Assuring Software Quality

David Bishop

How would you convince your management to allow your project team to incorporate quality reviews into the project life cycle?

There are many ways that someone could convince management to allow a project team to incorporate quality reviews into the project life cycle. One of the main points would be to present quality reviews not as more work to task the team with, but a way to ensure more work doesn't need to be done down the line. For example, if an issue is caught early in the work cycle it can be dealt with swiftly. However, if this bug is not caught early enough it could be near impossible sometimes to determine why the outcome of the program is giving incorrect information. A much less sever example of this happened on a program I wrote for work. When I was first coding it I determined colors for an html style would be indexed as 1, 2, 3 (the style's was set in an array and added dynamically). However, about halfway through the project I changed the indexing to 0, 1, 2. At first no issues really arose as the sample data I was testing seemed to be working. It wasn't until moving the code into a live environment that major issues started happening. The page wouldn't render at all and I could not for the life of me determine why. It wasn't until hours later that I realized that the code was simply calling index 3, which no longer existed, crashing everything else. This easily could have been avoided if my code had been reviewed earlier in the project and someone noticed this indexing issue. Time saving is not the only benefit that I would point out to management for adding quality reviews, another being overall alignment of business goals. As a lower-level developer, I rarely deal with the customer. Meaning that the product I am making tends to be to my own standards and preferences. However, with quality reviews added into the project life cycle we can gain a better understanding of what the customer really cares about and wants. As discussed by Flori Needle (citation 2), "customer feedback helps you learn about your audience on a deeper level". This means that for a company which implements quality reviews (especially customer reviews) will create a product that has improved customer satisfaction. Thus, creating a better product and one that sells better than a company that did not implement quality reviews.

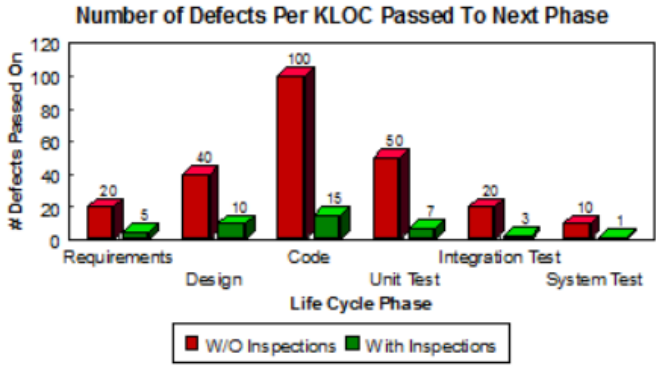
What kinds of quality review models (e.g., walkthrough, inspection, other) would you implement?

There are two quality review models that I would ensure are implemented into my own project. The first of which is formal inspection. This model is when a person or group examines the product to identify defects and other problems. Personally, this is a major review model to me and is something I consider a necessity in my own work. While I cannot delve too deep into what I've worked on (due to security reasons), having others perform a formal inspection can have major impacts on the quality of my code. On one of my projects as a new hire I didn't really understand the purpose of what I designed; I was given specifications but overall, how those linked together were far beyond me at the time. The first product I delivered was reviewed by one of the customers of the product. And while functional, it organizationally

didn't make any sense. His comments on the product had major effects on how I changed and reorganized the code. After this my product looked nearly unrecognizable and I received a wealth of positive feedback on the first event that utilized my product. The other type of review model I would implement would be peer review. While this is a very simplistic one seen in pretty much every project I've worked on, it is no less important. No matter how good of a programmer you are, there may always be issues or technique violations that may slip past you. Having a peer go through your work and comment on anything that seems out of place to them can be incredibly helpful. This isn't only to fix your own code, but also can help make your code more readable to everyone as you can add comments or rearrange code bits that made sense to you, but to an outsider are complete gibberish.

If you could do a software quality review at only one project life cycle phase, what phase would you pick…and why?

If I was required to only perform software quality reviews at a singular life cycle phase, I would personally choose the design phase. As discussed in this module, the longer you wait to perform code inspections the larger the number of defects may be found.




Based on my thought process and graph there are probably two questions that have formed, why not perform software quality reviews earlier and why not perform them at the coding stage? The first answer is simple, in the requirements phase not enough of the project is determined for a software quality review to really have much impact. As seen on the graph only 5 defects may be passed if you do inspection, but 20 will be passed if you don't do any inspections. This is the second smallest difference for the amount of defects passed along, meaning that performing it at this stage may be a waste of time if you cannot perform it at any other stage. Now for the second question, the answer is a little harder. Personally, I would prefer to do it at both the design and coding stage. However, since I was only allowed to choose one cycle phase I really dug deep. And while more defects may be passed during the code phase without inspection those defects may be simpler to identify or less destructive. The design phase is "a pivotal part of the production process (Stage 4), as developers will lean on it as their primary resource to build their code" (citation 1). Due to this an uncaught issue in this phase could majorly delay the timeframe of a project. For example, if a project is designed into

small chunks and handed to multiple teams major stalls could occur to a part of the project being put off to do later in the project. If a team needs that specific part of the project to move forward, they will be stalled however long until it is complete. This is currently happening in my office. We did not know that a specific item needed to be worked on earlier than it currently is scheduled to. Because of this we are stuck in a circular dependency where to determine how to write a piece of software we need the other teams requirements but the other team cannot create those requirements until we have designed our software. This so far has caused a stall on this project for two weeks and is looking like it will continue to be stalled.

What software quality metrics would you use to measure software maintainability, how would you use them, and where in the project life cycle would you use them?

There are a few software quality metrics that I would use to measure software maintainability. The first one would be coupling, simply this is a metric to determine inter-dependence between software components. Another would be Cyclomatic Complexity, which is the measure of how complex a software component's control flow is. Finally, I would utilize a technical debt metric, this can help determine the cost of future refactoring due to code shortcuts or deprecated libraries. I would utilize these during the coding and testing phases. During the coding phase looking at the coupling and technical debt metrics can have major impacts on how the code is written. For example, if a library or piece of software is determined to be used looking at the current rate at which it is maintained can help minimize possible technical debt. At my job we originally were utilizing a program called cesium and a wrapper service that was developed by external developers' angular cesium to implement it into a web app. However, some time into the project it was discovered that angular cesium had become deprecated due to a lack of development on it. This caused us to have to completely rewrite a large portion of our code to remove those dependencies or fall further into technical debt. The testing phase is when I would truly look at the Cyclomatic Complexity of the code, I would also utilize this during the coding phase to catch major complexities earlier. However, during the testing phase the final code has been implemented and is just being tested. This ensures that this metric is done at a time that has major impact. If someone were to attempt to determine complexity during only the coding phase, the metric may become outdated by the time it is determined rendering that work useless. However, if this metric is determined during the testing phase, we can ensure that it is completely accurate to the current code base.

Citations:


1. *The System Development Life Cycle: A phased approach to application security*. Security Intelligence. (2020, March 27). https://securityintelligence.com/the-system-development life-cycle-a-phased-approach-to-application-security/

2. Needle, F. (2023, August 24). *The benefits of customer feedback, according to experts*, HubSpot Blog. https://blog.hubspot.com/service/benefits-of-customer-feedback