An Analysis of Six Sigma for Software Development

Dan Shaeffer
Shaeffer Institute for Systems Architecture
Dallas–Fort Worth, Texas
contact@danielshaeffer.com

March 14, 2016

Abstract

During the post-World War II industrial era, statistical quality control and process improvement methodologies revolutionized manufacturing. Through the evolution of Motorola's Six Sigma framework, quantitative defect reduction methodologies sought to establish standardized tolerances of 3.4 defects per million opportunities. This monograph analyzes the theoretical and operational translation of Six Sigma into the software development domain. We dissect the two primary methodologies—DMAIC (for existing process management) and DMADV/DFSS (for initial greenfield process architecture)—mapping their discrete stages directly against the Software Development Life Cycle (SDLC). We review empirical industrial case studies, including multi-billion dollar cost savings across defense contractors, while critically examining standard domain objections regarding non-linear algorithmic complexity and non-repeatable creative processes.

Keywords: Six Sigma, DMAIC, DMADV, DFSS, Software Process Engineering, Defect Density, Quality Metrics.

1 Historical Origins and the Japanese Quality Movement

During World War II, Japan underwent an industrial boom, resulting in large increases in manufacturing output. In order to support the war effort, the output of heavy manufacturing firms increased an astonishing 252% [13]. After the war, the Japanese economy was in ruins; however, the large manufacturing sector and its supporting infrastructure were largely intact. Factories that made machines and other goods for facilitating destruction upon enemy nations were repurposed to make goods of a more peaceful nature, such as automobiles, sewing machines, and telescopes [17]. However, the immaturity of the Japanese manufacturing sector led to the output of inferior products that soon became the object of international derision. Although normally a liability, it was this very immaturity, combined with nuances of the Japanese culture regarding non-resistance to the adoption of innovative technology and management techniques allowed for a so-called quality movement to take place [10].

The Japanese quality movement came in the form of expatriates such as W. Edwards Deming and Joseph M. Juran bestowing their innovative quality management methodologies upon Japanese industry. These methodologies most notably included statistical analysis for the purpose of evaluating process capability, and process improvement methods such as Plan-Do-Check-Act [12]. These two methodologies are two distinct disciplines, but are also complementary in that the results of statistical analyses could be used to improve processes, and the very act of process improvement could help

prioritize which areas undergo statistical analysis. It is this synergy that forms the implicit goal of what came to be known as Six Sigma [3].

2 The Six Sigma Quality Framework

Six Sigma represents framework-based quality improvement methodologies that strive to achieve near-perfect defect elimination through use of data-driven process improvements [5]. Through the use of statistical analysis, Six Sigma provides irrefutable and quantifiable data regarding the overall performance of a process, as represented by defects per millions of opportunities. The goal is to achieve, at most, 3.4 defects per million opportunities, which is a defect quantity that represents six standard deviations between the mean and the overall number of chances for a defect [5]. Although statistical analysis using normal curves traces its roots to the 18th century, it was Motorola who trademarked the term Six Sigma in order to brand their process quality measurement methodologies. As the company very famously experienced a reported $16 billion in savings from the use of Six Sigma over a period of 15 years, and as other organizations realized substantial cost savings, the methodology quickly spread into extensive multi-domain usage [19].

3 Application to Software Engineering Processes

Despite its roots in manufacturing and other more traditional process-based industries, Six Sigma has evolved so that its principles may be applied to software development processes. This applicability stems from its generic approach, in that its most popular frameworks and implementation methodologies are not exclusive to any particular industry or domain. Although the complex nature of software renders the concept of achieving only 3.4 defects per million opportunities very unrealistic, the fundamental goal of Six Sigma remains the same, which is to introduce quality into the development processes in order to proactively impact quality, versus only utilizing reactive, late-cycle methods such as quality assurance activities. Regardless of the domain, quality is improved via the identification and standardization of best practices that are demonstrated to be repeatable [11].

The Six Sigma framework specifies that some sort of appropriate quantification of these processes must take place in order to prove their superiority. As such, quantitative measurements in the form of statistical analysis are used to identify variability and failure points, as well as the performance of processes. Once it is established that there is clear improvement in the process and that the process is repeatable, then it is to be considered a best practice. By designing or improving processes with an increased focus on quantifiable quality measurements and establishing best practices, the organization will produce higher quality outputs and reduce its reliance upon less cost-effective reactive quality control methodologies, among other benefits [14].

