Ensuring Quality with a Quality Team of One

Kim Janik

kim.janik@intel.com

Abstract

You are brought in at the start of a project as the quality lead, tasked with bringing up the quality team, standards and processes for the project. The development team is already fully staffed and churning out code like crazy. Surprise! The quality team is…you. How do you proceed when you are the only quality engineer on the team? How can you meet management expectations of a quality product? Learn five methods for improving quality on your team even if you are the only dedicated quality resource for your project.

Biography

Kim is a Sr. SDET at Intel Security, advocating quality, process improvement and teamwork. She received her M.S. degree in computer engineering from Portland State University. Kim credits her success to her passion for learning new things as well as her diverse background. Her varied roles have included ScrumMaster, Quality Champion, Validation Engineer, Software Engineer, Hardware Design Engineer, Computer Science Instructor and Zombie Apocalypse Survival Instructor.

1 Introduction

You promised yourself you would never again be the only quality engineer on the team. You asked the right questions during the job interview. You got to hear from the team members how happy they were working there. The hiring manager guaranteed you could hire a rock star validation team to meet all of the project goals. Little did you know that the budget would be cut between the time you were hired and the time you started. Does this sound familiar?

Unfortunately, this scenario isn't uncommon. Talking with engineers, they will grudgingly agree it has happened to them before. So why do we sweep it under the rug instead of addressing it? Has it become so common of a problem that we ignore it? Why are there not dozens of papers on the topic? Having a one-person quality team happens to the best of us. It is important to note that this is not the ideal situation. This paper is in no way advocating anyone should actually run a team with a single quality member as that would be taking the "small and scrappy" concept too far. In the long run, it is more beneficial to try to change the underlying mindset that often leads to this situation instead.

However, it is important to acknowledge that this problem does actually happen. As the newly hired quality lead, this now becomes the reality you need to face. In the worst cases, upper management still has the same expectations except you have less resources than you originally anticipated. Your first deliverable may be to provide a schedule of when quality milestones will be achieved. So, rather than give up, how can you still provide the best quality possible giving the lack of expected resources?

Before deciding on a course of action, you need to do some pre-work. Early in your arrival to the project, take the time to do a good, honest assessment of the current quality situation. Sometimes, there genuinely are no processes, procedures or tools in place. Take careful note of the people identified as part of your project. Next, list where you want to be in the future. This may or may not align with management's requirements so note any discrepancies. Next, organize your list from least to most important. Then, put estimates on each line item regarding how long it will take to implement with the current team (i.e. you). Be brutally honest. You are not doing yourself any favors by underestimating your time. At the same time, don't pad the work, either. If there are tools, licenses or other hurdles in the way, note those.

The rest of the paper presumes you have already evaluated the present quality situation and are familiar with the work being requested of you. This paper also expects that you are using Scrum or similar Agile methodologies. If that is not true in your specific case then adapt the suggestions to your situation where possible.

2 Ask for More

Sometimes, the first steps are the most obvious. When you don't have enough people to do the work you've been given, ask for more people. This is a tough scenario as it's likely that something has changed in the organization since you've joined. Perhaps the budget got downsized. Perhaps the expected team members left. Perhaps other extended job offers fell through. But, it doesn't matter if the original goal was to have a team of one hundred engineers working under you. Currently, you have one. It will never hurt to ask the question:

"Can more people be hired?"

If the answer is "no" because there's a hiring freeze, find out if there is an end date for when it will be lifted. If you are told "no" then do not hesitate to ask when you should follow up on the topic again. Also, consider asking if it's easier to hire in a contractor or consultant on a temporary basis to help jumpstart the work. While you may not want to give the impression of being a nuisance when you're brand new, you also don't want to leave anything on the table, either.

In many cases, it helps your case to state facts. The easiest way to do this would be to point to an existing document as a reference. You can tailor this to your particular situation, but an example might be:

> *"I was reading an article on improving quality that suggests the ideal developer to tester ratio is 1:1. It seems we are behind the industry curve. Is this something we can address?"*

In the best case, this will be sufficient to point out the imbalance. However, engineers as a whole are data-driven and more persuasion might be required. You can start by evaluating your team's efficiency. If you already have historical data then use it as a basis for your calculations. If not, use the most recent information you do have.

