elegant. Instead, evaluation of quality must be carried out by a familiar observer, such as in the context of code reviews. Similarly, responding to question #5, he thought individual measurement might be acceptable in trivial engineering tasks such as "very repetitive programming or report generation," but

not higher-order problem-solving. Instead, he believed measurement could be useful at a team level, tracking the performance of a component or the number of issues logged over a six-month period, with the team then interpreting.

Giboney's response to question #6 was direct: the team members themselves "know who pulls their weight and who they go to" for expertise. Therefore, peer feedback would be a helpful input to managers, who could combine this with their own opinion. As this is fundamentally a system of dealing with people, he believes it to be inherently subjective, not something expressible in numbers.

Katie Carter

Carter, for question #1, found engineering fundamentals in the need for a software engineer to think quickly and process information to solve problems. However, she drew a distinction in the significantly less predictable nature of software development roadblocks and how long a project will take to complete. Continuing to question #2, she believes that software architects or any others very familiar with the work have the ability to predict the amount of time a project will take, and project postmortems on missed deadlines could examine whether the estimate or work effort was off.

For question #3, Carter thought productivity evaluation to be quite straightforward, evaluating whether the engineer met the forecast set by the architect as previously discussed. However, beyond meeting that binary

condition, she thinks it is important how the work was completed, such as whether the engineer is “reasonably able to solve issues” encountered and provides transparent status updates to stakeholders.

Carter, in question #4, found no purely systematic way to identify quality. Tracking back defects to the originating engineer may be helpful, while outstanding performances must simply be subjectively identified, calling them out to the team for instructiveness and filing them away for performance reviews. On question #5, she said that any metric of code output may be "very valuable input" to a manager "very close," but it could not be used as a pure number absent that context.

Finally, for question #6, Carter does herself ask for peer feedback, additionally listening in to code and technical design meetings as sources of indirect information. She said the information gathered through those mechanisms on individuals as well as project-level consumer feedback, outage analysis, and postmortems to understand team successes and improvements must be "subjectively processed" by a manager able to "see through smokescreens."

Scott Schroering

Schroering's response to question #1 values those who plan well and "see the big picture," demonstrating a capability to envision the future, as opposed to those who code as they go since those engineers' projects tend to drag on. Part

of this quality is the ability to see problems ahead of time. He believes the biggest barrier preventing software development from being a more mature engineering discipline is the unpredictable client support work that constantly affects projects. On question #2, Schroering finds promise in the increasing use of Agile processes as a way to raise the visibility of progress and problems quickly, thereby gauging an individual's progress. He believes the setting of and accountability to daily goals under Agile can gain commitment from all participants and help “filter through the excuses.” He specifically contrasted such processes to the use of Microsoft Project, which despite its high degree of precision is treated as merely a loose guideline due to consistent inaccuracy.

Schroering responded to question #3 that ideal productivity cannot be fully expressed for a given project. Instead, engineers should “go the extra mile to identify existing issues while working on their project to reduce future work effort,” implementing the required functionality while testing effectively and considering the big picture. For example, the most productive engineer is one who can identify an issue with the requirements or technical specification early rather than simply implementing what is given, so wasted effort is avoided. In this spirit, he believes a system tracking defects back to individual can be helpful for the engineer to learn from mistakes through root cause analysis, but the aggregate reporting is not really helpful without knowing the severity of the problem or the comparative scope of the project in which it was created.

On question #4, Schroering had doubts on the ability of code reviews to identify quality as they "often aren't detailed enough." While he recognized Agile techniques such as pair programming would deliver the necessary detail, he believed it would not have management support as there is "not enough time" to put two engineers on one computer. He believed the best test of quality was having the code exercised in the field: if few issues occur, "it shows that desired holistic thought." For question #5, he believed there is "value to some extent in having an outside observer to find continuing trends – close team members might be more smoke screened by excuses, [since they are] involved in the day-to-day." He could also imagine doing only team-level tracking externally such as publishing project plans and measuring the achievement.

Schroering's response to question #6 advocated managers soliciting opinions and observations from technical and subject matter experts, but he did not believe peer input would be effective. Above all, he believed it important for an evaluation "to give constructive feedback even to good performers," and he thought peer feedback might be too kind. A tool that makes such feedback anonymous might abate that, but he thought an overall structure in which the best performers naturally "rise to the top" is the ideal team environment.

