Six Sigma applied to improve Integration Build process

Fábio S. Nagamine

Motorola Industrial LTDA – SP-340, km 128.7, Jaguariúna/SP – Brazil

Fabio.Nagamine@motorola.com

Abstract. *Integration Build is a Software Configuration Management process intended to ensure that a stable up to date baseline is available for further development or for client release. Six Sigma is a methodology for process improvement that uses measurements and statistical analysis. This article presents a case study showing how Six Sigma was applied to reduce human effort, improve quality and provide a quantitative analysis on the Integration Build process of a Motorola Software Component.*

Resumo. *Build de Integração é um processo de Gerência de Configuração de Software destinado a assegurar que um baseline estável e atualizado esteja disponível para desenvolvimento adicional ou para liberação ao cliente. Seis Sigma é uma metodologia para melhoria de processo que usa medições e análises estatísticas. Este artigo apresenta um estudo de caso demonstrando a aplicação de Seis Sigma para redução de esforço humano, melhoria de qualidade e obtenção de análise quantitativa do processo de Build de Integração de um Component de Software da Motorola.*

1. Introduction

Reducing human effort and improving quality are desirable achievements in any activity. Many authors (See [Dart 1990], [Brown 1991], [Berczuk et al. 2002]) encourage the application of automated tools to reduce effort and guarantee more stable results in Configuration Management activities. The benefits from switching process or adding a new tool, however, are rarely measured in quantitative terms such as number of defects, human effort hours or financial savings.

This article presents a case study showing how Six Sigma methodology was applied to a Motorola Software Component Integration Build process to reduce the human effort, guarantee more stable outputs and provide measure of gain in quantitative terms.

2. The Component Integration Build

Integration Build is the process of joining and building changes made by many collaborators on pieces of a software artifact. Building all changes together is necessary to ensure that the basic functionalities do not break [Berczuk et al. 2002].

Motorola Software organization implements Integration Build in many scales. Component teams implement the Integration Build process to gather and validate changes from individual Developer Engineers into a Component Build, that in its turn is

a piece on a Super Component Integration Build, that in its turn is a piece on a Product Software Integration Build (Figure 1).

For component level the Integration Build includes, besides basic functions of joining and building the code, a Smoke Test [Berczuk et al. 2002] and a set of quality verifications over process and product deliverables in order to attend organizational requirements. Most of the tasks performed, including the quality verifications, are assisted or fully executed by software tools.

The case study presented in this article targets a component from Motorola Software called in this work as "Case Component". The names and values for this component were replaced to avoid confidential information disclosure, without change the result of the study.

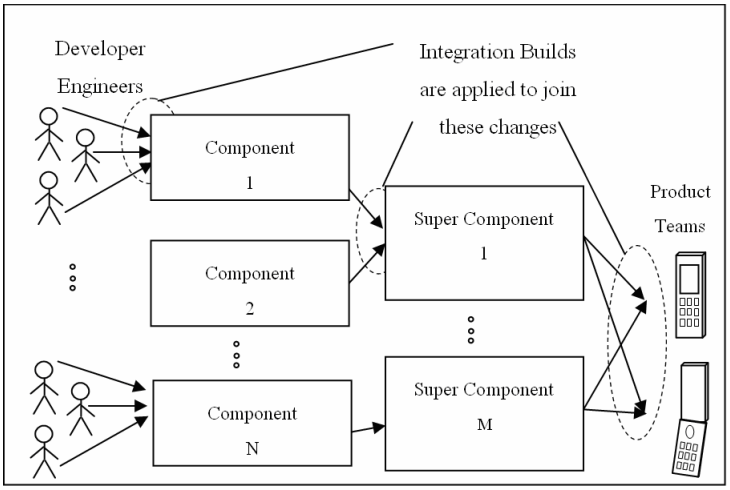

Figure 1. Integration Builds on Motorola Software

2.1. Issues on Case Component Integration Build

The "Pearson's correlation" is the degree of linear relationship between two variables measured as a real value *r* ranging from -1 to +1, where positive correlation indicates that both variables increase or decrease together, whereas negative correlation indicates that as one variable increases, so the other decreases, and vice versa. [November 2005]. Historical data indicates $r = 0.77$ between number of Integration Builds and total human effort spent by Case Component, i.e. to execute the Integration Build activities the team spends a determined amount of human effort that is positively correlated to scale of work.

The Case Component shows a positive trend in average number of Integration Builds executed along the last four years (Figure 2) to attend client's demand. The amount of effort required by the team increased reasonably together ($r = 0.77$) with the number of Integration Builds released. The number of human resources available however did not increase in the same rate due to organizational restrictions (Figure 2). Therefore an increase in efficiency is necessary in order to attend a growing demand for Integration Builds with no growth in the human effort required to perform these builds and no loss of quality.

