Six Sigma Approach in Software Quality Improvement

Cvetan Redzic
Motorola, Inc.
Cvetan.Redzic@motorola.com

Jongmoon Baik
Information and Communications University
jbaik@icu.ac.kr

Abstract

In this paper, we present the six sigma DMAIC approach which is used for software quality improvement. The goal was to identify and establish tactical changes that substantially increase the software quality of all software products over the next 2 years. We analyzed the data and based on the analysis expert decisions were made to determine which new technologies (tools, methods, standards, training) should be implemented and institutionalized in order to reach our goals. To measure the improvement from Six Sigma process changes we calculated our process capability baselines based on tactical changes, and we tracked and evaluated ongoing software product quality on a regular basis against these baselines to ensure that the software product quality goals were being achieved as planned.

1. Introduction

In this paper, we will present the Six Sigma DMAIC (Define, Measure, Analyze, Improve, and Control) approach that was used for software quality improvement. In the Define phase, we created the team charter, identified the current process, and identified “quick wins”. In the Measurement phase we measured the process and established its baseline based on historical phase containment effectiveness (PCE) data. Orthogonal Defect Classification (ODC) analysis, defect per million opportunities (DPMO) for historical six sigma baseline, and software reliability were used as quality predictors. In the Analyze phase, based upon the analyzed information, expert based decisions were taken to determine which new tools and process improvements should be implemented and institutionalized in order to reach the set goal. To account for risk and uncertainty, the Monte-Carlo simulation was used. These efforts led to new process capabilities and new baselines. Also, a cost benefit analysis was conducted. In the Improve phase, the pilots’ of the recommended tools and/or tactical changes were scheduled, and process improvements were made. In the Control phase, the digitize approach was used, i.e. responsibilities for progress tracking were assigned, processes and tools defined, metrics and reports (an early sign or warning so that timely actions can be taken) identified, and reviews scheduled.

Six Sigma (SS) is invented in Motorola as a strong methodology for quality improvement, specifically in: New Product Design, Training, Processes, Tools, and Tracking. It also dedicates the best resources to key projects that focus upon Margin Dollar Growth objectives through the following corporate level initiatives: - New Product Introduction Engineering; - Cost of Poor Quality; - Procurement
- Sector goals, to be consistent with Motorola corporate goals must include: - Customer Loyalty; - Revenue/Market Share; - Operating Earnings

Software Quality Six Sigma Project

The Software Quality SS project supports a customer loyalty, market share, and earnings goal. Its purpose is to identify and establish tactical changes in the software product development life cycle (DLC) that substantially increase the software quality of all software products for the next two years. The project goal was set at a 90% (50% in the first year) reduction in the number of remaining high severity defects over two years from the established baseline release. Incremental or modest continuous process improvement couldn’t be used here (short time frame) rather a quantum leap of substantial process improvement was necessary. As such, it was prudent and desirable that quantitative management methods such as six sigma and technology change management (TCM), as defined in SEI CMM Level 5, be used to reach this challenging goal.

In the next sections we describe the phases of the DMAIC process – Define, Measure, Analyze, Improve, Control [8], and conclude with a summary of our experiences and results.
2. DEFINE PHASE

In the Define phase, our goals were to understand the problem, identify input and output variables, form the team, define goals and milestones of the project, and understand the project’s merit. The most important outputs of this phase were that the project charter was established and a multiple division team was formed. Also “Quick Wins” team was formed to focus on improvements that could be made with low effort but with good return on investment (“low hanging fruits”).

The topics discussed in this phase are: project charter; SIPOC (Supplier, Input, Process, Output, and Customer); voice of customer (Kano analysis); voice of business; as is process map; and “quick wins”.

We discuss each of these in turn.

2.1. Project Charter

A project charter indicates an overall commitment between the Champion, Black Belt, and project team. It serves as the communication vehicle. The Software Quality project charter includes: a business case (project purpose), opportunity statement, goal statement, project scope, project plan, project benefits, and team selected.

Business Case:
The Software Quality SS project purpose was to identify and establish tactical changes that substantially increase the software quality of products released to its customers over the next 2 years. Complementing this tactical SS project on software quality is a strategic SS project for SEI Level 5 processes [12].