Dan Plubell

Plubell's characteristics for question #1 focused on mental agility and memory, such as a general curiosity to learn and "take things apart." In this, he found similarities with other engineering professions, but believes the software

field to have a much less well-defined skill set, such as varied languages and architectures. Responding to question #2, he pointed to task ambiguity preventing objective measurement, but he asserted that the important qualities he identified and general attitude could easily be observed by others with whom the engineer interacts.

On question #3, Plubell identifies that ambiguity at the outset of a project as making it undesirable to measure individuals against meeting an estimate. Nevertheless, he believes estimating is important. Since estimating is based on experience, knowledge, and judgment, a systematic approach to break down a project into units of work can help find similarities to past work. He believes a postmortem is important as a mechanism of continuous improvement, both at the individual level to estimate task time and the project level through a centralized database to track history. He is careful to note that such a system could be calibrated to drive good estimates, but different teams could not be compared, undermining the appeal of rolling up data to the organization level for performance review banding.

His response to question #4 saw little opportunity for systematic digesting of good and bad performances. Instead, both "take context," such as "the projects they're working on and the [type of] work they're doing," as some work is far more complex. The good can be filed away and celebrated in the review, while the bad may be learning opportunities. Nevertheless, when interpreted with sufficient context, trending may help identify continuing problems and

successes. For question #5, he echoed his response to the third question, pointing to the inability to normalize systems that have been calibrated separately as a barrier to metrics being comparable by an outside observer.

For question #6, Plubell believed a team-based, calibrated measurement "that serves as a proxy of reality" would be an ideal input for evaluations. Measurements would necessarily be digested in the context of the manner in which the work was done, considering less quantifiable attributes such as teamwork, communication, and attitude. Peer feedback -- perhaps cloaked by anonymity -- could be helpful, but he believes far more important is "an engaged manager" who is "observing the team." One measurement that has been proposed for Agile teams at Cerner is tracking the story points a team commits to and successfully delivers. His concern with asking either individuals or teams for estimates and then penalizing for misses is that padding would occur to make the numbers look good.

Katie Lofton

On question #1, Lofton indicated a good engineer is one who writes “understandable, efficient, and maintainable” code and constantly learns and improves. For question #2, she believed those characteristics could ideally be gathered as side effects of development, but stressed the need to have both a good process and good tools that support it.

On question #3, she believed productivity could be defined as the proper implementation of Minimum Marketable Features (MMF) within the prescribed time window while meeting defined quality measures, with defect accountability a tool used in evaluating quality. She rejected LOC and FP as abstractions, since neither relates whatsoever to value delivered to customers as MMF does, but also recognized that team difference in defining the size of MMFs would prevent organization-level comparisons.

She responded to question #4 by saying that the measure of quantity and quality delivered by an engineer must ultimately be the financial impact, as innovation must be marketable and actually implemented by clients. Recognizing the possibility an engineer might happen to be on a bad development team or a solution with a bad sales team, she might only use return on investment at a corporate level, but contribution as a resource at the team level. She stressed that "you have got to be able to use software engineers as a resource," with less latitude given to middle management for allocation. She addressed question #5 foundationally, asserting that using any type of measurement "empowers" engineers "even though they tend to object to it the strongest, since it gives some validity beyond an opinion." Additionally, she said, "any metric used consistently within one team has some merit," while recognizing the corollary that metrics are often incomparable between teams due to inconsistency.

For question #6, she advocated gauging productivity based on MMF delivery as an objective input into evaluation, but would not recommend the numbers being shared, primarily because "quantitative visibility within peer cohorts causes problems."

Interview Analysis

Before measurement can be considered, an understanding of the optimal traits against which measurement is being performed is necessary, which was the focus of the first two questions. While some consensus was found at a basic level, those in leadership positions all had substantially wider definitions of a good software engineer, giving more weight to attributes and approach that lead to career growth over the long-term rather than the week-to-week project deliverable. Additionally, subtle differences existed were exposed in further consideration. For example, Lofton, the only interviewee with no work history as a software engineer in formal development but whose role involves developing and tracking metrics to evaluate the development organization, had the most narrowly-focused definition of a good engineer. Overall, while all interviewees were able to quickly define attributes that could make one software engineer superior to another, none believed these most important attributes able to be tracked through metrics like LOC or FP. Similarly, all interviewees agreed that evaluations must be inherently subjective and should include substantial input from those close to the engineer, despite differing on the optimal sources, mechanism, and manifestation.