For example, let's suppose you have a fairly new team where the definition of done (DOD) was readily adopted and is being consistently followed. Each sprint is two weeks. On your team, the average developer has a velocity (V_{dev}) of two small user stories per sprint. Given that you have dedicated half of your time to test infrastructure development, you have a velocity (V_{qa}) of four small user stories per sprint. Start by working out deficit on a per sprint basis (D_{sprint}) based on the number of developers (N_{dev}) and testers (N_{qa}) you currently have:

$$(V_{dev} * N_{dev}) - (V_{qa} * N_{qa}) = D_{sprint}$$

Independent of the resulting numbers, this will give you a rough idea of the work facing you. You can extrapolate this out to an entire quarter ($D_{quarter}$) or even the next year (D_{year}).

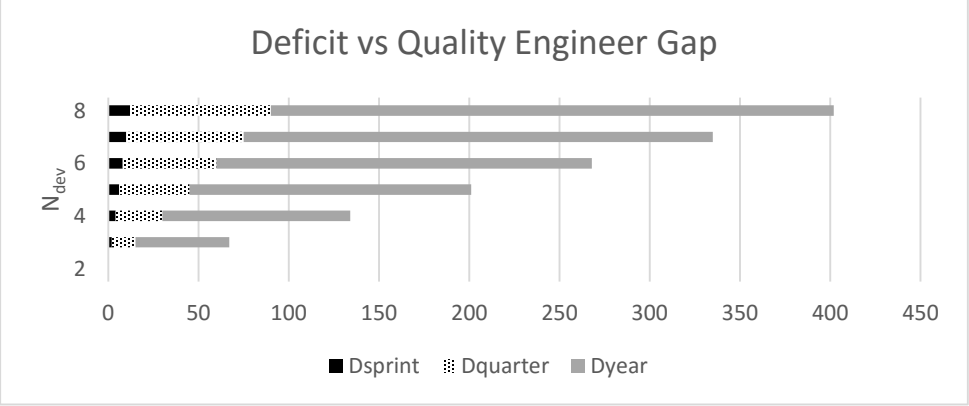
$$D_{sprint} * 6.5 = D_{quarter}$$

$$D_{quarter} * 4 = D_{year}$$

The following table gives you an idea of the impact as the number of developers increases.

N_{dev}	N_{qa}	V_{dev}	V_{qa}	D_{sprint}	$D_{quarter}$	D_{year}
2	1	2	4	0	0	0
5	1	10	4	6	39	156
8	1	16	4	12	78	312

You may find it useful to graph this data.



This is a painful trend to see, however it's a good example of how you can use real data to make your case. You can explain how many quality engineers you still need, given the team's current state. Is this the only equation you can use? Of course not! Only you understand what data is most likely to sway your company. Don't hesitate to bring in 'soft' figures as well.

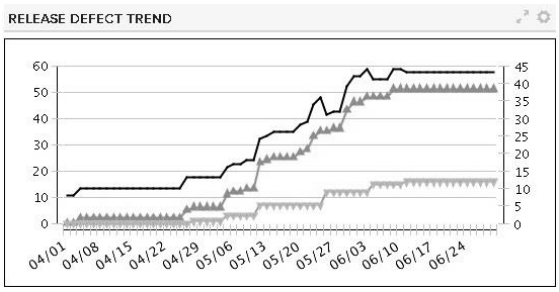
3 Expose the Deficit

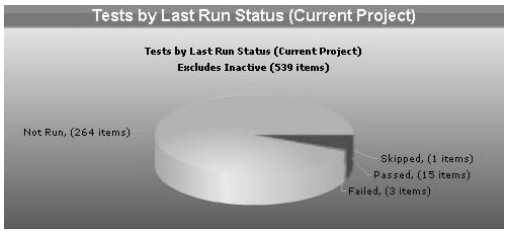
At the heart of this section is one important word: communication. You are in a difficult position. A good team will understand this and appreciate knowing what you are accomplishing for them. It is also possible that your team will need to be educated on the current situation. No matter what, you want to ensure open communication with your team, management and stakeholders. Your goal is to make as many people as possible aware of the work you are engaged in as well as what you are trying to accomplish going forward.