Opportunity Statement:
Implementation and usage of the new technologies (software tools, standards, methodologies, and training) will decrease the number of injected defects as well as increase the defect detection in the software products which will decrease the number of remaining defects in the delivered system products.

Goal Statement:
Reduce the predicted remaining severity P1/P2 defects by 50%, prorated for products released during the 12 months following a Q1 ’03 baseline while minimizing impact on schedule and cost.

Further increase the predicted remaining severity P1/P2 defects to a total of 90% prorated for products released over the subsequent 12 month.

Establish 2 Quick Wins per division for immediate improvements in 2003

Project Scope:
Process: Development Life Cycle
Start Point: M-Gate 11 (Lockdown System Requirements)
quick wins) but also contributed in brainstorming and discussion sections at the team level weekly meetings. **Project Benefits:** In addition to improved customer satisfaction/loyalty, the financial benefits of reducing escaped defects was estimated and showed significant results, and was set as additional goal of X amount million dollars savings. These savings then can be channeled into new Creation development activities that deliver more value to our customers and increase Motorola’s feature competitiveness.

The division uses software reliability growth model (Goel-Okumoto exponential model), [4], that considers the defects discovered in system test versus test execution time and determines the number of defects that can be expected in a release. By focusing on discovered high severity defects, and using maximum likelihood estimate (MLE), or least squares estimate (LSE), the exponential fitting curve could be established, during system test, for the total high severity defects predicted in the release. Comparing this prediction to the actual P1/P2s discovered in system test results in identifying the remaining predicted defects. The correlation between the predicted numbers of remaining defects, from several releases, versus the actual number of field problem found was made, which helped to increase accuracy in prediction of customer unique defects by new release.

### 2.2. SIPOC

In SIPOC (Supplier, Input, Process, Output, and Customer) diagram, Table 1, the process to be improved has been identified:
Name the process: Development Life Cycle (DLC)
State the starting & ending points: M Gate 11; M Gate2 Define outputs: SRE Remaining Defects
Determine customers: End-customers
List major steps of the process: SFRAS – system requirements, FRAS – feature requirements, IDS – interface design, HLD – high level design, LLD low level design, CODE, PTX – process test, FTX – feature test, IST integration and system test, and NPI new product introduction
Define inputs: phase containment data, reliability data, CRUD (customer reported unique defects) data
Determine suppliers: SFRAS, Development, IST, and NPI

<table>
<thead>
<tr>
<th>Suppliers Providers of the required resources</th>
<th>Inputs Resources required by the process</th>
<th>Process Top level description of activity</th>
<th>Outputs Deliverables from the process</th>
<th>Customers Receivers of the deliverables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sys Req</td>
<td>Req Docs</td>
<td>SW DLC</td>
<td>SW product/</td>
<td>End-Users</td>
</tr>
<tr>
<td>Developers</td>
<td>Design Template,</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### 2.3. Voice of the Customer

The analysis of customer satisfaction surveys, which have been performed for the past several years, point to software quality as a significant key driver to customer loyalty. We decomposed our customer loyalty data into a Kano analysis [13] to reflect our customer expectations.

Kano analysis, shown in Figure 1, classifies customer needs in software product as: - Basic Expectations (Must be) – These are the elements which customer expects to be met. If they are met customer will not be additionally satisfied, but if not met customer will be very dissatisfied. – Satisfier, features delivered to customers. Customer satisfaction is positively correlated with the number of the features in the product. – Delighters, attractive features are those which customer didn’t ask and pay for but which fulfill customer’s latent needs.

Software quality belongs to the customers’ basic expectations in Kano analysis, i.e. software quality is a “Must be” – customer does not like and does not expect failures in the software system. The delivered software features are satisfiers. Small enhancement and small features which customer didn’t ask and pay for but help product to be better or easier to use are delighters.

Our analysis, in this paper, addresses only half of the Kano model, by focusing on customer satisfaction and CRUD (Customer Reported Unique Defect) reduction, i.e. the portion below the Quality line.

#### 2.4. Voice of the Business

Software quality goals (thresholds) from the 2003 Balanced Score Card were set for the number of CRUD by division. By summer 2003, data indicated that the goals were nearly met and would be exceeded before end of the year.
2.5. As Is Process

The process under consideration is standard software DLC (Development Life Cycle) described by V-model shown below.

