Cross-project Defect Prediction

A Large Scale Experiment on Data vs. Domain vs. Process

Thomas Zimmermann
Microsoft Research
tzimmer@microsoft.com

Nachiappan Nagappan
Microsoft Research
nachin@microsoft.com

Harald Gall
University of Zurich
gall@ifi.uzh.ch

Emanuel Giger
University of Zurich
giger@ifi.uzh.ch

Brendan Murphy
Microsoft Research
bmurphy@microsoft.com

ABSTRACT

Prediction of software defects works well within projects as long as there is a sufficient amount of data available to train any models. However, this is rarely the case for new software projects and for many companies. So far, only a few have studies focused on transferring prediction models from one project to another. In this paper, we study cross-project defect prediction models on a large scale. For 12 real-world applications, we ran 622 cross-project predictions. Our results indicate that cross-project prediction is a serious challenge, i.e., simply using models from projects in the same domain or with the same process does not lead to accurate predictions. To help software engineers choose models wisely, we identified factors that do influence the success of cross-project predictions. We also derived decision trees that can provide early estimates for precision, recall, and accuracy before a prediction is attempted.

Categories and Subject Descriptors. D.2.8 [Software Engineering]: Metrics—*Performance measures, Process metrics, Product metrics.* D.2.9 [Software Engineering]: Management—*Software quality assurance (SQA)*

General Terms. Management, Measurement, Reliability.

1. INTRODUCTION

Defect prediction works well if models are trained with a sufficiently large amount of data and applied to a single software project [26]. In practice, however, training data is often not available, either because a company is too small or it is the first release of a product, for which no past data exists. Making automated predictions is impossible in these situations. In effort estimation when no or little data is available, engineers often use data from other projects or companies [16]. Ideally the same scenario would be possible for defect prediction as well and engineers would take a model from another project to successfully predict defects in their own project; we call this *cross-project defect prediction*. However, there has been only little evidence that defect prediction works across projects [32]—in this paper, we will systematically investigate when cross-project defect prediction does work.

The specific questions that we address are:

1. To what extent can we use cross-project data to predict post-release defects for a software system?
2. What kinds of software systems are good cross-project predictors—projects of the same domain, or with the same process, or with similar code structure, or of the same company?

Considering that within companies, the process is often similar or even the same, we seek conclusions about which characteristics facilitate cross-project predictions better—is it the same domain or the same process?

To test our hypotheses we conducted a large scale experiment on several versions of open source systems from Apache Tomcat, Apache Derby, Eclipse, Firefox as well as seven commercial systems from Microsoft, namely Direct-X, IIS, Printing, Windows Clustering, Windows File system, SQL Server 2005 and Windows Kernel. For each system we collected code measures, domain and process metrics, and defects and built a defect prediction model based on logistic regression. Next we ran 622 cross-projects experiments and recorded the outcome of the predictions, which we then correlated with similarities between the projects. To describe similarities we used 40 characteristics: code metrics, ranging from churn [23] (i.e., added, deleted, and changed lines) to complexity; domain metrics ranging from operational domain, same company, etc; process metrics spanning distributed development, the use of static analysis tools, etc. Finally, we analyzed the effect of the various characteristics on prediction quality with decision trees.

1.1 Contributions

The main contributions of our paper are threefold:

1. Evidence that it is not obvious which cross-prediction models work. Using projects in the same domain does not help build accurate prediction models. Process, code data and domain need to be quantified, understood and evaluated before prediction models are built and used.
2. An approach to highlight significant predictors and the factors that aid building cross-project predictors, validated in a study of 12 commercial and open source projects.
3. A list of factors that software engineers should evaluate before selecting the projects that they use to build cross-project predictors.

ESEC/FSE'09, August 24–28, 2009, Amsterdam, The Netherlands.

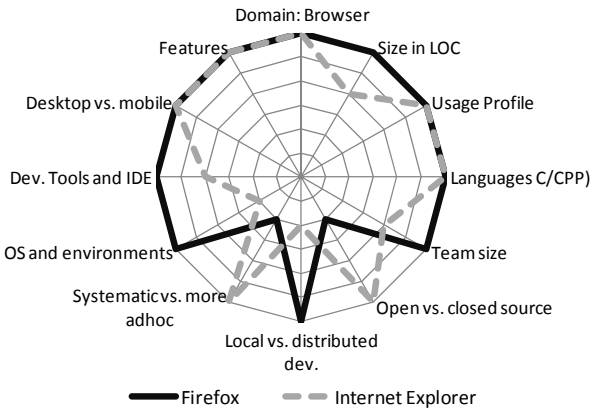

Figure 1. Comparing characteristics of Firefox and Internet Explorer.

