Experiences and Results from Initiating Field Defect Prediction and Product Test Prioritization Efforts at ABB Inc.

Paul Luo Li, James Herbsleb, Mary Shaw
Institute for Software Research, International
Carnegie Mellon University
Pittsburgh, PA
{paul.li,jdh,mary.shaw}@cs.cmu.edu

Brian Robinson
ABB, Inc
Wickliffe, OH
brian.p.robinson@us.abb.com

ABSTRACT

Quantitatively-based risk management can reduce the risks associated with field defects for both software producers and software consumers. In this paper, we report experiences and results from initiating risk-management activities at a large systems development organization. The initiated activities aim to improve product testing (system/integration testing), to improve maintenance resource allocation, and to plan for future process improvements. The experiences we report address practical issues not commonly addressed in research studies: how to select an appropriate modeling method for product testing prioritization and process improvement planning, how to evaluate accuracy of predictions across multiple releases in time, and how to conduct analysis with incomplete information. In addition, we report initial empirical results for two systems with 13 and 15 releases. We present prioritization of configurations to guide product testing, field defect predictions within the first year of deployment to aid maintenance resource allocation, and important predictors across both systems to guide process improvement planning. Our results and experiences are steps towards quantitatively-based risk management.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics – *Complexity metrics, Process metrics, Product metrics*

D.2.9 [**Software Engineering**]: Management – S*oftware quality assurance, Cost estimation*

General Terms

Management, Measurement, Economics, Reliability

Keywords

Deployment and usage metrics, Software and hardware configuration metrics, Software reliability modeling, System test prioritization

1. INTRODUCTION

The US Department of Commerce estimates that software field defects cost the U.S. economy an estimated $59.6 billion annually and that over half of the costs are borne by software consumers and the rest by software producers [29].

ABB is interested in mitigating the risks associated with field defects for itself and for its customers. Software reliability modeling may enable risk mitigation by guiding testing to remove problems before deployment [26], by enabling accurate maintenance resource allocation [18], and by focusing improvement efforts to maximize return on investments [3].

In this paper, we report experiences and initial empirical results from initiating risk management activities at ABB. Both technical results and insights on important practical issues associated with modeling field defects in a real-world setting are reported. Previous studies have reported results from modeling software field defects for risk mitigation (e.g. [31], [30], [13]); however, few studies have reported *experiences on the process* of arriving at such results in an industrial setting.

Our experiences provide insights in the following areas:

- How can a systems development organization select an appropriate modeling method for product testing (systems/integration testing) prioritization and process improvement planning?
- How can a systems development organization evaluate the accuracy of predictions across multiple releases in time?
- How can a systems development organization conduct analysis with incomplete information?

In addition, our technical results address the following empirical research questions:

- How can a systems development organization prioritize configurations (i.e. sub-systems and software platforms) for product testing?
- What modeling method produces the most accurate predictions of the number of field defects in the first year after deployment for maintenance resource planning?
- What categories of predictors (product, development, deployment and usage, or software and hardware configurations) have predictors that may be related to field defects and indicate possible areas for future improvement efforts?

We examine data from two real-time commercial software systems: a monitoring system and a controller management system. We have data from 13 releases spanning ~5 years for the first system and data from 15 releases spanning ~9 years for the second system.

Selecting an appropriate modeling method is an important practical issue because many modeling methods have been proposed/used (e.g. neural networks [15], trees [16], linear modeling [26]); however, research studies have provided little guidance regarding how to select an appropriate method given the goals/objectives of an organization. We find that accuracy, the criterion that research studies use to compare modeling methods, may not be the most important criterion in certain settings. Explicability (the ability to attribute effects to a predictor) and quantifiability (the ability to quantify effects of a predictor) of modeling methods may be more important considerations.

Evaluating accuracy across multiple releases in time is an important practical issue because evaluation techniques used in prior studies are not well suited for today's software systems, which have multiple releases in time that implement additional features. Cross-validation (i.e. leave-one-out cross-validation) and data withholding (i.e. random sub-sampling) are traditional statistical procedures for evaluating accuracy of predictions; however, we find that these procedures may not be adequate.

Dealing with missing/incomplete information is important to practitioners because information is often not available in real-world settings and conducting analysis without important categories of predictors (e.g. deployment and usage predictors for field defect predictions) jeopardizes the validity and accuracy of results; however, methods to deal with missing/incomplete information are not discussed in research studies. We find that by acknowledging incomplete information and collecting data that capture similar ideas as the missing information, we are able to produce more accurate and valid models and motivate better data collection.

We produce empirical research results on product testing prioritization (i.e. which sub-systems and software platforms to focus testing) because few studies have examined product testing (referred to as systems/integration testing in other organizations, explained in section 2). Most studies have focused on lower-level testing (e.g. changes [25], files [31], and modules [14]). However, product testing is sometimes the only place where complex interactions/interface problems can be detected. We prioritize the fault-proneness of software sub-systems and software platforms to guide product testing.

We compare modeling methods because many modeling methods have been used to predict the number of field defects in previous studies; however, few studies have compared the accuracy of predictions (which is the correct criterion when the objective is to predict the number of field defects, explained in section 5.2) in a real-world setting. In this study, we compare methods used in prior work (discussed in section 3.4).

Finally, we evaluate predictors because many different categories of predictors (i.e. metrics available before release) have been examined in the literature for the purposes of predicting/modeling/explaining software defects. Various studies (e.g. [9], [25], [4], [26]) have found each category of predictors (product, development, deployment and usage, and software and hardware configurations) to be important using various techniques (discussed in section 5.1). However, few studies have compared the importance of these categories of predictors simultaneously. In this study, we find that development, deployment and usage, and software and hardware configurations predictors to be important.

