Repository Mining and Six Sigma for Process Improvement

Michael VanHilst
Dept. of Computer Science & Eng.
Florida Atlantic University
Boca Raton, Florida
1 954 661-1473
vanhilst@fau.edu

Pankaj K. Garg
Zee Source
1684 Nightingale Avenue, Suite 201
Sunnyvale, California
1 408 373-4027
garg@zeesource.net

Christopher Lo
Dept. of Computer Science & Eng.
Florida Atlantic University
Boca Raton, Florida
1 561 346-4749
chrishlo@yahoo.com

ABSTRACT

In this paper, we propose to apply artifact mining in a global development environment to support measurement based process management and improvement, such as SEI/CMMI's GQ(I)M and Six Sigma's DMAIC. CMM has its origins in managing large software projects for the government and emphasizes achieving expected outcomes. In GQM, organizational goals are identified, appropriate questions with corresponding measurements are then defined and collected. Six Sigma has its origins in manufacturing and emphasizes reducing cost and defects. In DMAIC, a major component of a Six Sigma approach, sources of waste are identified. Then changes are made in the process to reduce effort and increase the quality of the product produced. GQM and Six Sigma are complementary. Both approaches rely heavily on the measurement of input and output metrics. Mining development artifacts can provide usable metrics for the application of DMAIC and GQM in the software domain.

Categories and Subject Descriptors
D.2.9 [**Software Engineering**]: Management – *productivity, software process models*.

General Terms
Management, Measurement, Reliability, Theory.

Keywords
Six Sigma, GQM, Process Improvement, Repositories

1.INTRODUCTION

Six Sigma and CMMI are two different approaches to process improvement that come from different perspectives. The two approaches are complementary. Combining the strengths of each approach yields an approach that focuses strongly on continuous and incremental process improvement while seeking metrics that are appropriate to the reality of software development. Within this perspective, we propose that mining artifacts found in large software repositories can provide useful metrics to support a program of continuous process improvement. Mining artifact repositories provides useful process metrics without adding overhead to the process being observed. Instrumenting artifacts, rather than people, supports other kinds of process improvement. While we have not yet put our ideas into practice, in this paper we explain our reasoning and place the proposal in the context of recent and historical trends in software and management theory. In future papers we will describe the experience of putting these ideas to use in a large software organization.

2.GQM, DMAIC, and Repository Mining

GQM GQM is a disciplined approach to defining and collecting metrics as part of a software development process improvement program. Originally developed by Basili's group in University of Maryland, it has since been adopted, slightly modified to GQ(I)M, as part of the guidelines for the SEI's CMMI. GQ(I)M stands for Goal-Question-(Indicator)-Measure. The 10 steps in a GQM process identify business goals, identify the questions to ask related to these goals and measurements that will help answer them, and create a plan to collect the measurements. The CMMI and GQM focus on measuring and managing the development process to predictably and reliably achieve organizational goals.

Six Sigma is a disciplined approach to continuous process improvement designed to increase customer satisfaction and profits while reducing defects and cost. The name derives from the ideal of 3.4 defects per million opportunities. Organizations with a three sigma level of defects (typical of software) are candidates for improvement. Beyond, six sigma, the investment is assumed not to be cost effective. Originally developed at Motorola, it has been popularized by many high profile companies including Honeywell, GE, 3M, Kodak, DuPont, and Allied Signal. Today it is widely applied to manufacturing and service-related processes. A good description of Six Sigma can be found on the SEI web site [21].

The origins of Six Sigma are instructive for software development. In 1985, Bill Smith argued that if a product was found defective and corrected during the production process, other defects were bound to be missed and found later by the customer during use of the product. This raised the question, was the effort to achieve quality really dependent on detecting and fixing defects, or could quality be achieved by preventing defects in the first place through manufacturing controls and product design? Smith's observation echoes the third of Deming's 14 points, not to rely on inspection and testing to achieve quality [3].

Six Sigma is an iterative approach based on undertaking a continuous series of initiatives to improve performance over time. The process improvement model is called DMAIC, an acronym for the following 5 steps.