Responses to many of the questions varied substantially based on the nature of the work done by the interviewees' teams. For example, Brandon Heck's responses were representative of engineering on teams involved in new innovation, as they must begin work on projects with unclear scope and unforeseen hurdles. On the other hand, Katie Carter recognized that a lot of coding on her team is relatively more predictable due to similarity with past projects. Nevertheless, there were many similarities of opinion of technical practitioners even across such differences. For example, both Heck and Carter made it clear that consideration should be paid to the correctness of the code created in a project

The general appeal of measurement was also highly influenced by an individual's role and experience. Heck's opinions on measurement often recalled those of DeMarco that the danger of implementing them poorly may outweigh any possible benefit to be gained. Additionally, he called out those portions of the job such as writing tests that are discarded in the metrics discussed in the literature review, which focus predominantly on the implementation of functionality. Lofton's approach is from the business perspective of treating engineers as resources, with the desire to maximize the output of the investment in a project.

Conclusion

In considering the literature and interviews, it seems possible to implement a more mature and systematic approach to evaluating the productivity of software engineers, thereby improving the fairness of the performance evaluation system from the typical current state, while not becoming driven solely by numbers. Underlying this design is the recognition that both engineers and their managers would benefit from having inputs to the process to ensure the evaluation of productivity is not one merely of subjective impressions. Proper implementation also requires a substantial amount of delegation from executives to trust that the engineering managers are evaluating individual engineers effectively using the metrics, without detailed oversight. However, it does not follow that executives would therefore have no visibility into or control of the system; managers would be held accountable for their role in the performance of the team as a whole.

Since software development teams deliver business value in different ways based on development process and project type, any possible metric of work completion would not work for all teams. Whatever metrics a manager decides is appropriate for their team, it is important that they be meaningful proxies of reality, tempered by the recognition that they will paint an imperfect and partial picture. Collection of the measurements must not impose substantial overhead on anyone other than perhaps the manager. Agile development approaches in particular have no tolerance for work that does not deliver

customer value. Conversely, it is important that measurements not be chosen merely because they are the easiest to gather from whatever tools and processes the team happens to be currently using. Such an approach would both undermine the meaningfulness of the metrics as representative, and adherence to such metrics would yield additional inertia preventing the team from moving to improved tools or processes in the future.

One of the most important factors in ensuring the system does not become dominated by metrics is to communicate them only to the front-line managers of engineers, rather than creating a cross-organizational scorecard that might be rolled up for executives or even one shared within the team. The presence of such systematic reports would inherently communicate to both engineers and front-line managers that managing those numbers is the most important output, rather than maximizing business value. Additionally, such use of metrics would indicate a false comparability while also inducing a harmful normalization of work in order to more closely approach such comparability. On the other hand, the front line manager, as a close, informed observer, has the necessary understanding to digest the gathered metrics within the context of the individual, the team, and the project, including the non-measurable attributes of work.

Design

Therefore, this author intends to gather the metrics and inputs described below for his engineer direct reports over the next annual performance period.

Taken together, they help interpret both the "what" and "how" of an engineer's output. These measurements will not be exposed directly to the engineers on either an individual or group basis, nor will the executive levels be given visibility into them. The purpose of the metrics is to provide an objective backing to what is a necessarily subjective process.

Story point commitment and completion per development iteration

Broadly speaking, story point completion is the primary number gauging the output of development work using the Agile process, as the team-based estimating takes into account the size and complexity of the work to be done to implement a narrowly-scoped piece of functionality. While the estimate on any one story may be higher or lower than the actual engineering work needed, it should trend toward equilibrium and will certainly yield consistency across the team. This factor makes systematic tracking important to understand how engineers are truly performing, rather than sporadic notation of successes or failures that may be aberrant. The commitment an engineer makes per iteration is an important corollary to the completion number, as it provides an insight into whether that engineer is more often helping or hindering the entire team from making its deliveries.

Defect accountability tracking, including severity

Defect accountability is the process of tracking back all defects reported on released software to the original committer of the problematic code. While

Cerner's current approach is to report defect counts per engineer, the more meaningful metric would capture the severity in terms of what functionality was lost, how often it occurred, what the client impact was, and the context of the original coding. That last attribute is perhaps the most important, as a manager must apply judgment when comparing two defects that exhibit the same attributes of outward severity but one occurred as the result of carelessness in a straightforward project while the other was an unforeseen flow in an extremely complex project.

An annual anonymous peer feedback system

Finally, the feedback system will require all engineers to provide annual feedback about all others, with the manager getting anonymous, aggregated reports on each. Using a mix of discrete choice and open-ended questions, engineers will be asked to examine the work of their peers in the aspects of technical implementation, architecture and design decisions, team communications, flexibility, and leadership. The feedback on how an engineer is operating from a teamwork perspective is important so the environment does not become poisoned, and peer feedback can indicate this in a different way than management oversight alone. Gaining code-related feedback is helpful in getting a more detailed view that can only come from those who are constantly involved in each other's work through the process of code reviews.