The Six Sigma body of knowledge identifies two major process improvement methodologies for use in improving quality in software development projects [6]. These techniques are known as Define-Measure-Analyze-Improve-Control (DMAIC) and Define-Measure-Analyze-Design-Verify (DMADV), which is also interchangeably referred to as Design for Six Sigma (DFSS). Although not exclusive to the software development domain, these techniques have been identified as being wholly applicable to the software development domain. Furthermore, as the techniques are both based on Deming's storied Plan-Do-Check-Act (PDCA) Cycle and, thus, share multiple common traits, they have distinct applications and are mapped to different software project management models [1].

4 The DMAIC Methodology in Software Maintenance

The most common approach to implementing Six Sigma is DMAIC [5]. DMAIC is a data-driven improvement cycle that focuses on improving and standardizing existing processes and designs. This is the core tool used for software development, as it allows for the most quantifiable assessment of process improvement due to its reliance on benchmarking and assessing the variances observed after implementation. This superb visibility, coupled with the constraint that it is to be used for existing processes, which are typically easier to assess than new processes, contributes to its popularity. DMAIC does not map directly to the Software Development Life Cycle (SDLC); instead, its steps map to the Process Management Life Cycle [7]. However, all phases of the SDLC may benefit from its application, as DMAIC is used when there are specific processes that need improvement no matter the phase of development.

4.1 Define

In the Define step, the business objective at hand is clearly enumerated without ambiguity. Along with the objective, the project's stakeholders, critical process outputs (known as CTQs – items Critical to Quality), timelines, and scope are resolutely defined. As pertaining to software development, the current quality concerns would be defined within the context of the software project; for example, excessive code defect rates, schedule slips, or misunderstandings of requirements. Only when the concerns are identified can the process of determining mitigating steps begin.

4.2 Measure

After the issues or concerns are enumerated and documented, the process moves on to gaining insight regarding the current state of processes, as well as developing indicators of progress or success. In the Measure step, the objective is to identify metrics, as well as obtain statistics and measurements regarding the quality issues identified in the Define step. These measurements, per the context of the defined metrics, provide the baselines that will be compared against during benchmarking activities with the goal of obtaining irrefutable quantification of whether or not the implementation of quality improvement techniques are demonstrating effectiveness. In other words, how and when measurements are to be taken, and the current state of the enumerated quality concerns are determined, as well as the type of performance desired. As pertaining to software development and the above example, once the scope of the quality improvement efforts has been determined, metrics are agreed upon and measurements that yield baselines are obtained regarding the frequency and timing of requirements revisions, the number of lines of code per day written, and the number of tasks implementers are required to complete that are not relevant to implementation efforts.

4.3 Analyze

Once the metrics have been enumerated and baseline measurements have been captured, the efforts become more analytical in nature. In the Analysis step, an introspective look is taken at the quality issues determined to be within the scope of the quality improvement efforts in order to enumerate root causes. The analysis of these posited root causes is to narrow their scope in order to identify top root causes upon which to focus. Further analysis of the narrowed field of enumerated root causes is to yield the actual, quantified effects upon output in order to determine the criticality of the process and the possible impact improvements will have. The objective is to analyze the data in the previous step in order to discover patterns in the issues with the entire project in mind in order to identify the

most effective improvement approaches and their potential placement. These patterns may exist not only within the context of defect categorization, but also by personnel and even patterns regarding the time of day the issue is encountered. In the domain of software development, analysis may yield the discovery that one development team constantly achieves objectives on time while others constantly miss their deadlines, but that the defect rate for the latter team is much lower. Rather than a shortsighted approach that would not take this interesting discovery into consideration, this data would help craft the most beneficial and cost-effective approaches in mitigating the identified issues.

4.4 Improve

The elimination of the root causes identified and quantified in the previous step is the objective of the Improve step. In this step, the solutions to the issues within the scope of the improvement efforts are identified and implemented. The use of an iterative approach, such as Deming's Plan-Do-Check-Act (PDCA) Cycle, is often applied in order to find, test, and implement fixes and other improvements [2]. Once the proposed solutions are in place in the tentative process, new baselines are taken in the same areas in which they were taken before with the purpose of benchmarking; that is, determining whether or not the improvements yielded the desired results. Once the piloted improvements are determined to be satisfactory, they are deployed.

4.5 Control

The last step in the DMAIC methodology is Control. In this step, the processes are monitored, with particular attention paid to the recently improved areas, in order to make certain that the changes are repeatable and continuing to have the desired effect. Using the metrics identified in previous steps, benchmarking takes place in order to determine that the current measurements are still satisfactory, as well as to identify further areas that need to undergo improvement. In the domain of software development, this would occur via the examination of the defect rates, schedule slippage rates, and requirements miscues in order to determine that actual improvement has occurred and continues at a desirable level.