Our experiences may help practitioners successfully initiate risk-management activities by providing insights and solutions regarding important practical issues that are not well addressed in prior research studies. The technical results in this experience report provide a point of reference for other researchers.

Section 2 describes improvement initiatives at ABB and the two systems we examined for this study. Section 3 provides context for our technical results. Section 4 describes issues we encountered in the data collection process. Section 5 discusses the issues we encountered in the data analysis process. Section 6 discusses the technical results of our analysis and how the results can help achieve the goals of the improvement initiatives at ABB. Section 7 summarizes the insights gained. We conclude in section 8.

1. SETTING

In our study, we examine two software products, which we will refer to as Product A and Product B, from ABB, a large real-time systems development organization with customers and production facilities around the world. ABB has recently initiated quantitatively-based risk management initiatives for its software operations.

The goals of the activities include:

- Improving testing to remove defects that customers may encounter
- Predicting field defects to enable more accurate maintenance planning
- Planning for future process improvement efforts

Product A is a real-time monitoring system. The core system has a growing code base of approximately 300 KLOC through 13 releases. The project has had approximately 127 thousand changes committed by approximately forty different people. The system has three major add-on packages that provide additional functionality. Field defect data from the add-on packages are included in the data. Product A's change history dates back to 2000.

Product B is a tool suite for managing real-time modules. The system has a stable code base of approximately 780 KLOC through 15 releases. Complete change information and author information are not available because of problems with the version control system (discussed in section 4.3). Based on expert information, approximately fifty people have worked on the project. The project dates back to 1996.

At ABB, *product testing* is testing the integrated software product. This testing process is commonly referred to as system testing or integration testing in the literature. However, ABB has other testing processes that it refers to as systems testing and integration testing.

2. PROJECT CONTEXT

In this section, we present related work in the area of software reliability modeling. The related work provides context for our technical results.

2.1 Type of modeling

In this paper, we establish relationships between characteristics of the two software systems and field defects as well as predict the number of field defects within a time interval.

Field defect modeling in prior work generally belongs to one of four categories:

- Relationships: These studies establish relationships between predictors and the field defects. For example, Harter et al. [3] establish a relationship between an organization's CMM level and the number of field defects in projects completed by the organization.
- Classifications: These studies predict whether the number of field defects is above a threshold for a given observation. For example, Khoshgoftaar et al. [9] classify modules as risky (will contain at least one field defect) or not risky (no field defects) for changed modules.
- Quantities: These studies predict the number of field defects. For example, Khoshgoftaar et al. [14] predict the number of defects for modules of two software systems.
- Rates of occurrences over time: These studies predict the field defect occurrence rate. For example, Kenny [6] predicts the defect occurrence pattern as captured by the Weibull model for two IBM systems.

2.2 Purpose of modeling

We establish relationships (i.e. determine important predictors) in order to prioritize product testing and to plan for future improvements, and we predict quantities to improve maintenance resource planning.

Many previous studies have focused on determining where in the code to test. Few studies (e.g. Mockus et al. [26]) have focused on which software and/or hardware configurations to test. In Mockus et al. [26], the authors analyze customer configurations and determine the defect proneness of different operating systems and product lines (i.e. hardware configurations). The authors suggest that the results can be used to improve product testing.

Predicting the number of field defects within the first year after deployment can enable accurate initial maintenance resource allocation. We predict for one year because budgeting at ABB is performed annually. Furthermore, the first year after deployment contains ~80% of all field defects for Product A and ~90% of all field defects for Product B.

Forecasting field defect rates (which provide information on both the total number of field defects and how the field defects will distributed over time) is more useful than predicting the number of field defects in the first year after deployment [21]. However, field defect rate forecast require fitting/predicting software reliability growth models (SRGMs) as explained in Li et al. [21]. We are not able to successfully fit/predict SRGMs for the systems in this study.

2.3 Categories of predictors

We present results comparing the importance of different categories of predictors of field defects.

Metrics available before release are *predictors,* which can be used by metrics-based modeling methods to predict quantities and to establish relationships. We categorize predictors used in prior studies using an augmented version of the categorization schemes used by Fenton and Pfleeger in [1] and Khoshgoftaar and Allen in [8]:

- Product metrics: metrics that measure the attributes of any intermediate or final product of the software development process. Product metrics have been shown to be important predictors by studies such as Khoshgoftaar et al. [9].
- Development metrics: metrics that measure attributes of the development process. Development metrics have been shown to be important predictors by studies such as Mockus et al. [25].
- Deployment and usage metrics (DU): metrics that measure attributes of deployment of the software system and usage in the field. DU metrics have been shown to be important predictors by studies such as Jones et al. [4].
- Software and hardware configuration metrics (SH): metrics that measure attributes of the software and hardware systems that interact with the software system in the field. SH metrics have been shown to be important predictors by Mockus et al. [26].

Most previous studies have not examined all the categories of predictors simultaneously. An exception is Li et al. [21], which compared the importance of predictors in all the categories simultaneously for an open source operating system. In this paper, we examine the importance of all categories of predictors simultaneously for Product A, but are not able to perform the same analysis for Product B (explained in section 4.2)

2.4 Modeling methods

We compare seven modeling methods from the literature for predicting the number of field defects in the first year after deployment:

1. Moving averages, used in Li et al. [18]
2. Exponential smoothing, used in Li et al. [18]
3. Linear regression with model selection, used in Khoshgoftaar et al. [14] and Khoshgoftaar et al. [11]
4. Clustering, used in Khoshgoftaar et al. [13]
5. Trees, used in Khoshgoftaar and Seliya [16]
6. Neural networks, used in Khoshgoftaar et al. [15] and Khoshgoftaar et al. [14]
7. Ratios, used in Li et al. [19]

These methods have been shown to be effective at predicting the number of field defects in other settings. In this paper, we compare the accuracy of predictions for two commercial software systems.

