Henry II on Vacation - Why Teams Matter

Posted At : March 15, 2018 12:23 PM | Posted By : Mark Kruger
Related Categories: Business Of Development

"Just add more resources..." is a comment I hear quite frequently in our corner of the tech world. It is often thought of as an easy "solution" to IT challenges. Unfortunately, adding additional developers can often result in further bottlenecks. The following is a real life example (the names have been changed to protect the innocent)!

Let's talk about Lead developer Henry II and his role within ACME Company. In spite of being obsessed over ownership of the Aquitaine and looking vaguely like Peter O'toole, he's a terrific programmer, smart, aggressive and a problem solver of the first order. He tackles tasks with a great deal of energy and seems to be able to see the whole picture. Were he the only programmer (a team of one) these qualities would serve him well to get the most out of what he has to offer. As it is, these qualities combine with ACME's development model to serve as a constraint to efficiencies of the team.

Henry, due to his sense of ownership and responsibility, does what needs to be done. He wraps his arms around tasks that demand attention, sometimes without differentiating between effective and ineffective use of his valuable time. He spends too much time working "within" the projects and and not enough time working "on" them – meaning the strategy and direction and planning and mentoring (the stuff a lead usually does) – where his domain knowledge is the most valuable. A quick illustration might be helpful.

ACME has a process that synchronizes files from a watched directory on an internal share out to the servers at the NOC. The process allows customer service to get new content onto the server by simply dropping a file in a local directory. Periodically one of the several dozen directories has a problem. A file to be synched "hangs" and is not copied over. No one knows why, but it is always a file in the same directory. The fix is to log into the synchronization software and reset it. This kick starts the synch and the file is then copied.

While Henry was touring one of his estates in the low countries this problem re-occurred. A ticket was written and a developer (doubling as Sys-Admin) looked at it but could not readily ascertain what was happening or find a way to fix it. The result was a note back to customer service that this task would have to wait till Henry the Armada returned from Denmark.

This event is not an isolated sort of event – Henry getting called in to save the day because he knows all of the who, what, when and how. So let's note a few things at work here in the aftermath.

Wrong Resource? The first question is probably, "why is a software programmer troubleshooting file synchronization?" There is a case where that is necessary – usually when a team is too tiny to have a lead at all. That's not the case here so, is this the best use of a key resource?

Documented Process? Is there a place where developers can go and search for troubleshooting tips on the synch process? This was not a new problem, it occurs periodically. The fix should be routine and documented so that anyone tasked with support, even a developer, could resolve the issue. By the second time someone had to fix this he or she should have opened a wiki page for "server content file

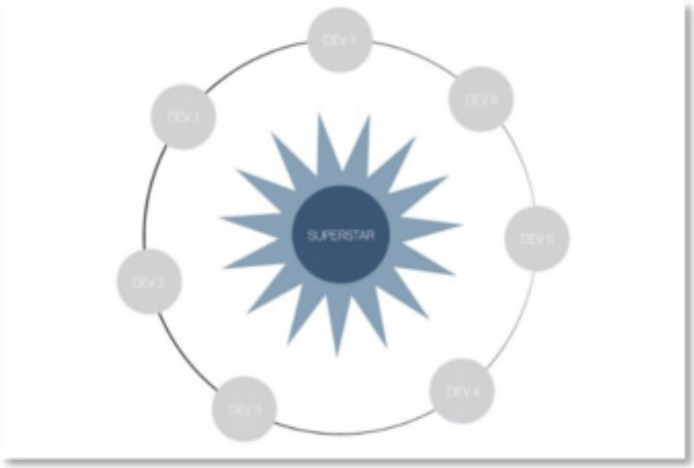
synchronization" and added a header called "troubleshooting" along with mitigation steps.

Troubleshooting 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 in the ham sandwich is a grouse?" This is not the sort of problem that has the potential to bring down the server or cause other problems down the road – i.e. it's a manageable, solvable problem with the knowledge at hand, even if you know nothing about the synching process. For example, a developer or sys-admin could copy the file directly to the server.

This brings to mind a third related question. Why are the developers tasked with support bailing on such a simple issue? There are probably two reasons.

- *Empowerment* – They did not feel empowered to resolve the issue. Perhaps they feared being locked in the Tower of London if their solution was not the one Henry endorsed. If this is the case then some mentoring is in order and some rethinking about roles.
- *Passivity and intransigence* – in large applications developers "get away" with punting. Software is a black box to most end users and developers can narrow their focus down to just the tasks they most enjoy or that they are the most comfortable with. In those situations and given a superstar resource (Henry) who knows and does everything that needs to be done through his own high level sense of responsibility, developers do not reflect the same urgency of end users on bugs and service issues – knowing it will be resolved without holding them to account. Note – this matters if you are going to have your developers supporting end users. They have to adopt urgency and take ownership of an issue.

