Module 12 Assignment

Assuring Software Quality

Brian Loughran

8/12/2019

Questions

1. How would you convince your management to allow your project team to incorporate quality reviews into the project life cycle?
2. What kinds of quality review models (e.g., walkthrough, inspection, other) would you implement?
3. If you could do a software quality review at only one project life cycle phase, what phase would you pick…and why?
4. 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?

Answers

1. There are a multitude of reasons to adopt reviews in the project life cycle, and my strategy to convincing management would revolve around communicating the potential benefits of having regular, productive reviews.
   To convince management to adopt reviews in the project life cycle, I would make sure to explain some of the statistical findings by teams who successfully implemented reviews in their process. One finding was that effective use of quality reviews was found to shorten project schedules by 10-30%, which can be a significant cost-savings for the host company. Another statistic shows that reviews are effective in finding defects in a project, and depending on the depth of the review, can find up to 90% of project defects during review.
   I would also highlight the benefits of catching defects early in the project life cycle, and the pitfalls of letting a defect propagate further into the life cycle of a project. Numerous studies in industry have been conducted to show the relative cost to fix a defect increased exponentially with time. Therefore, finding and eliminating defect early in the project's life is crucial to maintaining project cost and schedule estimates. This means that if a defect costs just 1 monetary unit to fix during the requirements phase, it can cost 10 times that amount to fix during the coding phase, 50 times that amount in the test phase, and more than 300 times the amount once the project has been implemented.
   One benefit of reviews that I have found in my workplace but was not mentioned in lecture was the benefit of being able to reference a review later as a powerful piece of documentation. Often times, the review material contains much of the same information as a quality piece of documentation. The vast majority of the company time spent on the review is doing preparation before the review to create the review material. I would pitch that time spent preparing for the review to be multi-purpose as both preparing for the review and generating good documentation for our project.
2. The type of review that I would implement varies based on the size and safety requirements of the project. Different styles of reviews have different benefits. Tailoring the type of the review based on the project can help both to accentuate the effectiveness of the review and minimize the potential pitfalls that are associated with reviews.
   For example, round robin reviews are great for projects which are a bit less refined and smaller in scope. It may be more difficult to get buy-in from multiple reviewers to thoroughly test a project with a very large scope; some reviewers may balk at the amount of work required.

However, having multiple people reviewing a project can be very helpful to refine edge cases and situations that you may not have expected as a developer.

An informal walkthrough can be helpful for informational purposes (teaching someone how to use a product), getting buy-in from potential users, and for communicating the state of the project to stakeholders. Informal reviews often have the lowest overhead, so performing an informal review can reduce the potential cost of doing a review in the first place.

Formal inspection is a useful tool for high-fidelity reviews. These can be useful for very large projects or projects which require a high amount of safety/security, such as government projects and projects which handle sensitive personal information. Formal inspections are more rigorous and structured than the other methods, which have the benefit of finding a higher percentage of defects (up to 90% for the formal inspection, where the other review types typically approach 60%). One potential pitfall of a formal inspection is the increased time that needs to go into the review, so as a company you should consider weighing the cost against the benefit of the review and make sure you are maximizing the benefit.

3. If I could do a software quality review at only one point in the project I would implement it after the test phase of the project. This review could serve as a good capstone to the project and provide effective documentation both to future users of the product and key stakeholders. After the testing phase is a great time to review because you have a near-complete project and can check that all of the functionality is working as intended. Conditional action items can be created at this time if there are any defects found during the review and those action items would have to be reviewed and filed as complete before the review is marked as complete.

   One potential pitfall to only having a review after the test phase is the relative high cost of fixing defects found during the review. Finding a defect in an earlier phase, such as the design or code phase would have significant cost savings in comparison to finding after the test phase. But, if there can be only one review, the benefits of seeing a complete, working product outweigh the pitfalls of more expensive debugs.

4. There are many potential metrics that can be used to measure software maintainability. Unfortunately, many of these metrics are not measured in absolute terms (like you can measure temperature, weight, or size) but are more contusive to measurement in relative terms. For example, a piece of software with ample comments, good documentation, etc. will likely be more maintainable than another piece of software. However, it would be more difficult to put an absolute number on the maintainability of a project. This is where we can use some different metrics to describe the maintainability of software.

   One such metric is coupling. Coupling measures the level of interdependence between components of a piece of software. Components that interact are coupled to a degree, depending on how dependant the components are on each other. Loose coupling generally begets higher software maintainability, whereas tighter coupling would have a negative impact on the maintainability of the software.

   Another metric that can be used to measure software maintainability is cohesion. Cohesion is a measure of how single-purpose a software component is. Higher cohesion would generally increase software maintainability, whereas lower cohesion could impact software maintainability negatively.