Similar to what was discussed in the last section, there is another obvious step to take when you don't have enough people to do the work you've been given. Reduce the amount of work. In most situations you can negotiate the amount of work that needs to be accomplished. Perhaps that complicated regression suite can come off the table early on. Maybe the formal UI testing can be de-prioritized until resources are available to complete it. At the end of these negotiations, you are likely still going to have an overwhelming amount of work to accomplish "sometime".

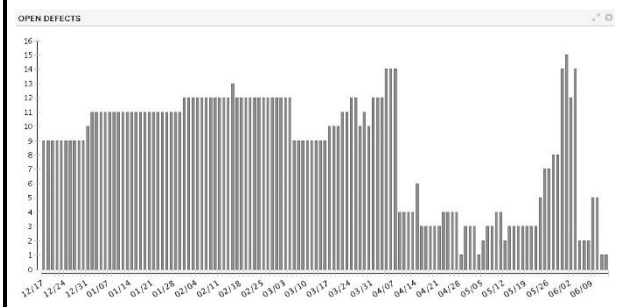
You're probably doing this already, but start publicly documenting the amount of work to be done. This is useful for two reasons. First, it can be very informative in showing why some work is not being completed. Second, it puts the focus on everything you've accomplished for your team. You want full transparency in the state of quality for your product. Not only is this a great work habit but it's also laying an information foundation that you can build on when you get to your other options in the sections below.

Quality reports can be an effective tool for communication. First, select a cadence. Emailing once a week, bi-weekly or monthly are all solid choices. Whatever you choose, be consistent in your mailings. Next, select your indicators. Only you can determine what is most critical to your team's needs. Some common indicators can be open bugs, test cases written, tool completion countdown, percentage of test cases automated. A few examples are show below.





Test Benchmarks	2/29/13
Test Cases Converted to Automation: %	5%
Test Cases Passing: %	50%
Unit Test Coverage: %	25%
Code Path Coverage: %	0%



It is important to note that this is not supposed to be a chance to complain or be negative. This is about facts. Yes, point out the deficit. But, don't forget to include the positive trends, too! Include your manager on these. Consider including your second-level manager as well.

If your team is employing Agile methodologies, you may be able to leverage them to help bring attention to quality. This can provide a baseline for user stories going forward, but it has an additional benefit. If the quality metrics you've laid out in the DOD are not met, your product owner will be faced with a decision – ignore the quality metrics to finish the work or delay the new feature to ensure quality. If the team doesn't already have a DOD or exit agreements, consider volunteering to help put those into place. You can include line items that will help the team align with the quality goals.

Last, there is one more skill to master in your communications. Learn to love "No." In your calculations from the above section remember you included time for test development. That is critical. To make progress, you need to keep developing better processes and procedures. You will hear, "can't you just skip this one week?" You might get a request to reinstate previously removed quality items. Learn and use the important one-word phrase. As these type of instances come up, simply say, "No" and point them back to the data if necessary. This does NOT imply rigidity; instead, it means that each future conversation becomes a trade-off to ensure your team is able to meet the commitments they agreed to.

4 Enlist Existing Help

By now, you might be feeling pretty discouraged. Your assignment has gotten off to a rough start. The amount of work hasn't really gone down. Additionally, you've established that reinforcements aren't coming. However, have you considered the possibility that perhaps they're already here?

First, this may be an uphill battle if you're starting fresh at a new company. Nevertheless, it's still possible. Talk to everyone. Have coffee with them and learn what they do. Learn what they enjoy. If you already know people in your company, ask for networking suggestions. You will find that there are people that may be able to help you.

Consider the following scenarios:

Peyton has just been hired as the ScrumMaster for three teams, including yours. After evaluating user story throughput, he comes to you to brainstorm possible improvements. However, in the course of these discussions, he offers to roll user acceptance tests into the definition of done, a feature which you had previously determined was below your cut line.

