Six Sigma and Software Engineering and Reliability

written by Manoj Khanna | April 18, 2003

I recently finished reading the book “What is six-sigma?” by Peter Pande, and Larry Holpp. In terms of Software Engineering, Six Sigma is much more than a specific analysis of software reliability. It is a quality improvement framework, and mindset focused on the measurement of process variation as the culprit for lack of quality. I want to point out that the term “six sigma” when used in conjunction with software reliability, has little or nothing to do with statistics, with distributions, with their moments, etc. It is a buzzword and will remain a buzzword until such a time as it is defined in statistically correct ways.

The real Sense for Six Sigma□□

Six Sigma as the name implies stands for six standard deviations from the mean. Sigma is a statistical measure of variability around the average. The concept of Six Sigma comes from reliability engineering prediction of system or component failure probabilities. For example, the wear out time of a component may be normally distributed – that is meant – standard deviation. So, we want a component having a very small of failure before its design life. If, we set this at one sigma from the mean, we get ~80% reliability, and 2 sigmas gives us ~95%, and 3 sigmas ~99%, and so on. Six Sigma gives us ~99.9997% reliability – near perfect; or, in other ways 3.4 defects per million.

Six Sigma and Software Reliability.

□□In terms of software engineering, however, it is not so quite clear cut as compared to mechanical or electronic components. Also in case of software reliability, we don’t have very good predictive models, failure models, etc. As

somebody suggested, that one approach to this could be to predict faults remaining as a function of faults found in earlier phases. In general terms, for software reliability, Six Sigma would mean that the software process will find ~99.9997% of all the faults before the software is put into service.

What do we need to do?

We need to adjust the design life accordingly. In common terms, the design life of shrink wrapped software is ten seconds before we open the package, and for the custom software ten seconds after the check clears.

In the language of Motorola official release:

> *”Motorola wants to be free of errors and defects 99.9997% of the time in all that it does. That means no more than 3.4 defects per million units.”- ‘Electronic Business’, October 16, 1989*

Statistical Tools – Improved Software Quality

Use of Statistical Tools to Improve Software Quality and some points to remember regarding this:

- Today, the complexity and size of software has grown substantially, along with the size and complexity of the silicon processors, perhaps exceeding Moore’s Law (a doubling of processing power every 18 months).
- The business risk of developing very large software systems has spurred the development of a very large shrink wrapped software industry, primarily because of the failure of many very large complex systems.
- Software factories, of which the primary case would be Microsoft, flourish by delivering very large, internally complex products, at prices consumers can afford to bear, exclusively by delivering extremely

large volumes of like products. The only technique that has proven effective for quality assurance is using thousands of volunteer quality inspectors (beta testers) to report the errors prior to the final release of the product. Because the cost of manufacturing beta copies is so low, it is far out weighed by the economic benefit the company receives from this type of testing process.

- Hence, can we ever assume that the software development industry will ever achieve on standardized uniform measure of software quality, given that to be relevant, the definition of a software standard must be reached between the consumer of the software are and the producer of the software ? I would conjecture, probably no.
- The reason for this is due to the nature of software. An algorithm may be provably correct but may be implemented in an inefficient manner. (A possible defect). It might be physically damaged in the duplication of a disk (a manufacturing problem), which might manifest itself by the consumer being unable to install and use the product. The cause of the problem, may remain the inefficient implementation of the algorithm, but it manifests itself in so many potential ways, it will be in all likelihood, impossible for the consumer to identify the defect, and unless a defect can be quantitatively measured it will be impossible to detect.
- At the very core of the problem, the inefficient algorithm might be the work of one designer or developer, being unaware that more efficient mechanisms might exist, or it may be the result of a specification error, or perhaps the algorithm subroutine was purchased from an outside supplier, who provided poor instructions regarding it’s limitations.
- Statistical tools can be used to analyze overall system quality, such as a transaction failure. These tools are severely limited in the applicability to an

individual software developer because the development task is typically□□ to design and write single software modules, as opposed to a large scale software□□ use.

- We keep learning more and developing new insights, so things will change, □□most probably through the use of better software partitioning and packaging□□ technology.

Conclusion□

In the end, the people at large, the **users** does not understand why a concept that is worthy and meaningful in the hardware and manufacturing domain ***does not*** apply to software. Consequently, the **users** might be mislead and ill-served because they are led to believe “six sigma” software is somehow comparable to “six sigma” hardware. Is it?? Does it??

[I am convinced that others who have read the authoritative literature on six sigma and have attended the appropriate training could talk more intelligently about this technology.]