![Figure 2. Software Development Life Cycle – V Model](image)

2.6. Quick Wins

Quick wins team examined best/good processes in each division. The team also overviewed standard software development processes by division, and identified opportunities, “low hanging fruit”, for easy process improvement from which return on investment could be high. They then made recommendations, some of which were accepted: - process non-optional execution; - proper usage of Flexilint tool; - designed template redesign; - code inspection effectiveness improvement.

By establishing the team, defining the project goals, and examining process under consideration for improvement, we were able to move to next phase in DMAIC process - Measure phase.

3. MEASURE PHASE

In the Measure phase, we documented the existing process in order to determine what we would measure to diagnose the problem in quantitative terms, requested data definition and data collection by each division, and evaluated if the measurement system was adequate, established a baseline for each division’s current process. We then displayed data graphically so that findings could be highlighted and used for the improvement effort. To achieve this we: evaluated what to measure; developed data collection plan and created operational definition; conducted source of variation study; calculated sigma level; set reliability goals; calculated As Is process capability baselines.

3.1. What To Measure?

Input, Process, and Output Indicators. As an input indicator, phase containment data was chosen as measurement of the consistency of the inputs to the process. For a process indicator, we used fault density as the efficiency measure. As an output performance indicator and effectiveness measure, the number of customer reported unique defects, CRUD, was used.

3.2. Data Collection and Operational Definitions

Standard operational definitions were used for phase containment, software reliability and CRUD data.

- Monthly phase containment data; the data showed the effectiveness of division in all phases of the development life cycle – the effectiveness in detecting injected defects and the remaining defects passed on to subsequent development phases.
- Weekly software reliability data; the data includes system test execution time, and defects found during that time.
- Weekly CRUD data; the data presents the number of customer reported unique defects against specific release.

3.3. Sources of Variation

We used the COQUALMO (COstructive QUALity Model) estimation tool, [1], [16], which is one of COCOMO II suite models, to estimate the number of defects injected and detected per delta KLOC by phase versus our actual values.

Figure 3 shows the big difference between actual defect detection versus COQUALMO estimates, in CODE phase.

![Figure 3. Actual defect removal vs. COQUALMO Estimate](image)
3.4. Sigma Level

The Sigma level, [8], was calculated for all three set of data: for high severity faults, from phase containment data set, for high severity reliability defects from system test, and for CRUD, shown in Figure 4.

The Sigma level is measured as DPMO (Defects Per Million Opportunities, DPMO = 10^6 * D/(N*O), where D means total number of defects, N is number of units of product or service, O is number of opportunities per unit of product or service for a customer defect to occur, and M is million. For example if we have in release: D = 398 high severity defects, and the number of delta LOC is 212,000, then DPMO = 10^6 * 398/212,000 = 1877 which is equal to sigma level of 4.4.

![Figure 4. Sigma level](image)

From Figure 4, we see that the sigma level for the subject division is very stable from release to release, and implies that development process is very mature and stable. Thus, to make a shift in this process to reduce escaped defects by 50/90%, not continuous process improvement, but very strong quantum leap improvement is required.

3.5. Software Reliability Goal Setting

Software product reliability, [4][6][9][12][10][15], during system test, was used as the measurement of software quality. Reliability goals could be set as: percentage of defect found versus predicted, MTTD (mean time to defect), or as the number of remaining defects (threshold), shown in Figure 5. As we saw, the reliability goal was set as the number of remaining defects.

![Figure 5. Software reliability goals](image)

3.7. As Is Process Baslines

We used historical data from several previous releases, to build process capability baselines for division. The baselines were established for defect injection process and for defect removal process.

Figure 6 shows the baseline for cumulative defect injection process. Defect could be injected till to the Code phase. The defect mean values, from previous releases, by phase were used to establish the central line, and three sigma intervals were used for lower and upper bounds.

![Figure 6. Defect injection process baseline](image)

3.7. As Is Process Baslines

In this phase we documented the process by establishing the process capability baselines, calculating the sigma levels, set the reliability goal etc. In the next phase we will provide the analysis to uncover potential areas for improvement in order to reach the project specified goals.