1) *Define* what is important. What matters to the customer?
2) *Measure* performance. How are we doing? What aspects of the process are affecting customer value?

3) *Analyze* opportunity. What could we be doing better? What are the variables that affect performance?
4) *Improve* the process. Plan a strategy for improvement and test it out.
5) *Control* the process. Institutionalize practices to sustain the improvement.

A key concept in Six Sigma is the "big Y". What is the greatest gain in measurable customer value (measured on the y-axis), that can be achieved by an investment (measured on the x-axis) in process improvement. At the beginning of each initiative iteration the process is analyzed to find threats to customer satisfaction and opportunities for improvement. Traditionally, the measurement part of the process is based on practices of statistical quality control.

A typical example of the application of Six Sigma might involve light bulb manufacturing. The measure phase discovers that recently the variance in the thickness of the glass has been increasing. Continuation of this trend could lead to breakage in shipping and higher costs. The source of the variance is identified (worn machine part, new operator, supplier, etc.) and corrective action is undertaken.

The strengths and weakness of GQM and DMAIC are complementary [9]. Implementations of the CMM are sometimes criticized for emphasizing repeatability over improving productivity. Six Sigma is sometimes criticized for being inappropriate for development processes characterized by the unique intellectual efforts of knowledge workers. DMAIC's strength is its focus on continuous process improvement and its iterative and incremental approach to achieving it. GQM's strength is in defining metrics that are appropriate to the business goals and to the process. In this context, the kind of information that can be found in software repositories adds value.

Mining software development repositories can be used to detect weaknesses and identify opportunities to improve the development process. Repository measurements can be collected without adding significant process overhead. In the past, there has been an impediment to using the kind of data that can be collected and inferred from the mining of software repositories because of its perceived lack of methodological and statistical rigor. However, there is an emerging understanding within both the GQM [16] and Six Sigma [15] communities that this kind of data yields real value. "In rapidly changing environments, precise numbers and elaborate statistical analyses are often less valuable than simpler answers to insightful, well-directed questions" [15].

Moreover, recent theories in process management and process improvement place greater value on the kinds of knowledge that can be found by mining development repositories in the pursuit of process improvement. Theories such as Obsolete Theory [13], Lean Management [20], Theory of Constraints [7], and Agile methods teach us to focus more on execution and less on planning, reduce waste, look for bottlenecks, balance reliable measures with measures that show value, and embrace change as a strategic advantage.

The concept of waste in lean manufacturing is attributed to Toyota's Taichi Ohno and Shigeo Shingo [17]. Waste is defined as any activity that consumes resources but delivers no value to the customer. Defects are a source of waste – once allowed to occur, they require rework at best, and at worst, lead to less useful or returned products and unhappy customers. Delay is also a source of waste, not only from increased development cost, but also from opportunities missed in the marketplace and in resources not available to produce more value. In Six Sigma, Black Belt practitioners achieve their rating through training and proven experience, where proven experience comes from achieving measurable reductions in waste.

3.Software Development as Production

Software development in large organizations can often be viewed as a production process. A typical team develops multiple variations (possibly variations over time) of a core product. In [23] software product line development is compared to manufacturing cars, where the basic car can be varied in terms of engine, seats, upholstery, etc. (In fact, part of Toyota's Lean Manufacturing is the SMED, Single Minute Exchange of Dies, concept of process retargeting for major variations.) When software development is viewed as production, features can be viewed as inventory. In this light, the Extreme Programming principle of "build the simplest thing" can be seen as a correlate of Ohno's concept of Kanban or Just-In-Time inventory. (Test-first and pair programming correspond to Shingo's Poka-Yoke or mistake-proofing, and source inspection, respectively.)

When software development is viewed as a production process, a valid question becomes, where are the bottlenecks? In software product line development, bottlenecks can be caused by poor architecture and code rot, problems with requirements, or linkages and dependencies between project elements. Inspecting development artifacts can be an effective aid to identify and measure potential bottlenecks.