We discuss selecting a modeling method to guide product testing and to guide process improvement planning in section 5.1.

3. DATA COLLECTION PROCESS

In this section, we discuss our data collection process. We describe the data sources and the metrics we collected from the data sources. We also discuss several issues we encountered in the data collection process.

3.1 Data sources

The data sources we used in our project were a request tracking system, a version control system, and experts.

Serena Tracker is the request tracking system used by ABB. An important feature of Tracker is data entry rules, which allows rules to be constructed and enforced automatically (e.g. reported release field must be filled out). This feature guarantees that key data fields are available across products and releases. More information on Tracker is at [36].

Microsoft Visual Source Safe (VSS) is a version control system used by ABB. VSS has features commonly found in all version control systems (e.g. check-in, check-out, history, etc.). More information is at [24].

Experts are personnel who have extensive experience in a given area. For example, the experts for Product A and Product B have been team leads for the respective products. The expert for Tracker is the lead for the problem tracking and reporting area.

Working with experts allowed us to select the correct information to examine. Without the experts' help, we would have needed a significant amount of time to familiarize ourselves with the data sources. By working with experts to select a subset of product metrics, we ensured that the metrics used were relevant to the organization (discussed in section 4.2.1). This reduction in the number of predictors mitigated the effects of multi-co-linearity and reduced the chances of random correlations. By working with experts to group application areas, we ensured that the groupings corresponded to actual sub-systems (discussed in section 4.2.4).

3.2 Metrics

A *field defect* at ABB is defined as a problem report in Tracker that has been determined to be a valid problem and whose submitted date is after the date of deployment (for a given release). Non-valid problems include problems that are determined to be: not a problem, works as designed, duplicates, or forwarded (i.e. a problem with another product).

3.2.1 Product metrics

We downloaded labeled versions of the source code for Project A from VSS and used the metrics tool Understand for C++ by STI [37] to compute the product metrics. Metrics were calculated for the entire release (e.g. summed across all files in the release).We were not able to download the source code for several releases of Project B due to problems with VSS (explained in section 4.3) and therefore did not include product metrics in the analysis for Project B.

Over 100 product metrics were available using Understand for C++; however, we used 32 in our study. We arrived at the subset of product metrics by working with experts familiar with systems at ABB to select predictors that were relevant (i.e. that might be related to field defects) and that captured each source of variation found in product metrics identified by Munson and Khoshgoftaar in [27].

The 32 product metrics collected are organized by the sources of variation (described in Munson and Khoshgoftaar in [27]) they capture in Table 1. (Some predictors capture multiple sources of variance, but are not displayed in multiple categories) Complete descriptions of the metrics are available from STI [37].

Table 1. Product metrics

Source of variance	*Predictor collected*
Control: control complexity	**Possible normal paths** **Essential complexity** **Knots (overlapping jump statements)** **Max Nesting** **Strict Cyclomatic complexity** **Halstead's Vocabulary**
Volume: size or volume count	**Classes** **Files** **Functions** **Lines of code** **Unique operands** **Total operators** **Total operands** **Halstead's Length** **Halstead's Volume**
Action: distinct operations and statements	**Global Deref Set** **Global Deref Use** **Unique operators** **Global Use** **Global Set** **Global Return** **Global Return**
Effort: mental effort required to generate program	**Halstead's Difficulty** **Halstead's Effort**
Modularity: Depth of syntax tree	**Base classes** **Coupling** **Inheriting classes** **Depth of inheritance tree** **Fanin (incoming calls + global vars read)** **Fanout (calls to others + global vars set)** **InFlow (incoming class)** **OutFlow (calls to others)**

3.2.2 Development metrics

We computed development metrics using change history logs in VSS and problem report information in Tracker. We were not able to collect development metrics from change history logs for Product B due to problems with VSS (explained in section 4.3). We created rough categories of development metrics based on the descriptions of the development metrics in the literature. The predictors we collected covered all but one of the categories.

The groupings of development metrics, examples of development metrics from the literature, and predictors used in our study are in Table 2.

Table 2. Development metrics

Group description	*Examples from literature*	*Predictor collected*
Problems discovered prior to release	number of field problems in the prior release, used in Ostrand et al. [31]	**Targeted issues:** for a release, problems whose fix is going to be in the release **Open issues:** for a release, reported problems that have not been examined at the time of release
Changes to the code	increase in lines of code, used in Khoshgotaar et al. [17]	**Deltas:** changes in the code repository **Added:** lines added **Deleted:** lines deleted **Changed:** lines changed
People in the process	number of updates by designers who had 10 or less total updates in entire company career, both used in Khoshgoftaar et al. [17]	**Authors:** number of different authors making changes to the code **Bottom Half Authors:** changes by authors who ranked in the bottom 50% in terms of the total number of changes to the project
Process efficiency	CMM level, used in Harter et al. [3]	NONE COLLECTED

3.2.3 Deployment and usage metrics

ABB did not officially collect deployment and usage information (e.g. the number of installations) for the two systems. Despite this limitation, by talking to experts, we were able to collect data from available data sources that we felt provided information on possible deployment and usage of the systems. We used the data with the specific aim of motivating better data collection: if the deployment and usage predictors we collected were shown to be statistically significant, then better deployment and usage information might be justified. We considered the type of release and the amounts of elapsed time between releases. Both kinds of predictors might give an indication of the number of users exercising a release (and thus the amount of usage). For example, a major release might have more users than a minor release. A release that was in the field for a longer period of time before the next release might have more users than a release was in the field for a shorter period of time before the next release.

The deployment and usage metrics we collected are in Table 3.

