The Paradox of Autonomy and Recognition

**THOUGHTS ON
TRUST AND MERIT
IN SOFTWARE
TEAM CULTURE**

KATE MATSUDAIRA

Who doesn't want recognition for their hard work and contributions?

Early in my career I wanted to believe that if you worked hard, and added value, you would be rewarded. I wanted to believe in the utopian ideal that hard work, discipline, and contributions were the fuel that propelled you up the corporate ladder. Boy, was I wrong.

You see, I started my career as a shy, insecure, but smart, programmer. I worked hard (almost every weekend), I wrote code for fun when I wasn't working on my work projects (still do, actually), and I was loyal and dedicated to my company.

MY PROJECT SHOULD HAVE BEEN JUDGED ON MERIT

Once, for six months I worked on a project with four other people—and I felt like my contributions in terms of functionality and hours contributed were at the top of the group. So you can imagine my surprise when at our launch party, the GM of the group stood up and recognized “Josh and the other team members for their hard work.” I stood there stunned,

thinking, WTF?! How was it that the GM was so out of touch with the team? Didn't our manager look at the check-ins and the problems being resolved? How did Josh, who had probably contributed the second-least amount to the project, end up being the only person singled out from the group? Not me, not the most senior person on the project, but the guy who seemed to spend a lot of time talking to my boss and the other people on the project.

Many of us have been there, though. We work tirelessly, doing our very best to get things done on time, but somehow we get passed over—in favor of someone else whom you honestly believe didn't contribute nearly as much as you did. What happened to meritocracy and being recognized for your work?

AUTONOMY MAKES MERITOCRACY IMPOSSIBLE

Fast forward to the present day. I work with a team of 18 amazing technologists, and it is my job to judge their performance. Only recently have I realized that in many ways it is nearly impossible to do so.

There is no objective, quantifiable way of doing this at scale without resorting to micromanagement—which is not something you even want to contemplate with a talented team. If this doesn't make sense to you, then let me describe various metrics people have suggested for judging performance (note that some of these are measurable, but others are more subjective):

- **Hours.** Ugh. I hate hour watchers. Writing code, at least for me, is like art, and when I am just not in the mood I

Penalizing your best players for having highly impacting bugs is the same as rewarding mediocre performance.

can't force myself to get things done. To use time spent as a productivity measure doesn't fairly represent the creativity and mentation required of a developer. Beyond that, it's not really all that feasible to track hours in a highly virtual environment. I love that people on my team can work from home—or whatever environment where they do their best work (this is the reason we have “no meeting” days)—but how can I possibly track someone's hours if they aren't in the office? Just like beans, counting hours sucks. Don't do it.

▀ **Lines of code.** This measure is flawed for many reasons—from the mantra that “the best code is the lines you don't write” to the simple anecdotal fact that it once took me three days to write a single line of code, while another day I wrote more than 10,000 lines of code (although, admittedly, part of that count included some substantial cut and paste). And, of course, deleting code can be very productive, too.

▀ **Bug counts.** Quality is obviously important, but finding bugs in production belonging to developers who otherwise write great code is not rare. This metric is seriously flawed in a profound way—it does not take into account that your best developers often produce a moderate number of serious bugs because they have been entrusted to design and implement the most complicated and critical pieces of an application/system. Penalizing your best players for having highly impacting bugs is tantamount to rewarding mediocrity.

▀ **Features.** Functionality is key, since when it comes to contributions, the features that are built into or added to the product should be directly tied to customer value. Of course,

judging on features can get complicated when multiple people contribute to one feature. Further, the details of the implementation can dramatically affect the effort and hours involved. For example, consider a recent project to add login to an existing site: implementing the feature using interstitial pages would have taken a few hours; however, the design involved using lightboxes, which increased the complexity around security and added days to the project to accommodate. Even looking at functionality and features as a performance metric can be misleading if you don't dive into the technical details of the implementation and its tradeoffs.