4. ANALYZE PHASE

In the Analyze phase, we investigated potential sources of variation by examining the relationship between input and output variables in order to uncover potential areas for improvement by conducting: -
4.1. Pareto Analysis

We used the Pareto analysis to analyze the escaped defect data. The escaped defects are defined as customer reported unique defect (CRUD). Figure 7 shows CRUD Pareto analysis from an orthogonal defect classification (ODC) perspective [6], [17]. This helped to identify which defect types are prevalent among CRUD. ODC defect types are connected with a specific software development phase, and show systemic process defects. By using ODC in EDA, we can concentrate on those phases which have the systemic problems.

EDA analysis showed that the highest number of CRUD is Algorithm defects. Algorithm defects are associated with low level design (LLD) phase. (Algorithm defects affect the task and can be fixed by implementing an algorithm or local data structure. They do not require a design change.)

When we analyzed ODC signatures, shown in Figure 8, ODC defect type distributions over a whole software release life cycle, we found that the distribution of algorithm defects peak in CODE phase instead of in LLD.

It can also be observed that there are too many injected assignment defects (value(s) assigned incorrectly or not assigned at all, indicates a few lines of code – the initialization of control blocks or data structure), and checking defects (program logic that has failed to properly validate data before they are used – loop conditions, etc.) which are associated with CODE phase. Therefore their distributions should peak in CODE phase; it was too many injected as already mentioned in Define phase with comparison with COQUALMO estimate.

4.2. Correlation between Release Software Size and Defect Injected

Next, we analyzed the correlation between high severity defects from three different sources and released software size: high severity release faults versus release size (shown in Figure 9), reliability defects versus size, and high-severity CRUD versus size. The 90% confidence interval was used to show how good correlation between size and defects is.

All three analyses showed strong correlation between high severity defects and size, and for each analysis the correlation coefficient was over 0.93. The correlation between software size and defects injection shows that our development process has been very stable, from release to release.

4.3. Software Reliability as a Measure of Software Quality from Customer Point of View

The project applied the Goel-Okumoto exponential model [4], for reliability measurement, show in Figure 10. Data gathered on a weekly basis from system test is
test execution time (TET) and defects found. A reliability workbook was created to simplify reliability calculations and charts presentation for quality assessment. From the input data, the parameters $n$ (total number of defects in system) and $r$ (defect discovery rate) are estimated by using maximum likelihood estimate, MLE, or least squares estimate, LSE.

New, reliability exit goals were set as 90% reduction in defects remaining from reliability estimate over a next two year, with 50% for 2004, as noticed previously.

Here we finished analysis phase and now we will describe the improvement phase in which we will identify, test and implement potential solutions in order to reach our goals.

5. IMPROVE PHASE

In the Improve phase the team gathered and statistically analyzed ideas for improvement, the relationships between injected and detected defects and release size were identified, the solution effectiveness was verified, and cost/benefit was documented. Also, new process capabilities were recalculated, as well as the associated sigma level.

To identify and select potential solutions for process improvement we: - built phase-based defect removal model; - generated improvement ideas (Cause and effect diagram); - evaluated & selected solutions (Cause and effect matrix); - established new process capability baselines; - built plan for tactical changes.

The SS software quality improvement team determined the tactical changes (new technologies) in developing software, were needed that provided benefits that either “reduced defect injection” or “improved defect detection”.

Given the goal to reduce escaped defects, and knowing that “testing in quality” during system test leads to missed schedules and too many escaped defects (see Voice of the Customer), the need exists to introduce fewer software defects or at least find them sooner - enabling fewer defects being passed to system test.

Until now, the team focused on gaining greater levels of understanding of the deficiencies affecting the current process through root cause analysis. The reduction or elimination of these root causes now became the basis for the solutions which the team generated.

5.1. Phase-Based Defect Removal Model

A defect removal model, [5], was developed and proved as a very useful for quality planning and quality management. It summarized the relationship among three metrics: defects escaped from previous phase, defect injection, and defect removal effectiveness.

The ideal scenario for improvement was to decrease defect injection by phase while at the same time increase removal effectiveness, i.e. defect detection.
improvement. The defect removal model is a quality management tool, and it can be applied to a stable division’s development process, meaning that the defect removal pattern by phase is within control bounds from release to release if no changes in the process are made.