Table 3. Deployment and usage metrics

Predictor collected	*Description*
Service pack	If the release is a major release, a minor release, or a service pack
Months since 1st released	Months since the system was first released
Moths since previous release	Months since the last release (service pack, minor release, or major release)
Months until next release	Months before the next release (service pack, minor release, or major release)

3.2.4 Software and hardware configuration metrics

Information on software and hardware configurations in use was not available at the systems level (e.g. the number of systems with a specific configuration or the number of systems using an operating system). Information was available at the individual defect level. Problem reports had data fields describing the application, the operating system, and the Internet Explorer version (for Product A only) associated with the defect.

We worked with experts to group applications into application groups (i.e. sub-systems) and to group operating systems into operating systems groups (i.e. software platforms) for product testing prioritization. Both Product A and Product B ran under versions of the Windows operating system. Product A also used versions of Internet Explorer.

The software and hardware configurations metrics we collected are in Table 4.

Table 4. Software and hardware configuration metrics

Predictor collected	*Description*
Sub-system	Groups of applications (functionality)
Operating system	Groups of versions of the Windows operating system
Internet Explorer version (Product A only)	Groups of versions of the Internet Explorer

3.3 Issues during data collection

In this section we briefly discuss some of the issues we encountered during data collection. The issues are divided into two categories: tools and process.

3.3.1 Tools

There was one technical problem with VSS. At least one file in VSS was corrupted for Product B. VSS reported an error getting the history of the file. This error prevented us from collecting some development predictors and all product predictors for Product B.

Problems with retrieving historical information from version control systems were not isolated to Microsoft VSS. Problems retrieving information from open source version control systems had been reported in Li et al. [20].

After talking to experts, we decided to manually by-pass the corrupt file (i.e. creating batch scripts to enumerate the files to examine). This process had been planned but had not been implemented.

3.3.2 Process

There were three problems with how Tracker and VSS were used. Neither Product A nor Product B recorded the phase of development in which a defect was discovered. Labels (in VSS) were not used correctly in Product A. Also, labels were not used at all in Product B.

First, the problem reporting form did not include a field for the phase found (e.g. unit testing, product testing, or field); therefore, the number of development defects metric was not available. We were able to use the time of release and build number to determine if a defect is a field defect. However, we could not determine the number of development defects because we were not able to attribute a non-field defect to a particular release. The development periods of service packs and major releases often coincided and no other information was available to distinguish between releases.

We used a combination of targeted changes and open issues instead. Targeted changes counted problems found prior to release that were earmarked to be fixed in the current release; however, targeted changes did not count problems found during development that were deferred to later releases. Open issues counted reported problems that had not been reviewed at the time of release; however, open issues included problems that were not found during development (e.g. problems from previous releases).

Secondly, neither project used labels correctly. We double counted some development information for Product A (e.g. attributed changes to both releases when computing development metrics) because Product A did not have separate development branches for major releases, minor releases, and service packs. The labels increased incrementally. Sub-labels (e.g. x.x for major releases, x.x.x for service packs) were not used. Some service packs were released very close in time to major releases and their developments overlapped. After talking to experts we concluded that all development information since the previous major release should be counted as development information for both the next major release and service packs in the interim.

Finally, Product B did not use labels. The development team decided that the labeling system in VSS was not flexible enough. The team often wanted to update only one file; however VSS did not allow single file updates in a label (i.e. all files in the repository must be relabeled even if only one file had been changed since the last label). The development team decided that the re-labeling process was too time consuming and elected not to use labels. Only deployment and usage metrics, development metrics from Tracker, and software and hardware configurations metrics from Tracker were available for Product B.

4. DATA ANALYSIS PROCESS

In this section we describe our analysis process. We discuss the methods we used to identify important predictors of field defects and procedures we used to evaluate predicted numbers of field defects within the first year after deployment.

All analysis was performed using the open source analysis package R [32].

4.1 Important predictor identification for product testing prioritization and improvement planning

In order to identify fault-prone configurations for product testing prioritization and important characteristics of the software system for improvement planning, we need to identify important predictors. Three methods are commonly used to establish a predictor as important in the literature:

1. Show high correlation between the predictor and field defects. This method is recommended by IEEE [6].
2. Show that the predictor is selected using a modeling method with selection. This method is used by Mockus et al. [26].
3. Show that the accuracy of predictions improves with the predictor included in the prediction model. This method is used by Jones et al. [4].

We used methods 1 and 2 to determine important predictors in our study. Due to data constraints, we did not use method 3.

We used rank correlation in our analysis to identify important predictors to guide process improvement planning. If there was a strong statistical relationship between a characteristic of the software system (i.e. a predictor) and the number of field defects, then it might be possible to use the information to improve quality.

We use rank correlation because our data are not normally distributed and because we want to minimize the effects of observations that are far away from the mean [4]. Rank correlation is ideal for establishing the importance of a single predictor. However, some predictors capture the same source of variation and are highly correlated with each other (e.g. lines of code and Cyclomatic complexity [2]). In addition, multiple sources of variation may need to be considered and accounted for.

We used general linear modeling methods with model selection, which identified predictors that complemented each other (i.e. capture different source of variation) for identifying fault prone configurations to guide product testing and for identifying important predictors to guide process improvement.

We used BIC (Bayesian Information Criterion) model selection to determine important areas for product testing (using Poisson regression) and to determine important predictors (using Linear regression). BIC is better suited to select predictors to explain causation [38] (where as Akaike Information Criterion (AIC) is better suited for selecting predictors for prediction).

Many modeling methods were available to analyze the data; however, not all modeling methods were appropriate given our objectives. In order to prioritize product testing we needed a modeling method to have explicability (i.e. being able to identify the faulty configuration) and quantifiability (i.e. being able to quantify the fault-proneness of one configuration relative to another). In order to plan for process improvement we also needed to have explicability (i.e. being able to identify the predictors that may be related field defects) and quantifiability (i.e. being able to quantify the relative importance of the predictors). Previous studies focused on using accuracy to select a modeling method; however, accuracy was not the main consideration for identifying important predictors for product testing prioritization and improvement planning.