➤ **Maintainability.** It is hard to measure and track something as subjective as writing solid, maintainable code—but anyone who has had to struggle with legacy spaghetti code will tell you that maintainability is worth the extra time for code that will involve long-term usage in production. Coders who spend the extra time to write highly robust, maintainable code are often penalized for it, simply because their true contributions will not be realized until years later.

➤ **Building skills and knowledge.** How do you measure the benefit of the time invested in learning a new technology well enough to use it effectively; or researching and choosing the proper tools to optimize your productivity; or making careful and deliberate strategic choices that ultimately make a project feasible and successful? Obviously, these are critically important steps to take, but an outside perspective might point out that a lot more work could have been accomplished in the same amount of time spent developing skills and acquiring knowledge.

■ **Helping others.** Many programmers are great, not for the work they do but for the way they enable others to be great. Just having these people on the team makes everyone else better. Mentoring and selfless assistance to others are critical to building and preserving a highly productive and cohesive team, but quantifying an individual's role in such activities can be incredibly difficult, despite the reality of the contribution.

There are probably 101 more factors that could be used to judge programmers' achievements—including the way they present themselves (having a good attitude, for example), how dependable they are, or how often they contribute innovative ideas and solutions. Very few of these are objective, concrete factors that can be totaled up and given a grade—and it is difficult to do so without diving into minute details or micromanaging a project.

THEN HOW DO YOU GET NOTICED?

If there's no reliable managerial measure of your contributions, how do you get noticed? Really it all comes down to one thing: trust.

Trust is like a currency. When managers give their team members autonomy and independence, they are trusting them to complete the assigned task, making wise and strategic decisions along the way, while proactively communicating problems long before they become problems. These managers are, in fact, investing their money in the team, and when they see the returns on that investment, they, just like any lucky investor, are quite pleased.

Trust, though, takes time, patience, and consistency. If you can't build a relationship with your manager, all that work means nothing. For someone to invest in you, you have to show you are, in fact, an investment. Ask yourself these questions:

- Does your boss trust you?
- Do your team and your peers trust you?
- Have you done a good job to earn their trust?
- How would your peers describe you to someone else?
- How influential are you within your organization?

As a manager, I have been, at various times, very fond of a particular employee, but then noticed that this person's peers didn't care for him or her, or they held negative impressions of his or her performance. In these cases, given the trust level I have with my entire team, the opinions of the collective can easily outweigh personal preferences. Think of trust as a graph and each arc between the people you interact with as a weight—so when it comes to performance, those weights really matter.

Projects, products, performance, and companies aren't just judged on their output, but on how they produce the output.

In the example project I began this article with, the one thing Josh did differently was that he didn't just do the work, but he also made sure that the management—my boss and the GM—knew what our team was doing. In retrospect, he was the reason our project was singled out in an organization with so many people. At the time, I resented Josh; but now, many years later, I realize that his contributions to our team

weren't just his code, but also his communication skills and the way he did his job.

As an aside, though, certain company cultures may reward Josh's approach more than others. The problem with some people like Josh is that over time they can optimize on "trust" and create a distorted view of their contributions. This is what I mean when I say "office politics"—and this isn't good, either.

One of my very smart friends told me a story about joining one big company and meeting tons of super-smart, highly functional, and productive people who were all about creating trust with their superiors by being hyper-visible:

"They talked the most at meetings, they interrupted people, they sent extremely verbose emails at 3 a.m. detailing the minutia of a meeting that took place the previous day, they cc'd long lists of seemingly irrelevant but high-ranking people on their emails, etc. And their bosses loved them and they got the best reviews, etc. After meeting these individuals and being both amazed and disgusted by their shtick, it started to become clear to us that the whole culture self-selects to this type of person. It didn't take us long to understand why so much 'work' happens but so little gets done."

WHAT CAN YOU DO AS A MANAGER?

As an employee, I want to be judged by my contributions and be part of a team that is a meritocracy. I also want autonomy

Make
sure
to
recog-
nize
when you see
improvements
in someone's
development
areas.

and the ability to own substantial parts of a project and not have someone looking over my shoulder.