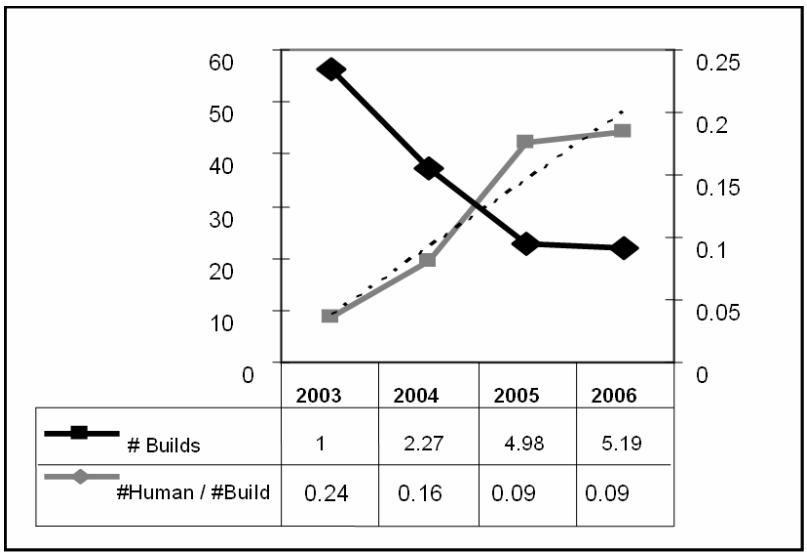

Figure 2. Builds from 2003 to 2006: Ascending line is demand of Integration Builds. Descending line is average rate of human resources available per Integration Build required.

3. Application of Six Sigma in Case Component

Six Sigma is a methodology for improvement or development of processes, services or products by means of measurements, statistical analysis and control of defects. Its application on software was discussed in previous works (see [Biehl 2004], [Bernardes 2006] for instance). According to Bernardes, "Six Sigma might play an important role in software processes regardless of the maturity[1] of the team".

Six Sigma defines more than one problem-solving framework, as business situations may vary on their nature. The most commonly used Six Sigma frameworks are DMAIC and DMADV/DFSS. For Case Component issue, DMAIC was chosen, since it is the most suitable for existing processes [Simon2007].

3.1. DMAIC applied to Case Component

DMAIC stands for Define, Measure, Analyze, Improve and Control. The next sections detail the application of these phases on Case Component.

3.2. Define phase

Define phase exists to formally define the DMAIC project, its elements, context, importance and purpose. For Case Component, the more relevant output from Define phase was the definition of what is the defect: any Build Integration whose effort is higher than the 1 hour[2] required by the sponsor of the project.

[1] Maturity, in this case, refers to CMM/CMMI maturity levels

[2] Real values were replaced to avoid confidential information disclosure.

3.3. Measure phase

In Measure phase, the most expected output is the Baseline, a historical measurement of indicators chosen to determine the performance before changes made by the DMAIC project. These indicators are also measured to assess the progress during Improve phase and to ensure improvement is kept after Control phase at the end of the project. The lead indicator is used to track the project goal whereas other indicators are used to track variables that affect the lead indicator.

For Case Component, the lead indicator is "staff hour spent per Integration Build released". The baseline for this indicator, an average of 12 months collection span, is 2.09 staff hours with control limits[3] [1.19, 2.99] (Figure 3). To attend the client's requirement of 1 hour, the lead indicator should be reduced by 52%.

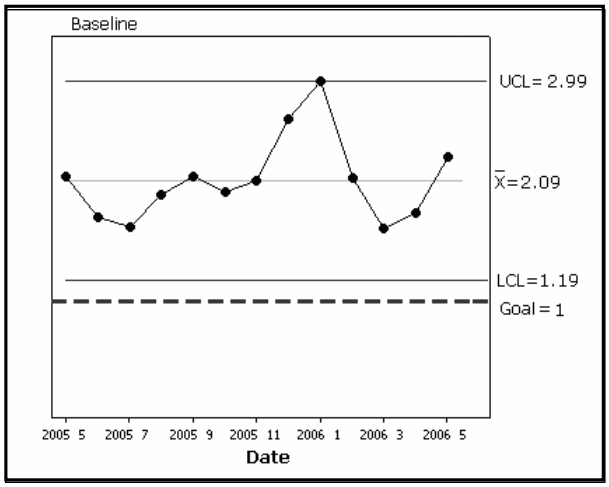

Figure 3. Staff hours spent per Integration Build released. Solid lines define the Baseline and dashed line is the Goal.