5 The DMADV / DFSS Greenfield Framework

The other major process management methodology is DMADV, also referred to as DFSS. DMADV differs from DMAIC in that it is focused on using statistical tools in order to promote the initial implementation of quality processes prior to the implementation of any processes [18]. In other words, DMADV is a framework used to create a quality process where no process already exists. In the domain of software development, this methodology is functionally equivalent to the SDLC; as such, DMADV may be folded into the SDLC with ease, rather than existing as an adjunct or substitute framework.

5.1 Define

The Define step entails the identification of the issues most critical to stakeholders, such as the project goals and outcomes. This is similar to the Define step in DMAIC; however, as the quality issues are unknown at this point due to the lack of an implemented process, the definition takes place in a different manner. Rather than using currently observed issues to guide the definition, items such as feedback, historical data, and brainstorming sessions, as well as experiences observed

in similar processes, may be used to define the objectives of the quality efforts [16]. In a software project, this is the step in which the project is scoped and the planning takes place; as such, this step is closely mapped to the Planning phase of the SDLC.

5.2 Measure

The next step is the Measure step. Again, in common with DMAIC, the metrics defined in this stage are used to collect data in order to determine how best to proceed in creating a valid design. However, rather than using metrics regarding current processes, metrics are utilized that determine the variance between business objectives and what is currently being experienced in order to later gauge how helpful the new process is in meeting the objectives stated in the Define step. Additionally, metrics are defined that guide the outcome of future implementation work, such as performance criteria. Although these types of metrics cannot yet result in measurements that help guide the design process, these will still provide a baseline by which the outcomes will be benchmarked. In a software project, this is the step that requirements analysis takes place in the form of identifying what is truly specified by the requirements and devising metrics to assist in validation activities.

5.3 Analyze

In the Analyze step, also in common with the similarly named step in DMAIC, the measurements obtained in the previous step are analyzed in order to determine patterns and provide insight into how to effectively meet the demands of the processes. This is accomplished via the testing of the initial outputs of the processes in order to establish a baseline that will be used to benchmark future results. Once this occurs, the best design options begin to become more apparent, as this analysis leads to the identification of methodologies that lead to the achievement of the required results. In a software development project, this step, along with the previous step, are mapped to the Analysis phase of the SDLC; as such, it is in this step that design options are analyzed in order to determine which option best meets the requirements of the project.

5.4 Design

After all of the design options and initial outputs are analyzed, the next step is Design. In this step, the final design is generated that will best meet the requirements outlined in the initial step. In a software project, it is in this step that the detailed design occurs, implementation takes place, and testing frameworks used for verification and validation efforts are developed.

5.5 Verify

The final step is Verify. This step refers to efforts used to determine that the results of the new process meet the requirements outlined in the Define step. The outputs are benchmarked against previously obtained baselines in order to validate the design. This step is iterative in nature, in that processes will undergo continuous improvement based on feedback or measurement data. In a software project, the Verify step is directly mapped to the Maintenance phase of the SDLC, up to and including its iterative nature. It is in this step that testing and the resultant bug fixes occur. The metrics identified previously are used to verify that the maintenance efforts are in line with the identified requirements and objectives. Once the verification and validation efforts prove to be successful, the project output is implemented or delivered.

6 Documented Industrial Case Studies

The results of the implementation of Six Sigma in the software development domain have been well-documented. As a result, there is a large body of case studies that provides an analysis of the impact Six Sigma has had on software development efforts. This analysis establishes the effectiveness of Six Sigma in the software domain as, on a large scale, Six Sigma has shown the ability to produce significant cost savings.

For instance, Raytheon implemented Six Sigma frameworks in their software process development efforts. Due to the process improvements and their impacts on quality and efficiency, the company realized $1.8 billion in financial gains, all attributed to their Six Sigma for Software Development campaign [8]. Another example indicates that Lockheed Martin experienced similar levels of cost savings after utilizing Six Sigma frameworks in software development efforts [15].

These financial benefits are achieved because companies that implement Six Sigma in software development efforts have reported improvements to efficiency and overall process quality that have resulted in decreased defect densities, decreased defect resolution time, and improved maintainability [5]. These improvements eventually translate into cost savings that may be reinvested within the project in order to improve quality in other areas. Thus, process improvements beget more process improvements, which increases quality even further, which may result in additional financial gain.

7 Critical Counterarguments and Rebuttal

Despite the potential benefits and documented successes, the use of Six Sigma methodologies in software development is not without criticism. Critics contend that Six Sigma is not well-suited for software development because software processes are complex and not inherently repeatable, as they do not comprise fixed entities or processes like that of manufacturing processes, nor is software measurable in a meaningful, scientific manner comparable with the "pure" engineering disciplines [9]. Furthermore, there is the accusation that Six Sigma may only achieve results because its champions only tout the results when the initial process was of such poor quality, that any process improvement methodology would have yielded results [20].