Emanuela has worked in finance for over 10 years. She loves her work but has reached a career plateau. She hears about a position overseeing the creation of a beneficial new financial tool that her team will be working on. The only qualification she's missing is PHP programming. Through mutual coworkers, she hears about the quality reporting webpage you want to implement. She agrees to work overtime for the next two months to write the webpage for you so she can acquire the new skillset.

Over lunch Joon shares her excitement about finishing her new build tool that she has developed for her team. Unfortunately, since she's ahead of schedule, she laments that the team won't have time for a first run for another two weeks. Since build tools are not yet a reality on your team, you offer your product as a beta test environment. Joon happily agrees.

Stefan is a seasoned product owner newly assigned to your product. His last product suffered from numerous sustaining defects originating from customer calls so he's concerned with early testing efforts. After discussing various options, he agrees to hold all new features for two Sprints in order to give you time to implement the unit test framework.

Lindsay is an intern in a neighboring group. Their project is ahead of schedule and Lindsay is looking for a short project to fill the remainder of the internship before going back to school. With their manager's permission, Lindsay agrees to spend 30% of their time working on your project.

If you weren't looking for these opportunities, you would have missed some of your best free resources. Some companies actually have programs in place to help employees find each other. These systems are generally a win-win scenario as they get more skills, exposure or beta testers and you get a new, albeit temporary, resource. This is also where including the second-level manager on quality issues can come back to help you. They might have someone interested in the work you're doing, leading to another possible source of help.

5 Pick the Low Hanging Fruit

Right about now, you might be questioning if the point of this paper isn't to just wiggle out of doing your job. No; that couldn't be farther from the truth. At our core, we are quality professionals. This section is all about how to make a meaningful impression in a tough situation. And, it is possible.

By now, you should have a good understanding for the quality requirements as well as the other team factors. Now, evaluate your list. Look for opportunities. We've already identified DOD as one potential prospect. These are going to be the quality goals that have the highest priority and yet can be accomplished in the shortest time. You may hear these referred to as the "low-hanging fruit".

Take the specific example of defects. It is a common problem for projects to have absolutely no defect process. However, the defect process as a whole is full of quick-hitting possibilities. You could get the team started on an existing defect tracking tool. You can write up a quick-reference defect process cheatsheet. You can provide defect training including what fields are important for resolution. You can start sending out defect reports. You can start a bug scrub meeting.

In addition to the defect process suggestions above, the following table will give you a few ideas for some additional low-hanging fruit. Again, the focus is on where you can make a biggest impact in a minimal amount of time. These specific examples were selected because they could be implemented inside of a single week.

Process	**Description**	**Why**
Test Plan Overview	A presentation that outlines the quality goals	Different from a formal test plan, the idea is to level-set quality goals with the team.
Team Webpage	A centralized web-based location that the team can add documents	This is a quick and effective way to share information with the broader team.
Version Control Tools	Tracks the changes to the source code during development	Manage multiple developers and allow source code recovery.
Coding Standards	Set of guidelines to standardize the source code's programming style	It's often easy to get team buy-in. Additionally, it can increase readability and reduce defects.

Code reviews	Peers review each other's code before it is checked in to find mistakes early	Mistakes can be caught earlier in development. Additionally, enables software architecture cross-training.

Another technique you can try is to bound quality tasks. This may work well in cases where your quality vision does not directly correlate with management's requests. Reach an agreement about how long should be spent on a given task. Make as much progress towards this goal as possible in the allotted time. Then stop. Review it with management. If they want more than what was completed, propose a trade-off with other planned improvements.

Don't forget that quality processes have a lot in common across projects. You do not have to re-invent the wheel. In fact, you can often quickly establish some standardized tools and processes by adopting them, in whole or in part, from other teams. For example, if your team does webpage development for the company's widgets, can you talk with the gizmos web development team and establish a standard look and feel? The more creative you can get in your collaborations, the more opportunities you will start to see.

