Daniel Marzari 450

CIS 332

Jim Worth

1/29/20

Software Quality

The difficulty with software quality is that it is subjective to the specific requirements of any given project. No two applications perform the exact same goal and even if their means are similar, they may have very different results or formats. In order to measure quality, these subjective variables are often defined and weighted according to the intuition of managers without any methodical view of these variables. Traditionally these variables have been selected almost impressionistically by the opinion of the developers and stakeholders and weighted by relevance to the final product (Hajdu-măcelaru 31). These attempts at defining software quality do a poor job at truly defining the quality of the product and often center around the end product rather than the way to it which results in a functional but not high-quality product. This approach does not accurately define quality and does not orient software toward future development even if it does organize the development cycle (Hajdu-măcelaru 34). It is therefore necessary that objective quantification of software quality and performance be applied with the aim of getting a quality product.

Fortunately, software quality does not need to be defined in abstract and subjective terms. Software engineers try to design a product that will accomplish a goal and not do anything unintended or erroneous (Nogo 65). It is possible to measure this in terms of the nature of the software's structure and methods - that is how well it accomplishes the task relative to other

approaches, and how efficacious it is in accomplishing the task. For example, in the present project of creating a web interface to connect with alumni, software quality could be defined not by the end result that the user sees and rather on the database structure, type, and methods used, the API calls, and frontend interface that displays and interacts with the user. The questions used stop asking “does this happen?” and starts asking, “how does this happen?”. This understanding of software significantly confines the vagueness of defining quality to a set of logical evaluation criteria. One approach to this is breaking down the fundamental attributes of a project and prioritizing the functionality of each - not so that any piece can be done poorly, but so that if and when these components come into conflict, it is easy to decide which must be less efficient for the sake of the overall software (Hajdu-măcelaru 32-33). Software quality, therefore, is essentially ensuring that the design, structure, and methods of the software minimize drawbacks, limitations, and errors respectively (Nogo 65).

Before diving into what automated software testing can do, it is important to note what it can’t or should be used to do. First, it should not be used as a primary debugging tool. All software that undergoes automated testing should be fully functional insofar as the developers realize. Testing is designed to expose flaws and bugs that are not immediately visible to developers and often pertain not to functional errors which pertain to the output of a function, but memory leaks, processing complexity, quantity of error messages, data structures, and algorithms used (Mshelia 21). Automated testing can and should be used to measure various quality components of source code. Each of the aforementioned metrics have tests associated with them that can help measure the quality of software (Mshelia 21). For example, in the alumni database, since it is unlikely that the database will contain more than 10,000 records in the next

10 years, a scalability metric can be considered lightly. On the other hand, it may need to be high performance (the database gets a lot of requests and updates), in which case a performance or efficiency metric should be weighed heavily. The measurements of these software testing are often viewed in terms of qualities such as efficiency, maintainability, reliability, scalability, and stability which allow for quick evaluation of software quality.

To sum, the best steps to take to develop high quality software are (1) defining the key attributes of the system, (2) identifying the key metrics essential to those attributes in terms of the desired scope of the software, and (3) using various quality metric tools to compile that data (Mshelia 24-25). Each attribute must be evaluated for design, structure, and methods identifying drawbacks, limitations that may arise upon their use and implementation (Nogo 65). The attributes must then be weighed in terms of the future scope of the software and chosen accordingly so that there would not be any roadblocks in future development. Key metrics should be used to keep progress on track that don't rely on customer satisfaction or end product functionality alone, but mainly on the quality of the product delivered. Not all these metrics need to be things like efficiency, and scalability, they could be usability (user-friendly), simplicity (easy to look at), or comprehensibility (easy to understand). As for these later variables, however, automated testing cannot be used. While some tools are language-specific, many multi-language analyzers have come out including MASU, PREST, SQUALE, and SSQSA which measure a variety of metrics using the source code (Mshelia 34-35). With these three simple steps, software quality along the development cycle can be maintained apart from the superficial definitions all too often used when looking only at the end product.

Works Cited

Hajdu-măcelaru, Mara, and Ioana Zelina. "An Adaptable Software Quality Model." *Creative Mathematics & Informatics*, vol. 27, no. 1, Jan. 2018, pp. 31–35. *EBSCOhost*, search.ebscohost.com/login.aspx?direct=true&AuthType=ip,shib&db=asn&AN=128896715&site=eds-live.

Mshelia, Yusuf Umaru. "MECOT: A Software Quality Metrics Collection Tool." *Journal of Systems Integration (1804-2724)*, vol. 10, no. 1, Jan. 2019, pp. 21–35. *EBSCOhost*, doi:10.20470/jsi.v10i1.360.

Nogo, Srđan, and Zoran Škrkar. "An Approach to the Design Software Automation Testing Environment." *International Journal on Information Technologies & Security*, vol. 10, no. 3, July 2018, pp. 65–74. *EBSCOhost*, search.ebscohost.com/login.aspx?direct=true&AuthType=ip,shib&db=iih&AN=131930359&site=eds-live.