Risky Files: An Approach to Focus Quality Improvement Effort

Audris Mockus
Avaya Labs Research
211 Mt Airy Rd
Basking Ridge, NJ
audris@avaya.com

Randy Hackbarth
Avaya Labs Research
211 Mt Airy Rd
Basking Ridge, NJ
randyh@avaya.com

John Palframan
Avaya Labs Research
211 Mt Airy Rd
Basking Ridge, NJ
palframan@avaya.com

ABSTRACT

As the development of software products frequently transitions among globally distributed teams, the knowledge about the source code, design decisions, original requirements, and the history of troublesome areas gets lost. A new team faces tremendous challenges to regain that knowledge. In numerous projects we observed that only 1% of project files are involved in more than 60% of the customer reported defects (CFDs), thus focusing quality improvement on such files can greatly reduce the risk of poor product quality. We describe a mostly automated approach that annotates the source code at the file and module level with the historic information from multiple version control, issue tracking, and an organization's directory systems. Risk factors (e.g, past changes and authors who left the project) are identified via a regression model and the riskiest areas undergo a structured evaluation by experts. The results are presented via a web-based tool and project experts are then trained how to use the tool in conjunction with a checklist to determine risk remediation actions for each risky file. We have deployed the approach in seven projects in Avaya and are continuing deployment to the remaining projects as we are evaluating the results of earlier deployments. The approach is particularly helpful to focus quality improvement effort for new releases of deployed products in a resource-constrained environment.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*process metrics, performance measures*

General Terms

Defects, Software risk management, software quality prediction, Knowledge transfer, Software repositories

Keywords

Software quality assurance (SQA), version control, defect tracking, defect prevention, risky file analysis

ESEC/FSE '13, August 18-26, 2013, Saint Petersburg, Russia

1. INTRODUCTION

Risk is omnipresent in software development and many development strategies are explicitly or implicitly geared to minimize it [11]. In all projects we investigated, we found that less than 1% of project source code is involved in more than 60% of customer reported defects (see Table 1). The approach presented here is geared to identify and mitigate customer-defect risk posed by such small parts of the code.

The approach relies on an empirical relationship between past history of development and future customer-reported defects. In particular, that relationship shows that areas with large numbers of past changes and large numbers of authors who left the project tend to have much higher risk of having customer defects in the future [10]. The analysis starts by collecting commits to the code (typically over a period of the past two to three years) and linking files in each commit with the associated change request (MR), date, and author. The project's issue tracking database is then used to identify customer reported defects and the company personnel directory is used to determine if the author has left the company. The same file may be modified in multiple branches or even different repositories (when the same code is used in multiple projects). Our approach links such files by identifying if over the commit history versions of the file existed that were very similar or identical among the branches or repositories [2, 6]. We use a similar approach to identify files that have an open source origin by checking if any versions match our large repository of open source code [8, 6]. Once the equivalent files are identified, the defects reported for all related files are linked to each equivalence class. In addition to defects, we also obtain a number of other statistics to identify expert developers and to help them make the decisions on how to remediate the risk.

Table 1: Context of the Projects

Project	N. of Files	Files in Top 1%	% of CFDs in Top 1%
Project A	24.6K	246	80
Project B	21K	212	74
Project C	2.2K	22	100
Project D	76.7K	302	61
Project E	36.5K	365	100
Project F	10.3K	103	63
Project G	7.2K	134	100

Note that projects, that have not experienced complete transitions or dramatic attrition of long-term participants, typically have experts who are able to identify such risky

areas. Yet even in such cases, experts may need a quantitative assessment to justify and focus more effort-intensive risk mitigation strategies.

The key contribution of this work is to describes experiences at Avaya with an approach to identify and remediate the riskiest areas of the code, with a particular focus on providing tools and procedures needed for a successful remediation.

2. RELATED WORK

Much research has been done that uses version control and issue tracking data to predict which files will have defects, e.g., [3], and what factors are good predictors of defects, e.g, [1].

Also, some approaches focus on predicting which changes will introduce defects [13] or focus on the defects that have the most impact [14]. All of this work assumes that some quality improvement action will follow the prediction. Our experience has been that development teams do not find the defect prediction to be useful, e.g., [14]. Thus we focus here on integrating risk identification and mitigation into a single easy-to-use package that includes analysis tools, checklists, and training.