As a manager, I want to give recognition and praise to the people who deserve it, and I don't want to micromanage and spend my days being big brother.

This implies an implicit contract:

I will give you autonomy and independence, but it is your responsibility to share status and information with me.

For example, a team member once told me he had worked so hard and had really given it his best; from my viewpoint, however, his progress wasn't up to the level of his teammates. When he was leaving the company, he told me all these things he had done, and I asked him, "Why didn't you share this with me before?" With that information I could have advised him to spend his time elsewhere on priorities that were more important to the business. His response: "I thought you would know." Don't make that mistake.

It is also important that as a manager you recognize improvement. This means understanding a person's strengths and weaknesses. If you observe someone's performance and see substantial improvements in one of that person's development areas, then that is definitely worth recognizing. For example, if you have an amazing engineer who is typically a poor communicator, but who then steps up and contributes not just great coding prowess to a project but also keeps other team members abreast of evolving risk factors, those sorts of achievements are worth praise.

Make sure you consider all the factors of a person's involvement in the organization. Take steps to ask good questions and solicit feedback from other members of the organization. Finally, let each person know your expectations around communication and progress.

AND WHAT CAN YOU DO NOW?

My conclusion from all of this is this: if you want autonomy and the ability to own and control your own domain and projects, then it is your job to push information and build trust with your team members.

In other words, you need to learn and do the following:

- **Follow through.** Do what you say, and consistently deliver on your commitments.
- **Proactively communicate.** When a task takes longer than you thought, explain why.
- **Improve your communication skills.** In order for others to hear you, sometimes you have to hone the way you deliver your message.
- **Volunteer information.** Make an effort to explain vague or hard-to-understand ideas and concepts. Share the details of your decisions and diversions. This is also important when you make mistakes—letting others know before they figure it out on their own will show ownership of the situation and can prevent misunderstandings later.
- **Be forthright and authentic with your feelings.** Even when you may hold a contrary opinion, communicate your thoughts (respectfully and with tact).
- **Don't talk behind the backs of others.** It is very difficult

to build trust if someone knows that you will say something negative about your boss, the company leadership, or another coworker.

- **Be objective and neutral in difficult situations.** Learn how to be calm under pressure, and act as a diplomat resolving conflicts instead of causing them.
- **Show consistency in your behavior.** This is important not only in follow-through but also in eliminating any double standards that may exist.
- **Learn to trust your team members.** This is one of the hardest goals to accomplish, but trust is a two-way street. Giving others the benefit of the doubt and learning how to work with them is essential to a strong mutual working relationship.

In turn, you may be lucky enough to have a good manager who will be able to ask you good questions and take the time to understand your contributions. If that is not your situation, then make sure you are sharing information with those around you, such as your peers, your boss, and other stakeholders.

Good leadership means keeping everyone on the same page. If you want independence, then it is on you to make sure people know what you are contributing.

LOVE IT, HATE IT? LET US KNOW feedback@queue.acm.org

Kate Matsudaira is an experienced technology leader. She worked in big companies such as Microsoft and Amazon and three successful startups (Decide acquired by eBay, Moz,

and Delve Networks acquired by Limelight) before starting her own company, Popforms (<https://popforms.com/>), which was acquired by Safari Books. Having spent her early career as a software engineer, she is deeply technical and has done leading work on distributed systems, cloud computing, and mobile. She has experience managing entire product teams and research scientists, and has built her own profitable business. She is a published author, keynote speaker, and has been honored with awards such as Seattle's Top 40 under 40. She sits on the board of ACM Queue, and maintains a personal blog at katemats.com.

Copyright © 2015 held by owner/author. Publication rights licensed to ACM.

ACM – What's in it for me?

A lifelong Learning Center, with more than 4,000 online books and instructional videos to help you up your game, from Agile Practices to XML.

Join ACM today at acm.org/join

BE CREATIVE. STAY CONNECTED. KEEP INVENTING.

Association for
Computing Machinery

Advancing Computing as a Science & Profession