Henry II on Vacation - Why Teams Matter

Posted At : July 27, 2022 12:26 PM | Posted By : Mark Kruger
Related Categories: Business Of Development

There are thousands of applications built by one brilliant programmer. It plays out like this: Bob, the founder of Acme, has a great app idea. He hires "a computer guy" - that's how my mom describes me, a guy who "works with computers" like I was an employee at Best Buy. This "computer guy" is fantastic. Let's call him Henry II because I just watched "The Lion in Winter" and Peter O'Toole and Katherine Hepburn are soooo good together. Where was I? Oh yes, Peter... er... Henry is a guru-level programmer. He thinks along with Bob and builds to spec. He partners with Bob - the idea guy - to help the product succeed. The product grows and becomes wildly successful. In short order, it is obvious to Bob and Henry that the application needs more help than Henry can provide. So they set out to build a supporting cast of developers around Henry to share the load. This scenario plays out thousands of times every year all over the world. A superstar needs help and help is hired. And that is where our real drama begins...

Henry is not just good - he is Tom Brady good. Despite his obsession with the Aquitaine and his chiseled features and commanding voice, he's a terrific programmer. He's smart, assertive, and a problem-solver of the first order. He has a broad skill set and true expertise in his chosen languages and platforms. He digs into tasks with high energy, but he also sees the whole picture. He knows the product, its uses, and how it makes money. As a team of one, he delivers on his promises. Bob is pleased as punch with him. However, when Bob adds 3 or 4 folks to the team, development flounders. Rather than increased productivity, there are many new problems to deal with. Henry hints that the talent Bob hired just isn't up to the challenge. Team morale suffers and office drama increases. What is happening?

In fact, the above scenario is as predictable as death and taxes. Bob assumed that adding resources would duplicate and enhance the development effort. Before he had one developer, now he has 4. He should get 4 times the productivity, right? He wasn't prepared for "Team" vs "Superstar" dynamics. The problem lies both with Henry and with Bob's expectations.

Henry, as the creator, is accustomed to controlling the app and its progress. He dives into problems and does what needs doing. He wraps his arms around tasks that demand attention. Because he's been doing everything, he fails to differentiate between effective and ineffective use of his time. As the workload increases, he spends too much time working on the application code and not enough time working on the product at a higher level. He should be working on strategy, direction, planning, prioritizing, and mentoring (the stuff a lead usually does for a team). His domain knowledge is being wasted on break fixes and operations. He muses, "That new developer, Sally, could do this, but it would probably take her 2 hours because she's never looked at this part of the application. I know I can do this in 15 minutes. I'm just going to do it." He's trapped in a pattern that is killing his team. A quick real-world illustration might be helpful.

An Acme process synchronizes files from a network directory to the web server. This process watches the network directory. When it sees a new file it copies it to the web server - easy peasy. Customer service publishes new content onto the server by dropping a file in a local directory. The trouble is that every 6 or 7 days, one of the files "hangs" and is not copied over. No one has discovered why. The workaround is to log into the synchronization software and restart it. This kicks off the synch process and the file is finally copied.

One week, while Henry was touring the low countries, this problem recurred. A developer on the team (doubling as Sysadmin) handled the ticket, but she could not ascertain what was happening. She wrote a note back to customer service, telling them this task would have to wait until Henry returned from Denmark. This is not an isolated event at ACME. Henry was often called in to save the day because he knows all the who, what, when, and how. After all, he built most of these processes and applications. But let's ask a few questions in the aftermath.

Wrong Resource? The first question is, why is a lead developer troubleshooting production file synchronization? There is probably a case where this is necessary, especially when a team is too small to have a lead at all. That's not the case here. Is this the best use of a key resource?

Documented Process? Given that this is a recurring problem, where can developers or sysadmins go to search for a fix? This routine should be documented so that anyone tasked with support, even a developer, could resolve the issue. By the second time someone had to fix it, he or she should have opened a wiki page for "server content file synchronization" and added a section titled "troubleshooting a hanging file copy," along with the fix.