Also note that Henry's role (and personality) serve as an enabler for this to occur. As long as he's the repository of knowledge and the chief fixer we can't resolve this problem. Fixes, routines, process idiosyncrasies and procedures belong to **institutional** knowledge, not **individual** knowledge. His role actually makes him a bottleneck for work and support rather than shortening the time to fix and deploy things. His developers are less productive than they could be because he is so good at his job.

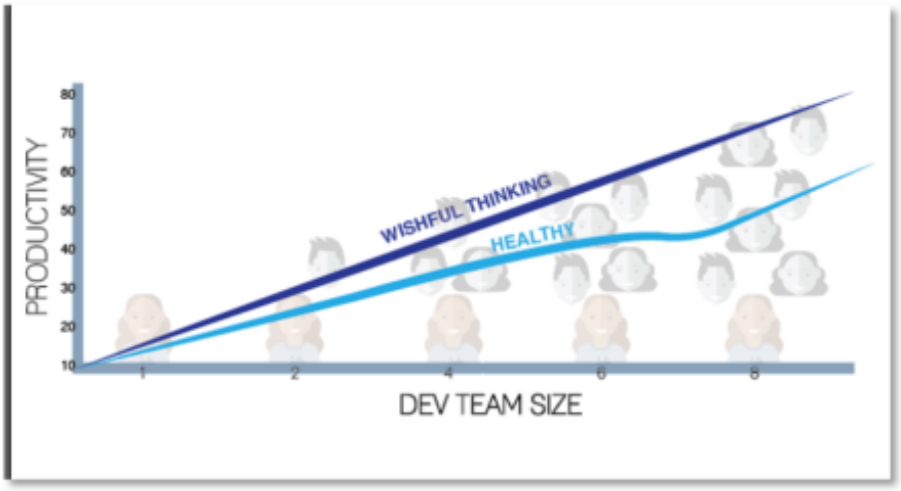


I call this team model the "superstar programmer". One guy is head and shoulders above everyone else, knows more, does more and everyone orbits around him. He does everything with excellence and alacrity but has so much on his plate that his speed and

knowledge are self-defeating.

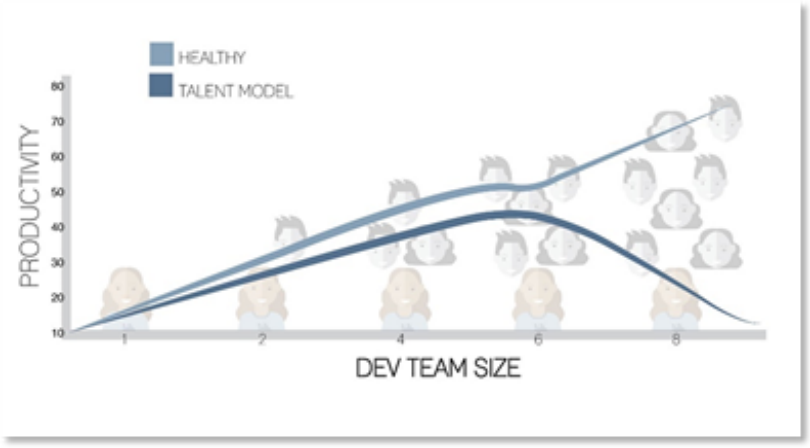
Teams matter. To get the most out of them they have to be organized around both talent and personal growth. Henry is doing too many things and especially too many trivial operational things that could be done by anyone. Those operational things need to be documented.

Let's discuss team productivity. Many people are surprised at the math of team development. If I can gain 150 hours of development, per month, from a single developer I should, theoretically, be able to get 300 hours per month out of 2 developers, 450 hours out of 4 etc. Reality doesn't work that way. A team is a network of individuals who work toward goals together. They must interact so as not to overlap. They have to meet and plan. You may not be able to divide the work up into discreet 150-hour chunks. With each new team member this problem is exacerbated as new connections, meetings, planning and divisions are made. So, a team of 5 developers may be 60 or 75 percent as effective as a team of 1 or 2. At some point economies of scale kick in and the productivity curve levels off. But the pattern resembles this chart.



Note that this inefficiency is a healthy curve. If you have done well at building your team you can expect this reduced level of productivity as the best possible outcome.

But what happens when have a rock star model? At 3 developers your rock star is managing the tasks and responsibility 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 rock star takes on more and more of the mission critical work himself. Support tasks and management tasks build up and your major talent becomes a fireman. His whole world is putting out fires all day long and he is behind the curve on each project and task trying to juggle assignments while he's still doing what he's always done.



The Fix

This is why teams and institutional organization really matter. Evolution is fine for birds but software developers tend to propagate mutations that are less rather than more beneficial. Make sure knowledge is systemetized 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 system - otherwise she will fly off the rails as you allow her to take ownserhip of everything. Finally, prioritize between the urgent and the important. If you *do* have a superstar, chances are your best use of his talents will be at a higher level than troubleshooting operational issues - unless he's a sys-admin at heart (in which case hang on to him like grim death).