Suggestions for additional work

The most apparent next step would be to track the performance of individual engineers and teams over a period such as a year, comparing three styles of productivity evaluation on different teams: one that relies on observation and informal feedback, another in which the above design is implemented, tracking metrics and formal feedback only at the manager level, and a third in which it is made clear that all metrics will be reported up executive channels. Each approach could be analyzed from the perspective of the team achieving its stated project goals, front-line manager feedback on their confidence of correctly understanding the productivity of engineers and belief that outputs are being appropriately measured, executive opinion on the performance of the team, and engineer feedback on the fairness and effectiveness of the system.

Depending on the outcome, incremental work may be warranted in improving the formal feedback system. For example, the wording and types of questions asked could have substantial impact on the outcome, so evaluating possible formats and implementing a comparative study with multiple teams could improve the utility of the system.

Additionally, there are structural influences on productivity beyond those reflecting an engineer's ability and application that are outside the scope of this paper. Further study of those tools and processes that enable and hinder

individuals from maximizing productivity would be helpful in achieving the shared goal of improved performance, while also providing the grounding necessary to resist the urge to constantly seek to adapt the latest fad on offer in the constantly evolving software field.

Bibliography

Albrecht, Allan J. “Measuring Application Development Productivity.” *Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium*, Monterey, California, October 14-17, 1979: 83-92.

Austin, Robert D. *Measuring and Managing Performance in Organizations*. New York: Dorset House, 1996.

Behrens, Charles A. "Measuring the Productivity of Computer Systems Development Activities with Function Points." *IEEE Transactions on Software Engineering* SE-9, no. 6 (November 1983): 648-652.

Brooks, Frederick P. *The Mythical Man-Month: Essays on Software Engineering*. 2nd ed. Boston: Addison-Wesley, 1995.

DeMarco, Tom and Timothy Lister. *Peopleware: Productive Projects and Teams*. 2nd ed. New York: Dorset House, 1999.

Deming, W. Edwards. "Improvement of Quality and Productivity through Action by Management." *National Productivity Review* (Winter 1981/1982): 12-22.

Erdogmus, Hakan. "Measurement Acquiescence." *IEEE Software* (March/April 2008): 4-6.

Fenton, Norman. "Software Measurement: A Necessary Scientific Basis." *IEEE Transactions on Software Engineering* 20, no. 3 (March 1994): 199-206.

Flitman, Andrew. "Towards meaningful benchmarking of software development team productivity." *Benchmarking: An International Journal* 10, no.4 (2003): 382-399.

Fowler, Martin, and Jim Highsmith. "The Agile Manifesto." *Software Development* 9, no. 8 (August 2001).

Fowler, Martin. "Cannot Measure Productivity." Martin Fowler's Bliki, entry posted August 29, 2003, http://martinfowler.com/bliki/CannotMeasureProductivity.html (accessed January 14, 2010).

Jones, Capers. *Applied Software Measurement: Global Analysis of Productivity and Quality*. 3rd ed. New York: McGraw-Hill, 2008.

Kniberg, Henrik. *Scrum and XP from the Trenches*. Toronto: C4Media, 2007.

Maxwell, Katrina D., and Pekka Forselius. "Benchmarking Software Development Productivity." *IEEE Software* (January/February 2000): 80-88.

Maxwell, Katrina D. "Collecting Data for Comparability: Benchmarking Software Development Productivity." *IEEE Software* (September/October 2001): 22-25.

Ridgway, VF. "Dysfunctional Consequences of Performance Measurements." *Administrative Science Quarterly* (September 1956): 240-247.

Schmietendorf, Andreas, Martin Kunz, and Reiner Dumke. "Effort estimation for Agile Software Development Projects." *Proceedings of the Software Measurement European Forum*, Milan, Italy, May 28-30, 2008: 113-123.

Schwaber, Ken. *Agile Project Management with Scrum*. Redmond: Microsoft Press, 2004.

Vegas, Sira, Natalia Juristo, and Victor R. Basili. *Identifying Relevant Information for Testing Technique Selection: An Instantiated Characterization Schema*. Dordrecht, Netherlands: Kluwer Academic Publishers Group, 2003.

Williams, Laurie, and Robert R. Kessler. *Pair Programming Illuminated*. Boston: Pearson Education, 2003.