If a project's change history shows a pattern of recent changes affecting more than the usual number of sites, an architecture problem might be indicated. Recent additions could be of a type that the architecture does not well support. Decreasing localization of change could also be a sign of code rot – repeated change over time tends to make code progressively more brittle to additional modification. In either case, the area of modification could be a candidate for refactoring.

Analysis of the email or SMS archives could reveal a volume of messages between developers and the internal customer prior to progress being made on specific features or requests. This pattern could indicate a problem with the process of requirements capture or specification. Further analysis of the types of features involved and the nature of the misunderstanding would be warranted.

Standard product line domain analysis practices, e.g. [12], are facilitated by the analysis of artifacts. A pattern of a high frequency of modification on the same pieces of code across multiple variants could indicates an opportunity to save effort by building a code generator to handle the differences [23]. Two pieces of code that often change together might indicate high affinity or coupling, while code artifacts that seldom changes together exhibit the opposite. Code sections that rarely see change are good candidates for inclusion in the core domain architecture. Analyses of these types can help build effective architectures that better support product line and model driven development.

Frequent use of manuals or searching the web may reveal an opportunity for training on the issues in question. Similarly a comparison of artifacts between two teams, where one team is consistently more productive than the other, might reveal types of training that would best aid the less-performing team.

Reducing defects can also be improved through inspection of process artifacts. Correlating defect reports with prior activities may indicate opportunities to reduce defects through process change. By analyzing sequences of behavior, it might be possible to identify where development shortcuts have been taken. Leveson's STAMP model for reliably safe systems assigns the root cause of system failures to failures in constraints on the process. Using this model, artifact evaluation could identify patterns of violating the constraints before they lead to defects in the product.

As software development organizations mature to CMMI levels 3, 4, and 5, their process artifacts contain more keys for correlation. Change events refer to change requests, and communications more often reference specific features, requests, and code. It is likely that as organizations use and find value in artifact analysis, properties of the artifacts that enhance their value for analysis will improve. The process we propose corresponds the CMM level 5 Technology Change Management, but adds specific measurement practices to drive the process.

4.MINING GLOBAL SOFTWARE DEVELOPMENT ENVIRONMENTS

In the past, approaches such as DMAIC and GQM have advocated putting measurement practices in place that collect measurements to feed the overall method. We think that such instrumentation approaches suffer from two main drawbacks: (1) they introduce measurement overheads in the process that can slow the process, and, more seriously, (2) they reify measurements and their instrumentations, affecting the behavior of the process and its participants. In contrast, we advocate that appropriate measurements be mined from the existing process and product data.

Fortunately, the existence of global software development environments (GDE), like SourceForge [22], and Corporate Source [4,5] and it's successor SourceShare [24], provide ample opportunities to collect appropriate data. A GDE provides a repository for multiple projects in an organization to store all project information in a single place [11]. Participants create a new project in a GDE, and subsequently all project communication (through email or discussion forums), version control data, and problem report workflows are captured and maintained in the GDE. We propose that GDEs can be extended with an DMAIC dashboard to interactively provide required metrics and analyses.

Since we do not have practical experience with this approach yet, we give some hypothetical examples of analyses and measurements that could be usefully mined from GDEs. One of the main tenets of Six Sigma is to reduce the number of defects per million opportunities in a product. In the case of software development, the opportunities for introducing defects are numerous, ranging from the abstract (error in understanding a requirement) to concrete (error in a program statement). Therefore, one category of charts that will be useful addition to GDE would be running charts of open defects per opportunity, e.g., open defects per thousand lines of source code. As the lines of code progress over time, and the defects are opened and closed, these charts can give a sense of how the process is maturing over time.

In the spirit of Open Source, a GDE advocates that users (or customers) of a software project have early and continuous visibility of the process. Hence, potential users participate in the email lists for discussions on feature requests and design changes. These discussions can provide a useful measurement of how involved are the users in the process? One can measure the number of emails coming from users versus developers over time.