The remainder of the paper is structured as follows: In Section 2 we motivate this paper with a cross-project prediction between Firefox and Internet Explorer. We then describe the data collection in Section 3 and the experiment in Section 4. We discuss cross-project predictability in Section 5. We conclude this paper with threats to validity (Section 6), related work (Section 7) and with consequences and ideas for future work (Section 8).

2. A STORY OF FIREFOX AND IE

As with any prediction model, defect prediction for software systems works only if enough adequate training data is available to initially feed the model. Often such straightforward data is not available. Missing training data is the initial incentive for our research question in this paper: "Can engineers use data from others projects to successfully predict defects in a different software system?" This situation is especially interesting if no post-release data exists within an organization.

In this case, we investigated whether Firefox data can be used to build a reliable prediction model for Internet Explorer (IE) and vice versa. Cross-project prediction manifests in two dimensions: the domain dimension, and the company dimension. This experiment falls under the category "same domain (i.e., web browser) but different companies (i.e., Microsoft and Mozilla Foundation)". The idea was to investigate whether the similarities of the web browser domain overrule the different development processes and corporate cultures, and whether Firefox and Internet Explorer still can be used to predict each other's defects.

In Figure 1, we use a radar chart to better illustrate similarities and dissimilarities of the individual characteristics we use a radar chart. Similar characteristics (such as *size in LOC* or *language*) have the same distance from the center point, while dissimilar characteristics have different distances (such as *OS* and *team size*).

In terms of the requirements of the web browser ***domain***, Firefox and Internet Explorer are rather similar. Both have components for DOM tree generation and manipulation, rendering, HTML and CSS parsing, protocol machines for HTTP, local caches, download monitoring, security aspects, navigation and history management, UI processes, frames, toolbars and menus, scripting engines such as JavaScript, XML, and graphics including SVG.

Firefox and Internet Explorer are also similar with respect to their additional *features* facilitating browsing the internet: tabbed browsing; pop-up blocking; a download manager; live bookmarks (RSS and other feeds); add-ons, user-defined extensions, themes, language packs or plug-ins; web technology support including HTML, XML, XHTML, CSS, JavaScript, DOM, MathML, SVG, XSL and XPath; micro summaries; security including sandboxing, same origin policy, external protocol white listing, a phishing detector, private data clearance, and malware detection.

When it comes to the ***development process***, Firefox and Internet Explorer are significantly different from each other: Dissimilarities can be found in the way the systems are being developed (*open vs. closed source*), the *degree of distribution* of teams (local in Redmond, USA vs. global), the *process that is used* (systematic large scale vs. more ad-hoc and agile development), different *operating systems and environments* (Windows vs. Linux, Mac, and Windows), and *tools and IDEs* used for development

We collected code churn [23] (i.e., added, deleted, and changed lines), complexity and pre-release bug metrics to build models (using logistic regression) on Firefox to predict Internet Explorer defects and vice versa. The results showed that Firefox data can predict Internet Explorer defects very well with a precision of 76.47% (how many of the binaries predicted as defect-prone are actually defect-prone) and a recall of 81.25% (how many of the defect-prone binaries can we predict). The opposite direction did not work out in terms of recall (only 4.12%). We tried to balance the granularity of files versus binaries and clustered the Firefox files into binary-like sets, following the Firefox directory structure, and re-ran the experiment. The result did not change, indicating that the granularity level likely had no impact on the outcome.

One might wonder why cross-project defect prediction is not always bidirectional. For Firefox and Internet Explorer a possible reason could be the mismatch between numbers of observations. Firefox has more files than Internet Explorer has binaries. Building a model from a small population to predict a larger one is likely more difficult than the reverse direction.

To summarize, Firefox and Internet Explorer share similar features and components because of their common domain; however, they are different in the process and tools used for their development. The results of our experiment show that Firefox is a strong defect predictor for Internet Explorer (but not the reverse) and an example for a cross-project predictor. These first results and insights motivated us to conduct further experiments with more systems from different domains as well as companies since the reasons to us were not obvious. We report on those experiments and their findings in the remainder of this paper.

Firefox predicts IE, but IE does not predict Firefox. Why?