Some projects have been transferred among locations and the impact of such transfers was shown to reduce the productivity, see, e.g., [9], thus reducing the amount of resources that could be used to reduce the risks and to improve quality. The identification of changes that are likely to introduce defects [13, 5] was integrated with a patch delivery process and provided a tool that flagged changes that exceeded a risk threshold. It was, however, focused on changes and not on risky areas of the code and it did not include remediation tools and practices as our approach does.

3. CONTEXT AND DATA

Avaya is a premier provider of call center and business communication software. It formerly was a part of AT&T, then Lucent. It has acquired a number of companies, notably the enterprise communication and networking division of Nortel. Avaya has numerous software projects with approximately 3000 developers with major R&D centers in the United States, Canada, Ireland, Russia, Romania, and India.

To conduct software intelligence activities we collect version control and issue tracking data from hundreds of Avaya projects and create a source-code-mart as described in [8]. Avaya primarily uses ClearCase, Subversion and Git for version control, with isolated projects still using SCCS. Modification Requests (MRs) are tracked via ClearQuest or JIRA, with isolated projects still using Sablime. Most of the projects use tools provided by the tools organization, thus it is possible to access source code and MR data from a few centralized locations. There are a number of steps that we go through to validate the quality of the collected data, see for example, [7].

4. METHOD

The method starts from the identification of candidate projects, proceeds to conducting analysis of the relevant data, populating the risk mitigation tool. This is followed by a meeting discussing the initial analysis, and a training session for expert developers based on a focused analysis that is geared to the specific needs of the project and based on feedback from the initial meeting. The projects are identified by working with executives as part of communication about the state of software in Avaya [4]. Once a project is identified we conduct the relevant analysis.

We gather data about each file in the source code repository and then prioritize the files and identify a candidate set of riskiest files (about 1% of all files) using a weighted algorithm based on empirical results described in Section 4.1. Product teams are then encouraged to review the candidate set of riskiest files to determine if the files do in fact pose a quality risk and, if so, to identify and deploy actions to mitigate the risks. Section 4.2 provides a checklist of suggested actions based on our experience assessing and mitigating candidate risky files in Avaya projects.

4.1 Model used for prioritization

The example below illustrates the way we determine weights used to prioritize the risk. The illustration is based on project B for which we conducted the analysis in early 2012. We chose all commits to the version control system for the last three years (since 2009) and fit a logistic regression model with an observation representing a single file, the response being whether or not the file had a CFD, and the predictors including various factors that have been shown to be associated with defects, e.g., size, complexity, changes, and past defects [3, 13, 5]; co-change (logical) dependencies [13, 1, 5]; call/data flow and the developer workflow network [1, 10]; test coverage [12]; and organizational change (expert owners leaving the project) [10]. As is typical in software projects, many of these predictors are highly correlated. We, therefore, chose a subset that was uncorrelated and explained a large fraction of the observed variance [7]. In particular, for project B we found that only a few predictors of future CFDs were needed: the number of past changes $nD - 1$, the number of System Verification (SV) MRs $nSV - 1$, the number of authors who left $nAL - 1$ and the presence of of static analysis warnings $nSW > 0$. The regression results are in Table 2. The weighting that was

Table 2: Logistic regression predicting CFDs. 21179 observations, 25% of deviance explained

Predictor	Estimate	StdErr	Deviance Explained
Intercept	-4.5	0.07	
$\ln nD$	0.55	0.04	1602
$\ln nAL$	0.9	0.07	138
$\ln nSV$	1.6	0.1	280
$nSW > 0$	0.5	0.13	14

used for ordering the candidate risky files in project B that results from the model is then:

$$0.55 \ln nD + .9 \ln nAL + 1.6 \ln nSV + .5I(nSW > 0).$$

4.2 Checklist of Suggested Actions

The approach is typically used in the early phases of development of new releases of a deployed product, and the actions are typically implemented either for the upcoming release, or, if the resources do not allow it, the remaining actions are implemented in subsequent releases. What follows is a brief summary of the checklist we constructed that suggest how to interpret and act upon various pieces of information.

Assign subject matter experts (SMEs) who understand a product's feature, its business usage, and the por-

tions of the code base that support the feature to examine each candidate risky file in priority order. Distribute this task across a set of subject matter experts (SMEs), typically the file owner for each candidate risky file. As described in Section 5, our online tool provides assistance in identifying potential SMEs.

