Lightweight Risk Mitigation for Software Development Projects Using Repository Mining

Stephen P. Masticola
Siemens Corporate Research
755 College Road East, Princeton NJ 08540
Stephen.masticola@siemens.com

Abstract

Many software projects fail to deliver their needed results on-time and on-budget. There are a variety of reasons why this may occur. For some of these reasons (notably deterioration of the codebase), corrective action is often difficult to cost-justify or to implement efficiently in practice. To address this, an approach of lightweight risk mitigation is proposed: mine risk data from configuration management and defect tracking systems, integrate this data with project-cost data in a flexible dashboard, and facilitate strategic refactoring with semi-custom transforms where necessary. This prescriptive information would simultaneously help the project manager to cost-justify repair efforts and lowers the cost of finding and fixing hot spots.

1. Introduction

It has been estimated that, in the year 2004, only 29% of software projects successfully delivered adequate results on-time and on budget. [1] Although this is certainly an improvement from past years, the cost of failed software projects remains a serious concern for the enterprises undertaking them. A persistent pattern of software project failures can, in fact, make some software-intensive enterprises unprofitable.

Even where project managers have a good intuitive understanding of why their project is in trouble, they often have difficulty justifying the cost and time of correcting its root causes. Data collection, decision support, and remediation are currently expensive, labor-intensive, and distracting, but, with improved tool support, they might not be inherently so.

Risk mitigation activities can be viewed collectively as a business process. To run the business as efficiently as possible, one wishes to minimize the cost and effort of risk mitigation. Hence, the focus should be put on *lightweight risk mitigation*, i.e., risk mitigation that aims at being effective with low cost and time impact to the project. To keep costs acceptable, repository mining will likely play a central role in lightweight risk mitigation.

Three activities are necessary in lightweight risk mitigation: data collection (via repository mining), decision support (reducing the mined data and making it actionable), and remediation support. This paper discusses some preliminary ideas for supporting the first two activities using repository mining and for harmonizing these activities with remediation support. The widespread problem of code quality in general, and codebase deterioration in specific, is used here as a motivating special case.

2. Background

The ideas here arose from the author's involvement with a software component which will here be called "Pocahontas." Pocahontas is a large, major component of a very complex real-time system. At the time of study, Pocahontas was roughly in the middle of its lifetime.[1]

Pocahontas development was done by a motivated and efficient staff, and was supported by a very good set of repository tools and procedures. Pocahontas, however, has a number of ongoing maintainability issues. Despite these, strategic refactoring was generally given lower priority than feature enhancements. One result appeared to be a steady deterioration of the Pocahontas codebase, which might possibly have resulted in serious adverse business impacts. The premise that the codebase was deteriorating was supported by the available project health metrics, but was difficult to prove more directly.

[1] Pocahontas is a real project, but the name is fictitious. To protect the confidentiality of our industrial clients and their businesses, all of the identifying information about the Pocahontas project, including quantitative information, has been disguised, in ways that do not affect the conclusions of this paper.

Management must balance the cost and ROI of maintenance tasks against those of feature enhancements over the product's lifecycle, and indeed over multiple product lifecycles. This balance between maintenance and feature enhancement is thus an important strategic business decision, and one that would benefit from a quantitative understanding of the total cost and ROI of each of these two activities. Lowering the cost and effort of making such measurements makes them possible; lowering the cost and effort of strategic refactoring can make it possible and save money on both small and large scales.

Because of these issues, it appeared useful to provide the Pocahontas project managers with the tools they needed to cost-justify strategic refactoring tasks. To date, the focus of this work has been on using automated code inspection to mine violation counts from different release versions of Pocahontas. My goal behind this is to detect "bad smells" [5] in the source code, to support decision-making about strategic refactoring.

3. Decision Support

In many respects, decision support is properly the central activity of lightweight risk mitigation. Project managers have only limited resources (budget, people, time, etc.) that they can put to work in their projects. The purpose of lightweight risk mitigation is to help project managers to conserve these resources, use them wisely, and justify obtaining more when needed.