5.2. Cause and Effect Diagram

The team used cause and effect diagram (the fishbone chart), shown in Figure 12, for idea generation on how to improve the defect removal process or increase prevention of escaped defects.

![Cause and Effect Diagram](image)

**Figure 12. Cause and Effect Diagram**

### 5.3. Cause and Effect Matrix

The team then used, a modified, cause and effect matrix to evaluate relationships and percentage of process improvement -- of key input, new suggested technologies against output -- for defect injection prevention and defect removal improvements by phase. Recommended technologies (methods, tools, standards, training) by teams:

- **Team 1**: - Non-optional process execution; - Proper usage of Flexilint; - Design template improvement; - Code inspection effectiveness
- **Team 2**: - Use case
- **Team 3**: - Joint system testing; - Fault prediction model (SLIM); - Klocwork; - Agile
- **Team 4**: - Reliability application throughout all test organizations; - Test automation; - Focus testing (problem areas); - Error leg testing
- **Team 5**: - NPI test; - Post fixes, system impact evaluation

Using the defect removal model, these expert opinion percentages of process improvement were recalculated, shown in Table 2, to get new “could have been” reduced value of escaped defects.

<table>
<thead>
<tr>
<th>Environment</th>
<th>Tools</th>
<th>Defect Tracking</th>
<th>Environment</th>
<th>Tools</th>
<th>Defect Tracking</th>
<th>Environment</th>
<th>Tools</th>
<th>Defect Tracking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agile</td>
<td>FMEA</td>
<td>GSD</td>
<td>Agile</td>
<td>FMEA</td>
<td>GSD</td>
<td>Agile</td>
<td>FMEA</td>
<td>GSD</td>
</tr>
<tr>
<td>Agile</td>
<td>FMEA</td>
<td>GSD</td>
<td>Agile</td>
<td>FMEA</td>
<td>GSD</td>
<td>Agile</td>
<td>FMEA</td>
<td>GSD</td>
</tr>
<tr>
<td>Agile</td>
<td>FMEA</td>
<td>GSD</td>
<td>Agile</td>
<td>FMEA</td>
<td>GSD</td>
<td>Agile</td>
<td>FMEA</td>
<td>GSD</td>
</tr>
</tbody>
</table>

![Defect Tracking](image)

**Figure 13. Cause and Effect Matrix**

<table>
<thead>
<tr>
<th>Methods</th>
<th>Training</th>
<th>Tools</th>
<th>Defect Tracking</th>
<th>Environment</th>
<th>Tools</th>
<th>Defect Tracking</th>
<th>Environment</th>
<th>Tools</th>
<th>Defect Tracking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agile</td>
<td>FMEA</td>
<td>GSD</td>
<td>Agile</td>
<td>Agile</td>
<td>FMEA</td>
<td>GSD</td>
<td>Agile</td>
<td>FMEA</td>
<td>GSD</td>
</tr>
<tr>
<td>Agile</td>
<td>FMEA</td>
<td>GSD</td>
<td>Agile</td>
<td>Agile</td>
<td>FMEA</td>
<td>GSD</td>
<td>Agile</td>
<td>FMEA</td>
<td>GSD</td>
</tr>
<tr>
<td>Agile</td>
<td>FMEA</td>
<td>GSD</td>
<td>Agile</td>
<td>Agile</td>
<td>FMEA</td>
<td>GSD</td>
<td>Agile</td>
<td>FMEA</td>
<td>GSD</td>
</tr>
</tbody>
</table>

![Defect Tracking](image)

**Figure 12. Cause and Effect Diagram**

The team didn’t have the time to run experiment testing for evaluation of these new technologies. So, the team included uncertainty in these projected process improvement percentages by running Monte Carlo simulation.

On this basis the potential improvements from implementation of new technologies/techniques for all divisions are calculated.

### 5.4. New Process Baseline for Defect Injection and Defect Removal Processes

New process baselines were recalculated based on the new PCE data, which were derived from the “could have been” process. These new baselines will be now used as process control bounds for releases under development for defect injection control, shown in Figure 13.

![New Baseline](image)

**Figure 13. New control bounds for defect injection process**

### 5.5. Tactical Changes Plan

When Would These Benefits Occur? As these tactical changes were identified in Q403, all divisions were in various phases of development for their next releases.