5.RELATED WORK

5.1Effort Estimation

Previous works on effort estimation have been focused on the metrics from the development of an entire system. The AMEffMo [10] project has shown that it is possible to estimate the amount of effort that went into four separate projects using the metrics that was gathered from each project. It should stand to reason that effort estimations for individual components of a project are also possible. In an evaluation study performed by Mockus and Graves [2] they set forth an algorithm that is able to estimate effort based on the size of a modification request as well as the type of change requested. It was even stated that if the effort for each change was known, then the size of the change would be known. However, the reliability of developer recorded efforts per module is questionable [8]. Therefore, effort was divided among all the changes performed within a given period.

Four variables were found to be significant in affecting the effort estimation model. The number of changes per modification request, individual developer productivity, the nature of the changes, and the time difference between the detection of decay and the request for the change [19]. In addition to these four main variables, other metrics can be used to measure effort such as the requirements or specification documents. [14] These were the major factors in this particular project and may be used as a starting point for investigating the cause of bottlenecks in a development process.

5.2Communication Gap

As email is a viable platform for communication among developers of a system, these messages may become important information in understanding the difficulties of developing certain features or modules of a system. The storing of these email messages into a database and later mining their contents has been proven to be possible in the Apache Web server project. Since these email messages follow a relatively structured format with information regarding the sender, receiver, date, and subject, these attributes have been shown they are able to be use as search variables in database query. Furthermore, the dates of these messages may be matched to the development timeframe of a particular feature in order to analyze bottlenecks and causes of increased effort during development. The number of developers that participated in changes or development can also be found through these techniques.

In addition to email messages are the pools of information located in the change logs of a CVS repository. In a study of the CVS repositories of an open source project, Mozilla and Bugzilla, [6] the large scale and ongoing nature of the project did not affect the mining. All that was needed was a time frame for which to analyze the data. This time frame restriction might also help narrow down changes performed at the same time as the development period for a feature or module that is being analyzed. Usually associated with each ChangeLog is a Bugzilla bug report which is free formed text written by the developer. These might also indicate where time was spent and what difficulties were encountered.

6.CONCLUSION

Large repositories of software development artifacts contain a potential wealth of information about the behavior and performance of software development processes. This data is available without adding overhead to the process in question.

Using this knowledge effectively requires an organizational commitment to change, and a context for asking the right questions. We believe that the combination of Six Sigma's DMAIC and CMMI's GQ(I)M, provides such a framework. We have explained the rationale and discussed recent trends in project management theory that add support to our view. As we are only now beginning to apply our ideas in an industrial setting, reports on our experience are left to future publication.

7.REFERENCES

[1] Alonso, O., Gertz. M., and Devanbu, P. **"**Database Techniques for the Analysis and Exploration of Software Repositories**"** *MSR '04: International Workshop on Mining Software Repositiories*, Edinburgh, UK, 2004. http://www.cs.ucdavis.edu/~devanbu/msr04.pdf

[2] Atkins, D., Ball, T., Graves, T., and Mockus, A. "Using Version Control Data to Evaluate the Impact of Software Tools: A Case Study of the Version Editor." *IEEE Transactions on Software Engineering,* 28(7), July 2002, 625-637. http://www.research.avayalabs.com/user/audris/papers/vedraft.pdf

[3] Deming, W.E., *Out of the Crisis*, MIT Press, Cambridge, MA, 1986

[4] Dinkelacker, J., Garg, P.K., Miller, R., an d Nelson, D. "Progressive Open Source." In *Proceedings of the International Conference on Software Engineering (ICSE'02)*. Orlando: ACM Press, 2002, 177-184. http://lib.hpl.hp.com/techpubs/2001/HPL-2001-233.pdf

[5] Garg, P.K. and Dinkelacker, J. "Applying Open Source Concepts Within A Corporation." *1st ICSE International Workshop on Open Source Software Engineering, Toronto, Canada, May, 2001.* http://sunarcher.org/jamie/pubs/OpenSourceInCorpEnvs_2001.pdf

[6] German, D.M. "Mining CVS Repositories: The SoftChange Experience." In *1st International Workshop on Mining Software Repositories*. May 2004, 17-21. http://turingmachine.org/files/papers/2004/dmgmining2004.pdf

[7] Goldratt, E.M. *The Goal: A Process of Ongoing Improvement*, 2nd rev. ed. North River Press, 1992.