We elected to use general linear modeling methods (Poisson regression and Linear regression) because they clearly identified the important predictors and quantified the relative importance of the predictors. Moving averages and exponential smoothing used in Li et al. [18] did not use predictors. Neural networks used by Khoshgoftaar et al. in [14], discriminant analysis with principal component analysis used by Khoshgoftaar et al. in [14], and clustering used by Khoshgoftaar et al. in [13] obfuscated the effects of an individual predictor by combining predictors. The ratios method used by Li et al. in [19] required choosing a predictor a-priori and did not allow comparison of the importance of the predictors. The trees method used in Khoshgoftaar and Seliya [16] could not be used to compare the relative importance of different predictors.

We used a four step process to prioritize product testing. First we elicited application groupings (i.e. sub-systems) and operating systems groupings from the experts. Then, as validation, we asked the experts to name the most defect-prone sub-systems based on their experience. This set of sub-systems served as a possible target for our analysis (i.e. the quality of our analysis depends on our ability to identify these defect-prone sub-systems). We included predictors that captured differences between releases (i.e. confounds). In our analysis, all the predictors in section 4.2 were included in the analysis as confounds. Finally, after we had determined the fault proneness of different areas, we validated the results by asking experts if the results matched their experiences.

4.2 Field defect prediction for maintenance resource allocation

Since ABB allocates maintenance resources based on the number of expected field defects, in order to enable accurate initial maintenance resource allocation, we need to accurately predict the number of field defects within the first year after release. Given this objective, accuracy is the most important consideration.

We used a forward prediction evaluation procedure to analyze the accuracy of predictions of the metrics-based modeling methods. Previous studies used either cross-validation [33] or random data with-holding [5]; however, neither procedure is realistic. In a realistic setting, only a non-random subset of data is actually available before release (i.e. only data from historical releases). Secondly, both techniques assumed that predicting for a past observation was the same as predicting for a future observation; however, the software systems and software development organizations were constantly evolving. We used a selective data withholding procedure: forward prediction evaluation. For each software release in time, we withheld all information not available at the time of release. We then used only information available at the time of release to predict for the next release. We evaluated the prediction for that release only. We repeated this process for each release in time. This technique better accounted for changes in time and gave a more realistic assessment of the accuracy of the modeling methods.

For confidentiality reasons, we evaluate each individual prediction using absolute relative error. Absolute relative error is defined as the sum over all observations, of the absolute value of the difference between the predicted value and the actual value divided by the actual value.

$\tilde{y}_i$ = predicted number of field problems for observation i

y_i = actual number of field problems for observation i

n = total number of observations

$$\text{ARE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{\overline{y}_i - y_i}{y_i}\right|$$

5. TECHNICAL RESULTS

In this section, we present the results of our analysis and how the results can help achieve the goals of ABB's initiatives. First, we present the relative fault-proneness of different sub-systems and software platforms to guide product testing. Then, we present the accuracy of predictions of the number of field defects within the first year after deployment to aid maintenance planning. Finally, we present the important predictors of field defects to guide future improvements.

5.1 Product testing prioritization

We present the results of our Poisson regression, where we regressed predictors described in section 4.2 on the number of field defects. Due to confidentiality restrictions, we do not identify the sub-systems or give exact quantities.

We present the four most defect-prone sub-systems (labeled 1-4) and their field defect tendency relative to the least defect-prone sub-system for Product A and Product B (e.g. sub-system 1 will experience ~9.85x more field defects than the least defect-prone sub-system). There are 38 sub-systems for Product A and 19 sub-systems for Product B. We present the faultiest Windows configuration relative to the safest Windows configuration. There are only two Windows configurations for both Product A and Product B. Internet Explorer version is not selected as a significant predictor. Other predictors selected (i.e. confounds) and their effects are indicated. Each effect is defect-proneness relative to a single unit increase in the predictor's value (e.g. for Product A an additional author making changes for a release will decrease the number of field defects observed by ~7%).

The results in table 5 and 6 present the effect of each predictor after accounting for the effects of the other predictors in the regression. Our results are more accurate than the results of simpler methods (e.g. comparing counts) that do not account for differences between releases (i.e. confounds. The first column lists the sub-systems, software platforms, and confounds. The second column indicates their relative fault-proneness.

Table 5. System test prioritization for Product A

Product A Predictors	*Estimated Effect*
Sub-system:	
Sub-system 1	9.85x
Sub-system 2	8.39x
Sub-system 3	8.13x
Sub-system 4	7.22x
Software platforms:	
Not Windows Server Versions	1.91x
Confounds:	
Service Pack	.18x
Open Issues	.999x
Num Authors	.93x
Months Before Next Release	1.16x
Months Since 1st Release	.97x

Table 6. System test prioritization for Product B

Product B Predictors	*Estimated Effect*
Sub-system:	
Sub-system 1	8.21x
Sub-system 2	5.56x
Sub-system 3	4.52x
Sub-system 4	3.68x
Software platforms:	
Win95/98	3.00x
Confounds	
Open Issues	.995x
Months Before Next Release	1.07x
Months Since 1st Release	1.02x

The sub-systems identified by experts as the most defect-prone are among the top 4 identified in our analysis. The experts also corroborate our results by concluding that each of the most defect-prone sub-systems identified is trouble-some based on past experiences.

Our analysis is not intended to replace expert knowledge. Our results complement expert intuition by providing quantitative evidence. This evidence will allow test engineers to back their decisions and recommendation with quantitative evidence. Our results also preserve knowledge. Experts may leave the company or may be promoted to other positions. Our process can transfer expert knowledge to other test engineers.