3.1 Scenarios

A decision support system will be the most useful to a project manager when the decisions answer the questions that are important to running the project efficiently. Some examples of questions a proactive project manager may ask are the following:

- How many incorrect bug reports are we getting? What is their cost?
- What parts of the software have the worst "bad smells"?
- What parts of the software are the most expensive to maintain?
- What parts of the software are consuming the most developer resources?
- What parts of the software are being fixed most often?
- How are these factors changing over time?
- Which of these factors are under control, and which are out of control?
- Can the "assignable causes" be found for the factors that are out of control?
- How much would refactoring help, based on past experience?
- How much would a refactoring effort cost, based on past experience?

There are two points to note about these questions. First, many of them are answered by *project health measures* that are phrased in economic terms: how much did something cost or how much effort would some activity take. The second point is that these questions are of a fairly ad-hoc nature. Many similar questions exist, so a fixed set of reports might not be adequate for decision support. However, to keep the cost of using the decision support system low, the data reduction should be kept easy to use by non-experts.

A less obvious point comes about when the project manger is trying to find *assignable causes* [7] for out-of-control project health measures: the original measures from which the answers are derived need to be readily accessible. This implies that the decision support system should allow drill-down to the original data, or as close to it as possible.

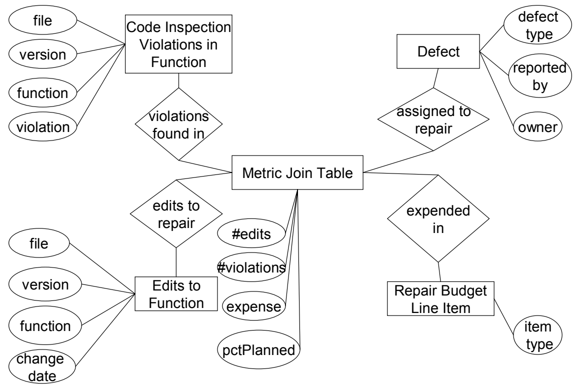

Figure 1: Example join table for software project decision support.

3.2 Data Integration

Fortunately, the dimensional modeling approaches used in on-line analytical processing (OLAP) and data warehousing [6] provide at least a partial strategy for combining flexibility with ease of use. In dimensional modeling, data are integrated as one or more "join tables," that include metrics that can be summarized by "rolling up" dimensions and/or eliminating records that are not of interest.

Figure 1 shows an example of a join table for use in a software project decision support system based on repository mining. The join represents summary information about defect repairs. Each record of the join in Figure 1 represents the following: "Some number of edits was made to a function to repair a certain defect. Before the defect was reported, some number of violations of a specific type was reported in that function by the automated code inspection tool. Some amount of money was spent for the repair, which was some percent of the given planned budget line item for repairs of defects of this type."

3.4 Example: Coding Convention Violations in Pocahontas

The repository mining effort in the Pocahontas project has to date focused mainly on exploring automated code inspection. A static code-checking tool was used to analyze six major releases of Pocahontas.

Figure 2 shows the fraction of the code inspection violations found in Pocahontas. The distribution of violations is more or less typical for a software project of its size and age.

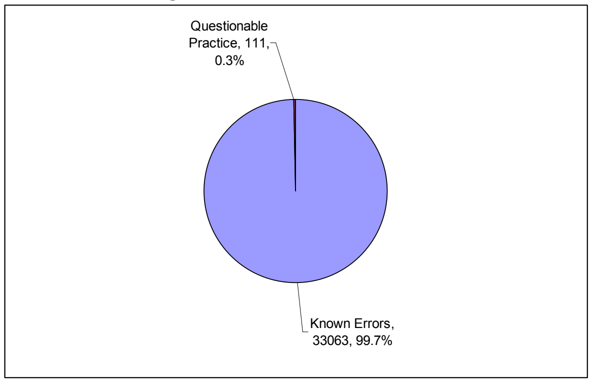

Figure 2: Total violations by major category in Pocahontas version 12.