For each candidate risky file, the SME analyzes the file and associated data and recommends one of the following three actions:

1: No action is required, if, for example, development is complete for this file; the candidate file will not be used in the near future; the candidate file is changed with a risky file, but is not itself risky.

2: Establish a control program involving additional review and testing of all changes to the file to mitigate risk from changes to the file. The following steps are suggested for such program.

1. For files with many many authors, the file owner creates a 1-page design guidance document to be consulted by anyone changing the file. The same design guidance document typically will apply to a set of files that all contribute to the same feature. Similarly, a test guidance document would be created.

2. Conduct static analysis after any changes made to the file. Suggested policy is that new defects identified by static analysis should not be allowed.

3. Code inspection of all changes should include the file owner and at least one other SME in addition to the author. Include design and test guidelines, static analysis results, and inspection checklists as reference material for the reviewers.

4. Create and execute unit tests that reflect changes to the file and follow the test guidelines, when available. Automate the tests where possible and add them to the integration build tests.

5. Alert functional test and system verification staff whenever a change involving a risky file is included in the software build. This alerts the test team to execute tests associated with past defects contributed by the file.

3: If the file is determined to be too fragile, complex, or poorly structured to support continued changes, the SME will recommend that the file be re-engineered (e.g. re-factored), The recommendation typically includes design and test guidance, automated tests, and the development and test effort to perform the reengineering. The product team development manager and project manager target a release for the file reengineering (e.g. current release, next release, deferred). This may be performed as part of a broader feature level restructuring effort. If the work is not targeted for the current release, establish an interim control program for the file. If the work is completed in the current release, establish a control program for the reeengineered file until its quality is assured.

Project development leaders address all files that have no remaining authors by assigning a file owner to each file with no remaining authors. File owners are SMEs who perform a critical role of quick response to field defects and making or guiding changes to the file. Many projects assign ownership at the directory level. That is, the file owner is responsible for all files in the directory and in any sub-directories. The file owner responsibilities include the following:

1. Educate yourself on the code content of the file and on the file's contribution to product features.

2. If needed, expand the suite of automated tests associated with the file.

3. If other authors change the file, oversee code changes to file, including participation in code inspections and review of test plans.

4. If many other authors change the file, create design and test guidance for the file (see above).

Table 3: Actions Taken by Project B

Refactor Immediately	20%
Refactor in Current Release	20%
Refactor in Next Major Release	3%
Refactor in 2 Releases	17%
Control Content	40%

Table 3 illustrates the types of actions taken by project B.

5. EXPERIENCES

Based on our experience, it is not always easy for an expert to determine the best course of action for a particular risky file. It is important for them to understand the nature of the risk and if it is likely to be realized in the future. To understand the nature of the risk it is critical to not only know why the file was identified as risky (typically the number of past CFDs and the number of authors who left), but also to know what functionality the file implements, what CFDs were fixed in it, when the CFDs occurred, and how central were the authors who left. They also need to know all other MRs, not just CFDs, to determine the best course of action. We, thus, present information in the online evaluation tool as summarized in Table 4. Related files are files that have been identical in the past. As such, related files are similar files in different releases of a product or in different products. The list of related files is important to understand the scope of a defect, and the coordination in change and testing required when modifying a risky file.

The list of file authors is important to identify the subject matter experts, to identify potential design and code reviewers, and to determine a primary audience for design and test guidance, if needed.

In some cases we add information related to the number of static analysis warnings and other information that may be helpful to make one of the decisions as outlined in the Checklist of Suggested Actions section.

To determine if the risky file will cause a defect, experts need to assess the future of the functionality implemented in a file. This typically comes from familiarity with release plans and product strategy. We currently do not present such information in the tool, but we provide some cues. For example, if the latest CFDs were detected more than a year ago, it may indicate that the functionality may have stabilized, while a lot of recent MRs may suggest increased risk of CFDs in the future.