Using the results, ABB can better allocate testing resources. The product test team for Product A has used the product test prioritization results to uncover additional defects in a sub-system previously thought to be low-defect.

5.2 Field defect predictions

We present the results of our predictions using 16 modeling methods and our forward prediction evaluation procedure. We are able to predict within ~24.6% ARE over three releases of Product A and within ~77.1% ARE over six releases of Product B for only major/minor releases (i.e. no service packs). The results for Product A are in Table 7, and the results for Product B are in Table 8. The three most accurate methods for each release are bolded.

We find that we are not able to predict accurately for service packs (SP), minor releases, and major releases together. The most accurate method for Product A (the clustering method) produces predictions with ~64.3% relative error for 4 releases. Of the methods that predicted for the most number of releases, the best method (Moving Averages of 1 Release) produces predictions with ~469.8% relative error. The most accurate method for Product B (Moving Average of 3 Releases) produces predictions with ~179.4% relative error for 10 releases. Of the method that predicted for the most number of releases, the best method (Neural Networks) produces predictions with ~213.8% relative error.

Table 7. ARE of predicted field defects for Product A

ARE Product A	*R1.0*	*R1.1*	*R1.2*	*Avg*
Moving Average 1 Release	**3.0%**	**51.7%**	**19.2%**	24.6%
Moving Average 2 Releases		54.0%	50.0%	52.0%
Moving Average 3 Releases			37.8%	37.8%
Exponential Smoothing 2 Releases		53.6%	44.6%	49.1%
Exponential Smoothing 3 Releases			53.9%	53.9%
Linear Regression with Model Selection			**17.6%**	17.6%
Clustering			50.0%	50.0%
Tree Split with 2 Releases	**3.0%**	**51.7%**	**19.2%**	24.6%
Tree Split with 3 Releases	**3.0%**	54.0%	**19.2%**	25.4%
Tree Split with 4 Releases	**3.0%**	54.0%	62.1%	39.7%
Neural Network	**2.3%**	52.3%	53.9%	36.2%
Ratios	51.4%	**48.4%**	22.0%	40.6%

Table 8. ARE of predicted field defects for Product B

ARE Product B	*R2.0*	*R3.0*	*R3.1*	*R3.2*	*R4.0*	*R4.1*	*Avg*
Moving Average 1 Release	**61.9%**	**17.6%**	**65.3%**	1628.6%	332.1%	27.3%	662.7%
Moving Average 2 Releases		31.4%	71.1%	1128.6%	207.1%	20.5%	452.1%
Moving Average 3 Releases		**9.8%**	**62.0%**	919.0%	154.8%	136.4%	403.4%
Exponential Smoothing 2 Releases		29.0%	70.1%	1216.8%	229.2%	**12.0%**	486.0%
Exponential Smoothing 3 Releases						410.7%	410.7%
Linear Regression with Model Selection		46.2%	70.6%	**785.7%**	**121.4%**	118.8%	342.0%
Clustering		31.4%	71.1%	**564.3%**	**66.1%**	45.5%	225.3%
Tree Split with 2 Releases	**61.9%**	**17.6%**	**65.3%**	1628.6%	332.1%	27.3%	662.7%
Tree Split with 3 Releases	**61.9%**	31.4%	71.1%	1628.6%	332.1%	27.3%	662.7%
Tree Split with 4 Releases	**61.9%**	**9.8%**	**62.0%**	1628.6%	332.1%	**4.5%**	655.1%
Neural Network	**61.0%**	32.4%	70.4%	1628.6%	332.1%	45.7%	668.8%
Ratios	**36.9%**	136.7%	70.9%	**160.2%**	**62.6%**	**8.4%**	77.1%

The poor accuracy of predictions for Product B is mostly caused by one release (Release 3.2). Release 3.2 has ~85% fewer field defects than the average release. An average release has ~6.4x more field defects. None of the modeling methods we examined is able to predict accurately for this release.

We conjecture that data limitations may have contributed to the poor results. There are only 4 major and minor releases of Product A and 8 releases of Product B. Other research studies examine modules or individual installations and have significantly more observations. For example, the study by Khoshgoftaar et al. [16] uses over 500 observations and the study by Mockus et al. [26] uses over 1000 observations. Limited information may have reduced effectiveness of complex metrics based methods and the benefits of metrics in many categories as described by Li et al. [21].

Using the results, ABB can allocate initial maintenance resources. If the estimate costs of resolving a field defect are available, then the predicted number of field defects can be combined with the estimated costs to project the amount of maintenance resources needed for the first year after deployment. ABB will need to evaluate if the error in predictions (the ~25% ARE for Product A and the ~77% ARE for Product B) are tolerable.

5.3 Plans for improvement

We present results of our analysis using rank-correlation and linear modeling with model selection.

The top three predictors of field defects selected using Spearman rank correlation for Product A and Product B are in Table 9. These predictors are more statistically important than the other predictors.

Table 9. Spearman rank correlated predictors

Product A Predictors	Correlation	P-Value
Open Issues	0.770	0.021
Target Changes	0.803	0.015
Service Pack	-0.520	0.162
Product B Predictors	Correlation	P-Value
Months Since 1st Release	-0.683	0.009
Open Issues	-0.643	0.015
Target Changes	0.424	0.130

The top two selected predictors using BIC model selection for each product are in Table 10. These predictors are statistically significant and capture different sources of variance.

Table 10. BIC selected predictors

Product A Predictors
Open Issues
Service Pack
Product B Predictors
Open Issues
Months Before Next Release

We have identified development, deployment and usage, and software and hardware configurations metrics as common factors that are related to field defects across two products in the same organization for process improvement planning.