These positions have some validity in that the software code itself does not undergo improvements based on any particular Six Sigma methodology, and also that in the absence of any actual process, any methodology may have produced better results. However, these assertions miss the point in that Six Sigma is not applied to the code itself, but rather the development process as a whole. If the process used to develop software undergoes improvement, then it stands to reason that the outputs would be of higher quality, because Six Sigma frameworks pertaining to software development ensure that processes undergo improvement or designed with a focus on quality in order to produce outputs that completely conform to the stated design and are produced correctly. This is, for all intents and purposes, the very definition of software quality [4].

Also, if the implementation of Six Sigma frameworks yields quality processes when no processes existed before, then it has met its objective, as this is the very point of using a quality framework to begin with: to either take an introspective look at processes and determine manners in which they can be improved, or to create quality processes with the aid of an established framework. If this occurs on any cost-effective scale, then its usage is justified.

8 Conclusion

In summary, Six Sigma is a proven methodology for implementing quality processes that have the potential to produce high quality outputs in any domain. Organizations that have implemented Six Sigma have experienced financial gain due to improvements to process efficiencies and output quality. Even if substantial financial gains are not realized, the introspective and quantitative nature of Six Sigma results in an intense analysis of business goals and processes, and has consistently shown to be a cost-effective endeavor.

Having mature, repeatable processes generated as a result of utilizing a solid quality framework is superior to relying on unrepeatable processes that result in inferior outputs. Therefore, organizations that are examining quality improvement methodologies would be well-served by consulting the Six Sigma body of knowledge in order to determine an appropriate approach for implementing improved processes.

References

[1] "DMAIC or DMADV: Which one to use?" Kaizen Institute (2015). Web. 22 Feb. 2016.

[2] "Plan-Do-Check-Act." *Wikipedia, The Free Encyclopedia*. Wikimedia Foundation, n.d. Web. 22 Feb. 2016.

[3] "Six Sigma." *Wikipedia, The Free Encyclopedia*. Wikimedia Foundation, n.d. Web. 22 Feb. 2016.

[4] "Software Quality." *Wikipedia, The Free Encyclopedia*. Wikimedia Foundation, n.d. Web. 22 Feb. 2016.

[5] "What is Six Sigma?" *iSixSigma*, n.d. Web. 22 Feb. 2016.

[6] Aggarwal, Maneesh. "Six Sigma Meets Software Development." *iSixSigma*, n.d. Web. 27 Feb. 2016.

[7] Ali, Abid, Ramaswamy, and Pandya. "Software Six Sigma – DMAIC vs. DMADV in Software." n.d. Web. 24 Feb. 2016.

[8] Al-Qutaish, Rafa E., and Khalid T. Al-Sarayreh. "Applying Six Sigma Concepts to Software Engineering: Myths and Facts." *7th WSEAS Int. Conf. on Software Engineering, Parallel and Distributed Systems*, 2008.

[9] Binder, R. V. "Can a Manufacturing Quality Model Work for Software?" *IEEE Software* 14, no. 5 (1997): 101–105.

[10] Cusumano, Michael A. "Manufacturing Innovation: Lessons from the Japanese Auto Industry." *MIT Sloan Management Review* (1988).

[11] Dey, Sarit. "A Lean Six Sigma Approach to Improving Efficiency." 2015. Web. 24 Feb. 2016.

[12] Folaron, Jim. "The Evolution of Six Sigma." *American Society for Quality (ASQ)*, 2003.

[13] Gluck, Carol. "The Useful War." *Showa: The Japan of Hirohito*. New York & London: W. W. Norton & Company, 1992, pp. 54–55.

[14] Harrington, James H. "Six Sigma for Internet Application Development." *American Society for Quality (ASQ)*, 2001.

[15] Heinz, Lauren. "Using Six Sigma in Software Development." *News @ SEI* (Software Engineering Institute), 2004, p. 43.

[16] Logan, Marvin. "Measure: Step 2 in the DMADV Process to Improve Your Supply Chain." Mar. 2014.

[17] Pile, Kenneth. *The Making of Modern Japan*. Lexington, MA: D.C. Heath, 1996, p. 242.

[18] Roseke, Bernie. "How to Implement Six Sigma in Your Organization." Mar. 2014.

[19] Tetteh, Edem G., and Benedict M. Uzochukwu. *Lean Six Sigma Approaches in Manufacturing, Services, and Production*. Business Science Reference, 2015, p. 264.

[20] Viswanathan, V. "Six Sigma Not Necessarily Beneficial to Software." Web. 2016.