Our approach was refined (and continues to be refined) as we deploy it to new projects. It is important to note that the variation in context: age of the project, number of developers, size of the codebase, number and type of customers

Table 4: Example Data for a Candidate Risky File

Type of Data	Description of Data
List of CFDs	A link to the CFD, the date, and an abstract are provided by the tool to aid the SME in understanding the defect that the file contributed to.
List of Related Files	The name of each related file, last commit date, first commit date, number of commits, and last author to make the commit are identified by the tool. The list is sorted by most recent commit date.
List of File authors	The name, email address, phone number, number of deltas made by the author, and total number of deltas made by the author to all related files are provide by the tool. In addition the first and last date that the author made commits are provided. The list is sorted by number of deltas made by the author.
List of all MRs	A link to the MR, the date of the MR, an MR abstract, and an indication of whether the MR is a CFD is provided by the tool for each MR against the file. The list is sorted by most recent date.
Lines of Code	The size of the file in lines of code are provided as well as the percentage of the size of the file compared to its maximum size.

may affect the types of risks that are the most salient and the kinds of actions that are most appropriate. For example, a risky area with no remaining authors may be gated from changes if no future development is expected, but the best course of action may be to reengineer or completely rewrite the area if considerable development is expected.

6. SUMMARY

We have presented an integrated approach to identify risky areas in the code base of a product and to reduce the risk in those files.

The analytic part of the approach includes tools that utilize commonly available data sources from version control, issue tracking, and personnel directories. The collected measures are used to present a set of candidate risky files in conjunction with additional information to aid subject matter experts in determining the nature and severity of risk in each file. The online tool is augmented by a checklist that suggests the most relevant actions depending on the file characteristics. Finally, we provide training for expert developers to complete the package.

So far we have deployed this approach in 7 projects ranging from 8K to 26M lines of code and involving from 3 to 700 developers over their lifetime. The first project to use this approach is now deployed, and we will assess the impact of this approach by performing a follow-up risky file analysis and by comparing the quality of the deployed release and a prior release of the product.

7. REFERENCES

[1] M. Cataldo, A. Mockus, J. A. Roberts, and J. D. Herbsleb. Software dependencies, the structure of work dependencies and their impact on failures. *IEEE Transactions on Software Engineering*, 2009.

[2] H.-F. Chang and A. Mockus. Constructing universal version history. In *ICSE'06 Workshop on Mining Software Repositories*, pages 76–79, Shanghai, China, May 22-23 2006.

[3] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy. Predicting fault incidence using software change history. *IEEE Transactions on Software Engineering*, 26(2), 2000.

[4] R. Hackbarth, A. Mockus, J. Palframan, and D. Weiss. Assessing the state of software in a large enterprise. *Journal of Empirical Software Engineering*, 10(3):219–249, 2010.

[5] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi. A large-scale empirical study of just-in-time quality assurance. *IEEE Transactions on Software Engineering*, 2013.

[6] A. Mockus. Large-scale code reuse in open source software. In *ICSE'07 Intl. Workshop on Emerging Trends in FLOSS Research and Development*, Minneapolis, Minnesota, May 21 2007.

[7] A. Mockus. Software support tools and experimental work. In V. Basili and et al, editors, *Empirical Software Engineering Issues: Critical Assessments and Future Directions*, volume LNCS 4336, pages 91–99. Springer, 2007.

[8] A. Mockus. Amassing and indexing a large sample of version control systems: towards the census of public source code history. In *6th IEEE Working Conference on Mining Software Repositories*, May 16–17 2009.

[9] A. Mockus. Succession: Measuring transfer of code and developer productivity. In *2009 International Conference on Software Engineering*, Vancouver, CA, May 12–22 2009. ACM Press.

[10] A. Mockus. Organizational volatility and its effects on software defects. In *ACM SIGSOFT / FSE*, pages 117–126, Santa Fe, New Mexico, November 7–11 2010.

[11] A. Mockus. Quantifying and conveying risk in software development. In *1st Symposium on Mining Software Archives*, Monte Verita, Switzerland, March 10-15 2013. Keynote.

[12] A. Mockus, N. Nagappan, and T. Dinh-Trong, Trung. Test coverage and post-verification defects: A multiple case study. In *International Conference on Empirical Software Engineering and Measurement*, Lake Buena Vista, Florida USA, October 2009. ACM.

[13] A. Mockus and D. M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, April–June 2000.

[14] E. Shihab, A. Mockus, Y. Kamei, B. Adams, and A. E. Hassan. High-impact defects: a study of breakage and surprise defects. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, ESEC/FSE '11, pages 300–310, New York, NY, USA, 2011. ACM.