We find that deployment and usages factors (Service Packs for Product A and Months Before Next Release for Product B) and software and hardware configuration metrics are important predictors. Identifying these two predictors to be important motivates collecting better deployment and usage information.

We find that two development factors (Targeted Changes and Open Issues) are highly correlated to field defects across both products at ABB using rank correlation. Identifying these two predictors to be important motivates collecting better development defect information (discussed in 4.2.3).

These two predictors also identify areas where the development process can be improved. For example, the Open Issues metric, which measures the number of reported problems that have not been examined at the time of release (see section 4.2.3), can be used to better manage releases. ABB can use the results in two ways. First, it can change the development/deployment process to delay deployment of Product A when there are large numbers of unexamined problems at the time of release. Second, if there are large numbers of Open Issues, then ABB can reduce the scope of the next release of Product A and allocating enough time and resources to resolve field defects.

6. LESSONS LEARNED

In this section, we summarize the insights we gained regarding the practical issues of selecting an appropriate modeling method for product testing prioritization and process improvement planning, evaluating accuracy of predictions across multiple releases in time, and conducting analysis with incomplete information.

6.1 Selecting an appropriate modeling method

We have found that identifying areas for improvement and prioritizing product testing both required attributing effects to specific predictors (explicability) and being able to weigh the relative importance of different predictors (quanitifiability). Given our objectives, explicability and quanitifiability are more appropriate criterions than accuracy for selecting a modeling method. Given the criterions, we elect to use linear modeling with model selection in our study.

Other researchers have noted problems with using accuracy as the criterion for selecting a modeling method (e.g. Shepperd in [35]); however, we are among the few to identify specific situations in which accuracy is not the most appropriate criterion and the modeling method to use given the situations.

6.2 Evaluating accuracy of predictions across multiple releases in time

We have found that the forward prediction evaluation procedure gives a more realistic assessment of the accuracy of prediction of modeling methods for multiple releases in time.

From talking with experts and examining change logs we concluded that the software systems and the software development organizations for both Product A and Product B changed during the observational periods (13 releases between 2000 and 2005 for Product A and 15 releases between 1996 and 2005 for Product B). The evaluation procedures used in previous studies assumed that time did not matter and did not consider change in the software system and the development organization. Our forward prediction evaluation procedure accounted for this evolution process.

Our forward prediction evaluation procedure is taken from time-series analysis; however, our application to software field defect prediction evaluation is novel.

6.3 Conducting analysis with incomplete and missing information

We have found that by acknowledging data deficiencies and using available information, we are able to find rationale for better data collection and produce useful analysis results.

For example, we were not able to collect exact deployment and usage; however, we used available information to derive rough measures. Although not perfect, the rough measures enabled more accurate predictions (since the measures were selected as important predictors). By showing that the rough measures were important, we might be able to make an effective case for better data collection.

7. CONCLUSION

In this paper, we have reported experiences and empirical results from initiating quantitatively-based risk management activities. We have presented technical results prioritizing product testing, comparing modeling methods for field defect predictions, and planning for process improvement efforts. In addition, we have reported experiences on selecting a modeling method when explicability and quantifiability are important, selecting an evaluation technique when time is a factor, and conducting analysis with incomplete data.

Our study contains preliminary findings. We have not fully gauged the adequacy of our results. For example, we do not know if predictions with ~25% ARE for product A and ~77% ARE for product B are accurate enough to aid maintenance resource allocation. However, initial feedback from experts has been positive, initial usage of the results (discussed in section 6.1) has been promising, and the results have contributed to buy-in for process improvement activities.

More work is necessary to derive benefits for ABB. We need to investigate how current product testing is conducted. We need to complete data collection and analysis for Product B. We need to expand our analysis to more projects within ABB. Finally, we need to investigate the results in more detail. For example, we need to determine why the open issues metric has a positive relationship to field defects for Product A but a negative relationship to field defects for Product B.

Improved testing, better maintenance resource allocation, and better process improvement planning can lower the risk of field defects for software consumers and software producers. The results in this paper are steps towards achieving that goal using quantitatively-based methods.

8. ACKNOWLEDGEMENT

This research was supported by the National Science Foundation under grants ITR-0086003 and CCF-0438929, and by the Carnegie Mellon Sloan Software Center. We would like to thank Anne Poorman, Janet Kaufman, Patrick Weckerly, and Rob Davenport for their input and expert knowledge.

9. REFERENCES

[1] Norman Fenton and Shari Lawrence Pfleeger. *Software metrics (2nd ed.): a rigorous and practical approach.* Boston, MA: PWS Publishing Co., 1997.

[2] Norman Fenton and Martin Neil. Software metrics: road map. *Proc. ICSE*, May 2000, pp. 357-370.

[3] Donald E. Harter and Mayuram S. Krishnan and Sandra A. Slaughter. Effects of Process Maturity on Quality, Cycle Time, and Effort in Software Product Development. *Management Science,* vol. 46 no. 4, Apr 2000, pp. 451-466.

[4] Myles Hollander and Douglas A. Wolfe. *Nonparametric statistical inference*. New York, NY: Wiley & Sons, 1973.

[5] Wendell Jones, John Hudepohl, Taghi Khoshgoftaar, and Edward Allen. Applications of a Usage Profile in Software Quality Models. *Proc. 3rd European Conference on Software Maintenance and Reengineering*, Mar 1999, pp. 148-157.

[6] IEEE standard for a software quality metrics methodology. *IEEE Std 1061-1998*, Dec 1998.

[7] Garrison Kenny. Estimating Defects in Commercial Software during Operational Use. *IEEE Tr. on Reliability,* vol. 42 no. 1, Mar 1993, pp. 107-115.