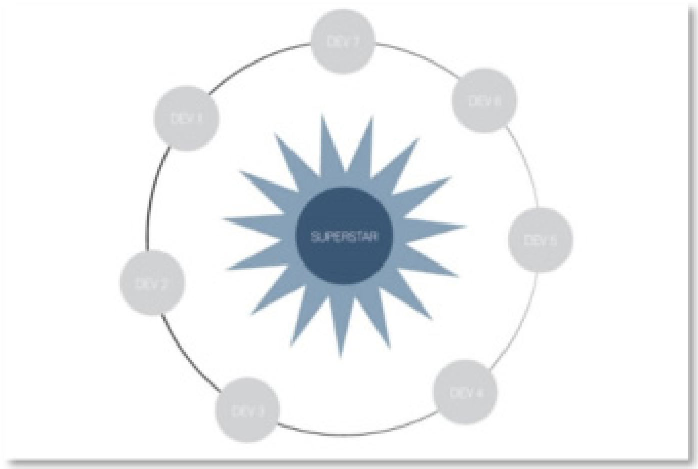
Ownership? If this had been one of my team members who bailed until "Henry comes back from hunting grouse" I would have questions – and not just "what's a grouse?" This is a trivial problem. It's not going to bring down a server or cause a cascade of failures. In short, it's a manageable, solvable problem within the grasp of any competent IT person, even if they know nothing about the synching process. For example, you could simply copy the file directly to the server. Why not resolve the issue (even temporarily) *without* bothering Henry's fancy-pants brain.

This brings us to our fourth related question. Why are the developers, tasked with support, bailing on such a simple issue? Here are two possible reasons:

- *Empowerment* – They did not feel empowered to resolve the issue. Perhaps they feared being locked in the Tower of London if the solution was not the one that Henry endorsed. More likely, they've been dinged repeatedly for not doing it as fast or in the same way as Henry. On returning, there's every chance that superstar Henry would *re-do* their chosen solution so it was done "the right way". They are gun-shy and do not feel empowered to take ownership of the problem and its solution.
- *Passivity and intransigence* – On large teams, developers often get away with punting. Software is a black box to most end users. They don't know whether a problem is trivial or really does need Henry's turbo-charged aptitudes. Meanwhile, developers often prefer to narrow their focus down to just the tasks they most enjoy. In those situations, and given a superstar resource who knows everything, developers do not reflect the same urgency as end users regarding bugs and service issues. They know it will be resolved without holding them to account. They also know they'll be dinged if they don't resolve it in a certain way.

Henry's chosen role (and personality) serves as an enabler for this unhealthy dynamic to occur. As long as he's the repository of all knowledge and the go-to fixer, we can't resolve this problem. Fixes, routines, process idiosyncrasies, and procedures are locked in his head, but he no longer has the capacity to manage that information. There's just too much to do and too many fires to extinguish. Such information must belong to an **institutional** knowledge reservoir, not an **individual** knowledge reservoir. With all his star power, his role makes him a bottleneck. When building the team, Bob has added to his responsibilities without removing his ownership of the application and knowledge. His capacity is far less than it was as a team of one because he's busy doing his previous job while making a pass at doing the new job of leading a team. His developers are less productive than they could be because he is so good at *their* job he can't give it up.

Meanwhile, Henry has "feelings". He built a successful application that is growing and thriving. He's been treated like an expert and given high status. He's the "go-to" guy for problems. He's been told over and over he's the magic man. "What would I do without you!" Bob is fond of saying. Of course, some of this is true right? He *is* an expert and a great problem solver. But Henry, like nearly all developers, conflates his ability to learn, move fast, and get things done, with certainty about how things *aught* to be done. He becomes doctrinaire - the pontiff of the church of Henry the Divine Expert.

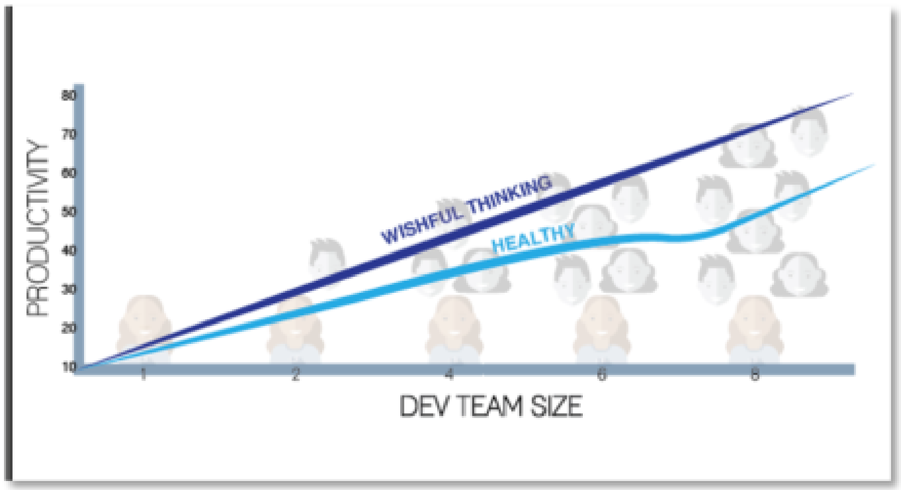


This is the "superstar programmer" model. One guy is head and shoulders above everyone else. He knows more, does more and everyone orbits around him. He works with excellence and speed, but he undervalues his team because they do not do things the same way he does - or as fast. So he keeps doing the wrong job. He takes so much onto his plate that his speed and knowledge are self-defeating.