### Table 2. As Is and Could-Be DRM models

| Phase | Defect Escaped Defects per KLOC | Defect Defect at exit of Phase Removal Removal Phase |
|-------|---------------------------------|-----------------------------------|-----------------------------------|
| REQ   | 2.25                            | 2.2                               | 86%                              | 1.9                              | 0.32 |
| IDS   | 0.32                            | 0.75                              | 100%                             | 0.42                             | 0.49 |
| HLD   | 0.75                            | 0.75                              | 50%                              | 0.42                             | 0.49 |
| LLD   | 0.75                            | 0.75                              | 50%                              | 0.42                             | 0.49 |
| CODE  | 0.75                            | 0.75                              | 50%                              | 0.42                             | 0.49 |
| PT    | 1.11                            | 1.11                              | 100%                             | 0.97                             | 0.16 |
| FT    | 1.11                            | 1.11                              | 100%                             | 0.97                             | 0.16 |
| ST    | 1.11                            | 1.11                              | 100%                             | 0.97                             | 0.16 |
| P-R   | 1.11                            | 1.11                              | 100%                             | 0.97                             | 0.16 |

![Table 2](image)

**Table 2. As Is and Could-Be DRM models**
As such, only those phases yet to begin could utilize the change. An effectiveness matrix was developed for each division. This showed by division which change would be used and when (release) and time to achieve the full (100%) effect of the benefit.

Pilot Plan

Pilot plans were created for every new technology deployment and every organization to show time frame and implementation plan.

Now we’re moving to the last phase – Control. To have strong follow-up on technology implementation and quality improvement, results are reviewed regularly, during monthly team reviews, and reported to Sector and each Divisional General Manager. Corrective actions are developed as needed.

6. CONTROL PHASE

In the Control phase the team is working to ensure that the gains are sustained for the long term, new processes are documented, and training is provided. These efforts include: - responsibilities; - performance reviews; - metrics for software quality tracking; - monthly quality reviews

6.1. Responsibilities

To ensure implementing the solution and making the solution part of the normal business practice, responsibilities are dedicated to key team members for: - execute, evaluate, and standardize the solution; - data and metrics generation

6.2. Performance Reviews

Reviews for project success evaluation and tracking are scheduled on a monthly basis. The purpose is to review the performance of the process using the metrics, identify the gaps, and prioritize necessary improvements.

6.3. Metrics to be internally used for software quality tracking

The following metrics were suggested for tracking software progress through:

Development life cycle: - Fault prediction model (SLIM); - Inspection effectiveness metrics; - ODC metrics; - Defect removal model

System test: - Software reliability

Field: - ODC and Escaped defect analysis, - Field reliability

6.4. Software Quality Monthly Reviews

Metrics were chosen for monthly Software Quality reviews:

SLIM – for fault prediction and tracking, shown in Figure 14.

Defect Injection and Removal Model, shown in Figure 15.

Software reliability – for software reliability goal evaluation, shown in Figure 5, 10 and 11.

CRUD vs. Goals, shown in Figure 15.

7. Conclusions

Software is a very complex product. To measure software quality we can not use just one metric. We have to slice and dice through software process and product, to evaluate process capability and...
performance, and product quality, in different phases of development life cycle before software is ready for release.

Here we demonstrated six sigma (SS) usage to achieve software quality improvement. We used a structured methodology for addressing this quality improvement project. Representatives from across divisions collaborated to bring the best of the best. We used a common set of tools and methods, and sharing of best practices across all the divisions. By identifying critical measures that are necessary to evaluate the quality of our software, developing a measurement plan to effectively collect data, and establishing a baseline performance, we were able to start institutionalizing a culture where quality can be predicted and drive the development process to deliver the required quality.

We showed how technology change can be used in order to help people be more effective and efficient in their jobs. Using this, process and product quantum leap quality improvement was made in order to reach our set goal.

8. References


[8] Motorola 4X4 Six Sigma Black Belt Training Material


[17] Saraiya N., Redzic C., Premal M.; “Quantifying ODC Defect Type Signature Baseline – Case study”, Motorola, S3S Symposium, Chicago, 2004