[8] Taghi M. Khoshgoftaar and Edward B. Allen. Predicting fault-prone software modules in embedded systems with classification trees. *Proc. HASE*, Nov 1999, pp. 105-112.

[9] Taghi Khoshgoftaar, Edward Allen, and Jianyu Deng. Controlling Over-fitting in Software Quality Models: Experiments with Regression Trees and Classification. *Proc. METRICS,* Apr 2001, pp. 190-198.

[10] Taghi M. Khoshgoftaar and Edward B. Allen and John P. Hudepohl and Stephen J. Aud. Application of Neural Networks to Software Quality Modeling of a Very Large Telecommunications System. *IEEE Tr. on Neural Networks*, vol. 8 no. 4, Jul 1997, pp. 902-909.

[11] Taghi Khoshgoftaar, Bibhuti Bhattacharyya, and Gary Richardson. Predicting Software Errors, During Development, Using Nonlinear Regression Models: A Comparative Study. *IEEE Tr. On Reliability,* vol. 41 no. 3, Sep 1992, pp. 390-395.

[12] Taghi Khoshgoftaar and John Munson. Predicting Software Development Errors using Software Complexity Metrics. *IEEE J. Selected Areas in Communications,* vol. 8 no. 2, Feb 1990, pp. 253-261.

[13] Taghi Khoshgoftaar, John Munson, and David Lanning. A Comparative Study of Predictive Models for Program Changes during System Testing and Maintenance. *Proc. ICSM,* Sep 1993, pp. 72-79.

[14] Taghi Khoshgoftaar, Abhijit Pandya, and David Lanning. Application of Neural Networks for Predicting Program Fault. *Annals of Software Engineering,* vol. 1, 1995, pp. 141-154.

[15] Taghi Khoshgoftaar, Abhijit Pandya, and Hamant More. A Neural Networks Approach for Predicting Software Development Faults. *Proc. ISSRE,* Oct 1992, pp. 83-89.

[16] Taghi Khoshgoftaar and Naeem Seliya. Tree-based Software Quality Estimation Models for Fault Prediction. *Proc. METRICS,* Jun 2002, pp. 203-214.

[17] Taghi Khoshgoftaar, Vishal Thaker, and Edward Allen. Modeling Fault-prone Modules of Subsystems. *Proc. ISSRE*, Oct 2000, pp. 259-267.

[18] Paul Luo Li, Mary Shaw, Jim Herbsleb, Bonnie Ray, and P. Santhanam. Empirical Evaluation of Defect Projection Models for Widely-deployed Production Software Systems. *Proc. FSE,* vol. 29 no. 6, Oct 2004, pp.263-272.

[19] Paul Luo Li, Mary Shaw, Jim Herbsleb, Bonnie Ray, and P. Santhanam. An Empirical Comparison of Field Defect Modeling Methods, *CMU Tech Report CMU-ISRI-06-102,* 2006

[20] Paul Luo Li, Jim Herbsleb, and Mary Shaw. Finding Predictors of Field Defects for Open Source Software Systems in Commonly Available Data Sources: a Case Study of OpenBSD. *Proc. METRICS,* Sep 2005, (to appear).

[21] Paul Luo Li, Jim Herbsleb, and Mary Shaw. ForecastingField Defects Using a Combined Time-based and Metrics-based Approach: a Case Study of OpenBSD. *CMUTechReport, CMU-ISRI-05-125,* 2005.

[22] Zhaohui Liu, Nalini Ravishanker, and Bonnie Ray. Modeling Dynamic Reliability Growth Using Bayesian Methods. *Reliability Review*, vol. 23 no. 1, Mar 2003, pp. 5-9.

[23] Michael Lyu. *Handbook of Software Reliability Engineering*. McGraw-Hill, 1996.

[24] Microsoft Visual SourceSafe. msdn.microsoft.com/ssafe/

[25] Audris Mockus, David Weiss, and Ping Zhang. Understanding and Predicting Effort in Software Projects. *Proc. ICSE,* May 2003, pp. 274-284.

[26] Audris Mockus, Ping Zhang, and Paul Luo Li. Predictors of Customer Perceived Quality. *Proc. ICSE,* May 2005, pp. 225-233.

[27] John Munson and Taghi Khoshgoftaar. The Dimensionality of Program Complexity. *Proc. ICSE,* May 1989, pp. 245-253.

[28] John Musa and Anthony Iannino and Kazuhira Okumoto. *Software Reliability*. McGraw-Hill, 1990.

[29] National Institute of Standards and Technology. *The economic impacts of inadequate infrastructure for software testing.* Planning Report 02-3, Jun 2002

[30] Magnus Ohlsson and Per Runeson. Experience from Replicating Empirical Studies on Prediction Models. *Proc. METRICS,* Jun 2002, pp. 217-226.

[31] Thomas Ostrand, Elaine Weyuker, and Thomas Bell. Where the Bugs are. *Proc. ISSTA,* vol. 29 no. 4, Jul 2004, pp. 86-96.

[32] The R project for statistical computing. www.r-project.org

[33] Richard Selby and Adam Porter. Learning from examples: generation and evaluation of decision trees for software resource analysis. *IEEE Tr. On Software Engineering*, vol 14. no. 12, Dec 1988, pp. 1743-1757.

[34] Norman F. Schneidewind. Body of Knowledge for Software Quality Measurement. *IEEE Computer*, vol. 35 no. 2, Feb 2002, pp. 77-83.

[35] Martin Sheppard. Evaluating Software Project Prediction Systems. *METRICS,* Keynote speech, 2005.

[36] Tracker. www.serena.com/Products/professional/tracker/

[37] Understand for C++. www.scitools.com/metrics.txt

[38] Sanford Weisberg. *Applied Linear Regression, 2nd Edition.* New York, NY: Wiley and Son, 1985.