To get the most out of a team it must be organized around both talent, personal growth, teamwork, and reasonable expectations. Rogue superstars upset and disrupt the team dynamic. Henry is shooting himself in the foot by doing too many things and especially too many trivial, operational things that could be done by anyone. And then there's Bob. Bob has been conditioned to expect superstar-level work out of each developer. But it's tough to find superstars, and they don't work well together. It's more likely he has hired competent developers to be on the team who are NOT Henry-level savants. So, Bob's expectations for team productivity are wildly unrealistic.

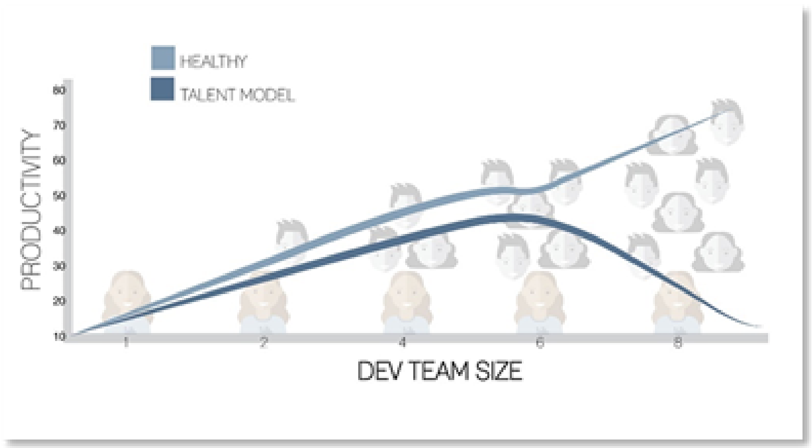
Many people are surprised at the productivity math of team development. If I can gain 100% productivity from a single developer, I should get 200% productivity out of 2 developers, 300% out of three, and so on. In other words, 3 developers should be able to accomplish 3 times as much as one developer - right? Not only is that wrong, it's not even close to reality. You are not duplicating a self-contained, individual development effort. You are changing *the way you do development*.

A team is a network of individuals who work toward goals together. They must interact constantly. They must be careful not to overlap. They need complete knowledge of the tasks of others, so they don't change something that affects another member. They must meet and plan. They must constantly share knowledge and come up to speed on each other's work. With each new team member, this problem is exacerbated as new connections, meetings, planning, and divisions of labor are made. New team members exact a cost on existing team members because they must be brought up to speed on the code and process. This is the exact problem that "project management" and methodologies like Agile and Scrum are meant to help mitigate. A team of 4 developers may operate at 60 or 70 percent productivity (per developer). As a team grows, economies of scale kick in, and the productivity curve levels off. The pattern resembles this chart (I apologize - I do not remember where I acquired this chart, it's been a while).



Note that this inefficiency is a *healthy* curve. It's the best-case scenario. If you have done well at team building, you can expect this reduced level of productivity as the *best possible outcome*.

But what happens when we have a superstar model? At 3 developers your superstar is managing the responsibilities of his own work plus 2. At 4 developers the problem is at a breaking point. The curve dips down. There is not enough supporting work to divide up among the secondary developers. The superstar takes on more and more of the mission-critical work himself. Support tasks and management tasks build up and your superstar becomes a fireman. His whole world is putting out fires all day long, and he is behind the curve on each task. Meanwhile, he does what he's always done while sniping at his team members.



The Fix

Teams and institutional organization really matter. Evolution is fine for birds but software developers tend to propagate mutations that are not beneficial. Make sure knowledge is systematized and institutional. Don't be seduced by the superstar. To get the most out of her or him you will need to build a well-supported, systematic SDLC accompanied by strong roles and divisions. Be on the lookout for doctrinaire attitudes about how things are done. If she has been with you a long time you will need to get her buy-in for offloading tasks. You'll need *accountability*. She will need to stay in her lane. If she retains ownership of everything your team will suffer. Finally, prioritize between the urgent and important. If you *do* have a superstar, chances are your best use of her talents will be at a higher level than troubleshooting operational issues - unless she's a sysadmin at heart (in which case hang on to her like grim death).

If you choose to make your superstar a lead, think hard. The soft skills - the ability to manage talented folk with egos as large as their own - are often more important than technical competencies in such a role. There's a reason Tiger Woods would be a terrible swing coach. Yes, a lead must be a strong developer, but she will spend most of her time in communication, mentoring, peacemaking, commiserating, and calling devs to account without burning down relationships. All those things are skills that do not come easily to a superstar developer. As someone who employs many them, I know from experience.