3.4. Analysis phase

The main activities of Analysis phase are raising and validating root causes of the problem being attacked. In Case Component, nine root causes were validated (Table 1). Eight root causes were validated from 22 potential causes raised in brainstorm session, and one additional root cause was suggested and validated during the Improve phase. Validation is made using statistical tools such as Hypothesis Test and ANOVA[4]. In this DMAIC project specifically, one root cause[5] was first noticed after Analysis phase, but this is not the regular case.

[3] LCL (Lower Control Limit) is mean minus three standard deviations; UCL (Upper Control Limit) is mean plus three standard deviations. In Normal distributions, 99% values are expected to be within control limits interval [LCL, UCL].

[4] Stands for "ANalysis Of Variance". Further study on ANOVA can be found at [Smith 1998].

[5] Root Cause #1 (Parallel Builds are complex) were raised during simulations run on Improve phase.

3.5. Improve phase

Improve phase consists of selecting and implementing solutions to eliminate or reduce the impact of root causes discovered on Analysis phase. In Case Component, each of the nine root causes elected in Analysis phase had potential solutions raised on brainstorm sections. A decision table[6] was used to elect the four higher priority solutions for implementation (Table 1).

Table 1. Decision table to prioritize potential solutions.

#	Root Cause	Scope of change	Influence in lead indicator	Cost of change	Priority
1	Parallel Builds are complex	1	1	2	2
2	Late adds/removals	1	2	1	2
3	Form[7] not validated	1	3	1	3
4	Form written manually	1	3	1	3
5	Form not tested	1	2	3	6
6	Too many forms	3	1	3	9
7	Cloned forms	3	1	3	9
8	Too many build lines	3	1	3	9
9	Slow Environment	2	2	3	12

3.5.1. Solution #1

The first solution implemented attacks root causes 2, 3 and 4 in (Table 1) and is related to the input of the process.

Developer Engineers submit changes by means of request forms containing an unequivocal description of elements to be integrated and built, besides change tracking identification and quality information. Originally, all forms content were gathered and placed in the form manually by Developer Engineers, resulting in unstable input for Integration Build, and impact of 28% builds affected by issues on forms[8].

To address this issue, Developer Engineers were encouraged to use a software tool that automatically gathers and fills in information in the form. This tool was developed by Case Component team in association with Developer Engineers representative in order to make it attractive for its users. Besides fulfilling the development team needs, this tool provided a powerful control mechanism, since its

[6] Input values are: Scope (1 = internal process, 2 = environment, 3 = external decision); Influence (1 = presents positive correlation, 2 = statistical analysis not conclusive, 3 = presents no correlation); Cost of change (1 = internal process change, 2 = other teams support, 3 = external process change). Output Priority is (Scope x Influence x Cost).

[7] Changes are submitted using forms requesting the inclusion of a change in an Integration Build.

[8] Measurement is average of Jan-Dec 2005. Since distribution is not Normal, no control limit is given.

functionality and its output are stored in a database kept by Case Component team, enabling control over what actions and verifications shall be performed prior form submission.

3.5.2. Solution #2

Second solution implemented attacks the first root cause in (Table 1). ANOVA shows that efficiency of the Build Engineer decreases as more Build Integrations are done in the same day (Figure 4).

Build Engineers were supposed to be saving time by running in parallel, but they were not. Integration Build procedure had been designed for single runs. Therefore, to run in parallel, the Build Engineer had to manually switch from one build to another, keeping in mind particularities of many tools that had to be run in several builds in varying states. Besides higher means, the group of 3-4 parallel builds presented greater variances, a considerable issue in Six Sigma projects since as broader is the variance as more unpredictable is the behavior of the process.

To solve this issue, the existing tools were joined in a single module, adding mechanisms to isolate the environment used by each build and an agent to control the build procedure flow. The controls added by this agent reduced the effort by eliminating the need of Build Engineers to start and monitor many individual tools, and reducing the need to switch from one build to another.

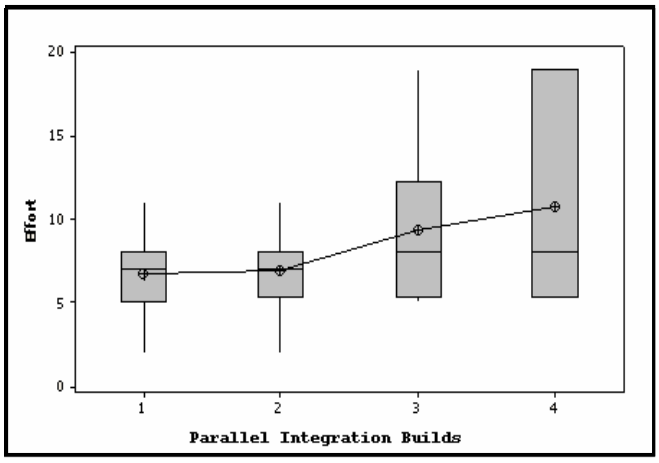

Figure 4. ANOVA of Effort per build vs. Number of parallel builds. Metric obtained from effort and integration builds daily recorded by Build Engineers during one month.