[8] Graves, T.L. and Mockus, A., "Inferring Change Effort from Configuration Management Data.**"** In *Metrics 98: Fifth International Symposium on Software Metrics,* Bethesda, Maryland, November 1998, 267-273. http://www.research.avayalabs.com/user/audris/papers/effort

[9] Hong, G.Y. and Goh, T.N. "A Comparison of Six Sigma and GQM Approaches in Software Development." *Journal of Six Sigma and Competitive Advantage*,1(1), 2004, http://www.inderscience.com/storage/f125119371042861.pdf

[10] Huffman Hayes, J., Patel, S., and Zhao**, L.,** "A Metrics-Based Software Maintenance Effort Model" In Proceedings of the *8th European Conference on Software Maintenance and Reengineering*, Tampere, Finland, March 2004. pp. 254-258. http://selab.netlab.uky.edu/Homepage/csmr_ameffmo_hayes_2004%5Eas_published.doc

[11] Inoue, K., Garg, P.K., Iida, H., Matsumoto, K. and Torii, K.. "Mega Software Engineering." Accepted for *PROFES 2005*, Finland, June 2005

[12] Jacobson, I., Griss, M.K., and Jonsson, P. *Software Reuse: Architecture, Process, and Organization for Business Success.* Addison-Wesley, Reading, MA, 1997

[13] Koskela, L., and Howell, G. "The Underlying Theory of Project Management is Obsolete." In *Proceedings of the PMI Research Conference*, 2002, 293-302. http://www.leanconstruction.org/pdf/ObsoleteTheory.pdf

[14] Lehman, M.M., Perry, D.E., and Ramil, J.F. "Implications of Evolution Metrics on Software Maintenance." *ICSM'98*, November 1998. http://www.ece.utexas.edu/~perry/work/papers/feast2.pdf

[15] Martin, R. "Validity vs. Reliability: Implications for Management." *Rotman Magazine*, Winter 2005. http://www.rotman.utoronto.ca/integrativethinking/ValidityVSReliability.pdf

[16] Morasca, S., Briand, L.C., Basili, V.R., Weyuker, E.J. and Zelkowitz, M.V. "Comments on 'Towards a Framework for Software Measurement Validation'.*" IEEE Transactions on Software Engineering,* 23(3), March 1997, 187-188

[17] Ohno, T. *The Toyota Production System: Beyond Large-Scale Production.* Productivity Press, 1988.

[18] Park, R.E., Goethert, W.B., and Florac, W.A. *Goal-Driven Software Measurement —A Guidebook,* Software Engineering Institute, 1996. http://www.sei.cmu.edu/pub/documents/96.reports/pdf/hb002.96.pdf

[19] Perpich, J.M., Perry, D.E., Porter, A.A., Votta L.G., and Wade, M.W. "Anywhere, Anytime Code Inspections: Using the Web to Remove Inspection Bottlenecks in Large-Scale Software Development." *1997 International Software Engineering Conference (ICSE97)*, Boston Mass, May 1997. http://www.ece.utexas.edu/~perry/work/papers/icse97.pdf

[20] Poppendieck, M. and Poppendieck, T. *Lean Software Development: An Agile Toolkit.* Addison-Wesley, Reading MA, 2003.

[21] Siviy, J. "Six Sigma." Software Engineering Institute, 2001. http://www.sei.cmu.edu/str/descriptions/sigma6_body.html

[22] http://www.sourceforge.net

[23] Weiss, D. and Lai, C.T.R. *Software Product-Line Engineering: A Family Based Software Development Process.* Addison-Wesley, Boston, MA, 1999