6 Convert Others

When you've tried all the normal methods for getting your quality improved it's time to fight (a little) dirty. In the previous sections, we talked about finding the willing converts. This section is about securing the unwilling ~~victims~~ volunteers. This will go a lot smoother if you have complete management buy-in on what you're about to do. If needed, you can re-visit the equations you developed above to help with justification. If you feel it would help, extrapolate the number of unproductive hours. Having developers working on QA might be a more palatable choice over developers doing busy work. Even if you don't have complete manager buy-in, a few of these proposed avenues will work even with a skeptical manager.

As a first step, consider converting your own developers. They are already part of your team and know the software well. This is going to be hard for a wide variety of reasons, but it's worth it because they have the programming skills you need. First, some engineers do not enjoy aspects of quality engineering. If you ask a developer to write a process document, they may consider it akin to chewing their own leg off. That brings us to another hurdle you'll face: the mindset. In some companies, there is the idea that quality engineers are somehow less skilled than developers. On one occasion, a seasoned quality engineer was asked by a developer, "You can program? Why aren't you a developer then?"

Start using the tools at your disposal. Education is an important aspect of your job. To address the mindset gap, consider listing out the technical skills needed to be a developer vs a quality engineer. In most cases, the underlying skillset is essentially the same. Scrum is an excellent example of this. In a true scrum team, you have only the product owner, the ScrumMaster and the team. There is no line between developers and testers. If your team has a strong Agile inclination, consider proposing that the team move to a 'true scrum' model and start erasing the line between the two roles. Offer to provide training to ease the transition.

At the end of the day, however, you may not be successful in fully converting your developers into quality assets. You can still get them to help in other ways, however. Find work where the developers have something to gain for themselves. Unit testing is a great example. They may not have come up with the idea but may take it over once suggested. It will identify early defects and the developers can manage it themselves, which may be more appealing than having the quality team own it. Last, have the developers complete tests on each other's work. If the quality team is unable to test in a timely manner, consider suggesting a developer.

"I won't have time to look at that user story until tomorrow. However, Reilley is really familiar with that interface. Have you considered having Reilley take a look at it?"

This has the added benefit of reducing friction in a group. Instead of being viewed as a blocker, you're offering them a solution to their current problem.

In your quest to convert the developers, don't forget about your other team members. ScrumMasters may be more sympathetic about achieving quality on a new product. Additionally, your goals may align with some of theirs. A technical writer with a few hours free may be able to help with reviewing early process documents. Even your manager isn't off limits here, especially if they were previously in a role with skillsets that align with your team's quality needs. It is an opportunity for them to keep their skills fresh and stay in closer contact with the team.

7 Conclusion

All of the techniques in this paper have been used in an industry setting over the last ten years. Communication and transparency will serve you well in any situation. Some of the ideas presented work better in particular environments. The manager's personality will influence your choices. Coworkers' temperaments come into play in many cases. You will need to be creative and flexible to reach your goals.

At the end of the day, you have a rough job ahead of you. Yet they hired you for a reason. You are in the unique position of solely owning the quality of the product. You can make a big difference even with your limited resources. Do not be afraid of being the first person to ask the question. Do not be afraid to hear 'no' repeatedly. And do not be afraid of making a few waves if it will result in a quality product.

References

Koopman, Phil. 2015. “Tester to Developer Ratio Should be 1:1 For A Typical Embedded Project", Better Embedded System SW Blog, entry posted February 2, http://betterembsw.blogspot.com/2015/02/tester-to-developer-ratio-should-be-11.html (accessed July 6, 2016)

McConnell, Steve. 2007-2008. “Software Development’s Low Hanging Fruit”, http://www.construx.com/LHF/ (accessed July 6, 2016)

Vroomans, Remko. 2014. “User Acceptance Test in Scrum”, Agile/Scrum blog, entry posted July 28, http://www.scrum.nl/prowareness/website/scrumblog.nsf/dx/user-acceptance-test-in-scrum (accessed July 6, 2016)

Whittaker, James. 2011. “The 10 Minute Test Plan”, Google Testing Blog, entry posted September 1, http://googletesting.blogspot.com/2011/09/10-minute-test-plan.html (accessed July 6, 2016)