3.5.1. Results of Solutions Implemented

Five months after the implementation of first solution, the proportion of builds affected by issues on forms fell 14% on average, with 95% confidence interval (8.64%, 19.54%) estimated by Hypothesis Test. The input for Integration Builds became standardized and more likely for automation. Second solution was released two months later and, in three months, number of staff hours per Integration Build reduced to an average of 0.92 hours with control limits [0.63, 1.2], achieving the goal and finishing the Improve phase (Figure 5).

3.6. Control phase

The purpose of last phase of DMAIC is to maintain the improvement after project close out. To ensure Case Component Integration Build process would not degrade to higher effort builds, three actions were taken: the implemented solutions were incorporated as part of Case Component organization process; the new tools were given to Tools team who is now responsible for supporting the application; and metrics raised on Measure phase started to be monthly tracked by means of Control Charts[9] to trigger adjusts always when necessary.

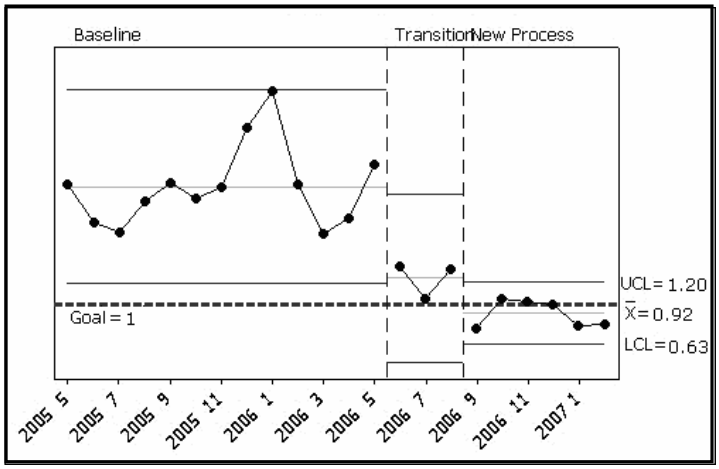

Figure 5. Staff hours spent per Integration Builds – Control chart before and after project implementation shows indicator shifted from 2.09 to 0.92 hours, and control limits reduced from [2.99, 1.19] to [1.2, 0.63].

4. Conclusion

Results show that Six Sigma methodology and DMAIC framework successfully worked to reduce human effort and variance in an Integration Build process. Improvement actions performed on Case Component Integration Build had total cost of 580 staff hours. Solutions implemented saved a total of 1100 staff hours after four months.

Besides financial savings, the reduction on variance and introduced controls resulted in a more predictable process, what reduces the number of "emergencies" and provides more accurate estimates.

[9] In Control Charts, average and control limits of indicator are represented as lines along the temporal axis. Whenever an element (average or individual value) falls above Upper Control Limit or Below Lower Control limit, the element must be analyzed in order to check whether it is result of a special cause of variation or a process change: the former might indicate a new unexpected agent in the process, the later might mean that or the process is not being followed or the process must change to accomplish environment/requirements changes.

References

Berczuk, S. and Appleton, B. (2002) "Software Configuration Management Patterns: Effective Teamwork, Practical Integration", Addison-Wesley. Boston, MA, USA. p. 97-127.

Bernardes, L. (2006) "Six Sigma support to software process improvement", Presentation made on SEPG LA 2006. Sao Paulo, SP, Brazil. Online at: http://www.esi.es/SEPGLA/sepglaPresentations_por.php#Pre29 [Accessed March 19, 2007].

Biehl, R.E. (2004) "Six Sigma for Software", IEEE Software, vol.21, no.2, p. 68-70.

Brown, A., Dart, S., Feiler, P., Wallnau, K. (1991) "The State of Automated Configuration Management", Technical report, Software Engineering Institute, Carnegie Mellon University.

Dart, S.A. (1991) "Concepts in Configuration Management Systems", the 3rd International Workshop on Software Configuration Management. ACM Press, New York, NY, USA, p. 1-18.

November, M.T. (2005) "Visualizing correlation", Journal of Computational and Statistical Graphics, vol. 14, no. 1, p. 1–19.

Simon, K. (2007) "DMAIC versus DMADV", Online at: http://www.isixsigma.com/library/content/c001211a.asp [Accessed March 20, 2007]

Smith, T.A and Hickling, F. (1998) "One-Way Analysis Of Variance Under Heteroscedastic Errors", NTTS'98 – New Techniques & Technologies for Statistics. Sorrento, Italy.