Summaries like Figure 2 tend to focus attention on the known errors and lead people to dismiss "bad smells." Such practice is risky; all convention violations intuitively indicate "bad smells," and correcting them may focus strategic refactoring attention where it is most needed.

Figure 3 shows the trends in "Questionable Practices" violations. The uniformly increasing trend tells the project manager that it is getting progressively harder to correct the "bad smells." However, it doesn't, by itself, say whether or not the code is deteriorating overall.

Figure 4 gives a little better information about trends in the maintainability of the code, by normalizing to violations per LOC. There was no trend of steady deterioration throughout the time period that was examined. Instead, violation density improved for versions 7 through 9.

Analyzing assignable causes, it turned out that this was due to the addition of several new modules to Pocahontas in versions 10 through 12. These new modules had lower violation density than the older code.

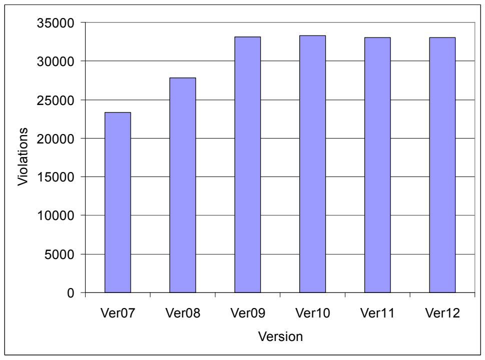

Figure 3: Trends in "questionable practices" violations. (X-axis indicates version number, not time.)

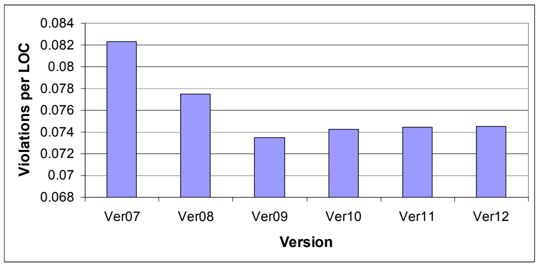

Figure 4: Trends in violation density. (X-axis indicates version number, not time.)

Figure 5 gives some indication of the place from whence the worst smells come. File31 shows a large spike in violation density for version 8. This measure is likely to be out of statistical control.

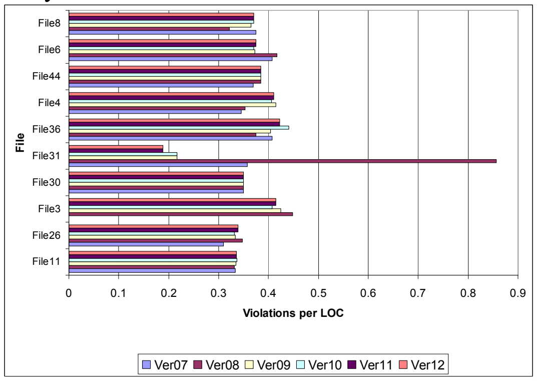

Figure 5: Violation density trends for the 10 densest files.

4. Data Collection

Figure 6 shows a typical data collection architecture, in which repositories of several types are instrumented for lightweight risk mitigation. Some key points for organizational acceptance of data collection are that:

- Data collection should operate automatically.
- The data should be as complete and accurate as possible. (Automation helps with this.)
- The ETL processes should not directly integrate the data into the dimensional models, because the schemas could be changed on either side. Timestamps and necessary metadata should be retained to support assigning causes to out-of-control measures. [7]

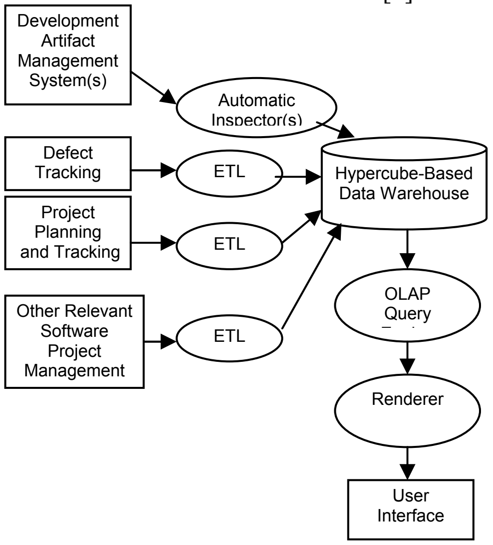
Development Artifact Management System(s)
Automatic Inspector(s)
Defect Tracking
ETL
Project Planning and Tracking
ETL
Other Relevant Software Project Management
ETL
Hypercube-Based Data Warehouse
OLAP Query
Renderer
User Interface

Figure 6: Typical data collection architecture for software project decision support.

5. Remediation Support

When convention checkers are used to mine "bad smells," it may be advisable to develop specialized refactoring transforms, to reduce the total work of repairing violations. As an example, Pocahontas version 12 violated one particular questionable-practice rule (concerning function parameters) in nearly 4000 locations, and the worst single file had over 500 violations. Cleaning up all instances of this violation manually would thus have been prohibitively costly.

More generic refactoring tools are also quite useful in remediation support. For example, the refactoring of "change smells" in [3] used Fowler et. al.'s *Extract Method* and (likely) *Extract Class*. [5] These transforms are already supported by at least one commercial IDE plugin tool [8].

By creating bundles of semiautomated transforms that specifically remediate the "bad smells" detected by repository mining, we harmonize the remediation support tools to the decision support.

6. Conclusions and Future Work

Lightweight risk mitigation based on repository mining and harmonized remediation support offers the hope that it might be possible to economically justify and efficiently counteract the effects of codebase deterioration in commercial practice. What remains to be done is:

- An exploratory version of the data collection and decision support systems.
- Statistical quality control charting.
- A database aggregation function for density data. This will involve a specialized representation for density data.
- A low-effort mechanism to extend the decision support system by adding new data reductions (and reports) as needed.

Also worth exploring is the possibility of mining artifacts from agile processes, such as backlog lists in Scrum [9], and how clustering techniques can best be used in the context of guiding refactoring [4].

References

[1] J. Johnson, *My Life is Failure: 100 Things You Should Know to be a Successful Project Leader*, The Standish Group International, West Yarmouth, MA, 2006.
[2] M. VanHilst, P. K. Garg, and C. Lo. "Repository Mining and Six Sigma for Process Improvement," *International Conference on Software Engineering, Proceedings of the 2005 International Workshop on Mining Software Repositories*, Saint Louis, MO, 2005.
[3] J. Ratzinger, M. Fischer, H. Gall. "Improving evolvability through refactoring." *International Conference on Software Engineering, Proceedings of the 2005 International Workshop on Mining Software Repositories*, Saint Louis, MO, 2005.
[4] O. Alonso, P. Devbanu, and M. Gertz. "Database techniques for the analysis and exploration of software repositories." *International Conference on Software Engineering, Proceedings of the 2004 International Workshop on Mining Software Repositories*, Edinburgh, Scotland, 2005.
[5] Fowler, M., Beck, K., Opdyke, W., and Roberts, D. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley, 1999.
[6] R. Kimball and M. Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. Wiley, 2002.
[7] W. A. Florac and A. D. Carleton, *Measuring the Software Process*, Addison-Wesley, 1999.
[8] SlickEdit, Inc. http://www.slickedit.com/
[9] K. Schwaber and M. Beedle. *Agile software development with Scrum*. Prentice-Hall, 2001.
[10] D.C. Montgomery. *Introduction to Statistical Quality Control*. Wiley, 2004.