3. DATA COLLECTION

Based on the results of the Firefox and IE cross-project prediction experiment, we decided to analyze seven more Microsoft and three more Open Source Software (OSS) systems: Microsoft's Direct-X, Internet Information Services (IIS), Windows Clustering, Windows Printing, Windows File System, Windows Kernel and SQL Server 2005; Apache's Derby and Tomcat, and Eclipse. Our hypothesis was that the sample of projects resembles some substantially large applications which share some commonalities (e.g. IIS and Tomcat, File System and Derby, etc.). The goal was to find out which cross-predictions work and which project characteristics contribute.

Table 1. Software systems studied.

System	Releases	No. of versions	Level	Total LOC	Total Churn
Firefox, an open source web-browser	1.5, 2.0	2	File Component	3.2 – 3.3 MLOC	0.64 – 0.95 MLOC
Internet Explorer (IE), Microsoft's web-browser	7.0.6001.18000 (released with Vista)	1	Binary	2.30 MLOC	2.20 MLOC
Direct-X, is a collection of APIs for handlings tasks related to the multimedia on Windows platforms	DirectX 10 (released with Vista)	1	Binary	1.50 MLOC	1.00 MLOC
Internet Information Services (IIS), a set of Internet-based services for server systems using Microsoft Windows platform	IIS 7.0	1	Binary	2.00 MLOC	1.20 MLOC
Clustering, part of the Windows Server system to enable computer servers to work together in a networked fashion	released with Vista	1	Binary	0.65 MLOC	0.84 MLOC
Printing, is print mechanism for Windows to print documents	released with Vista	1	Binary	2.40 MLOC	2.20 MLOC
File system, manages the core file system activities in Windows	released with Vista	1	Binary	2.00 MLOC	2.20 MLOC
Kernel, is the core engine that controls and governs the execution of the Windows operating system	released with Vista	1	Binary	1.90 MLOC	3.20 MLOC
SQL Server 2005, SQL server is Microsoft's relational database management system.	SQL Server 2005	1	Component	4.6 MLOC	7.2 MLOC
Eclipse, an open source Integrated Development Environment (IDE)	2.0, 2.1, 3.0	3	File Component	0.79 – 1.3 MLOC	1.0 - 2.1 MLOC
Apache Derby, a Java relational database system developed by the Apache foundation	Series 10	4	File	0.49 – 0.53 MLOC	4 – 23 KLOC
Apache Tomcat, is a servlet container providing a web server environment to run Java applications	5.x, 6.x	6	File	0.25 – 0.26 MLOC	8 – 98 KLOC

3.1 Projects

Table 1 gives a summary of the systems we used for the study. We used multiple versions of each system and analyzed a total of 35+ Million LOC (MLOC) for our experiment (the table shows minimum and maximum size of each analyzed system for multiple versions). Microsoft systems could be observed on a binary level; Apache, Firefox, and Eclipse on a per file level. In addition we observed Firefox and Eclipse on a per component/plug-in level to assess the effect of granularity of measurement on the results.

3.2 Code measures

For our experiments we collected the following metrics for each system: number of observations (file count, binary count, component count), total lines of code (LOC), added LOC, deleted LOC, modified LOC, number of edits (commits), cyclomatic complexity, number of bugs (i.e. pre-release), number of developers, number of defects (post-release, the predicted variable). The churn metrics were collected relative to the previously released version; for example over a period of 1.0 to 1.5, 1.5 to 2.0 for Firefox; for IE from 7.0.5730.13 (released with XP) to 7.0.6001.18000 (released with Vista).

When building our prediction models using logistic regression, we use relative measures of the extracted metrics (added LOC, deleted LOC, modified LOC, cyclomatic complexity, and pre-release bugs). Relative measures are normalized values of the various metrics obtained during the development process. We use total LOC as the normalization parameter. Munson et al. [21] use a similar relative approach towards establishing a baseline while studying code churn. Studies have shown that absolute measures like LOC are poor predictors of pre- and post release defects [12] in industrial software systems. In an evolving system, a relative approach is highly beneficial to quantify the change in a system. Also prior work by Nagappan and Ball [23] showed that relative code churn measures are significantly stronger predictors of defect density in the Windows Server 2003 system than absolute code measures. They found that 89% of defect-prone binaries in Windows Server 2003 can be identified using relative code churn measures. For our study, we also checked how well predictors perform when built from absolute code metrics. None of them were able to predict well for other projects, which is why we focus on relative measures in this paper.

The relative measures used in our experiments capture various aspects of software development.

Added LOC/Total LOC: A large magnitude of this metric indicates the possibility of new features being written in the file/binary.

Deleted LOC/Total LOC: This measure indicates whether significant functionality has been removed from the files/binary. This is also a source of potential problems when code was deleted without checking for code dependencies.

Modified LOC/Total LOC: This measure is used to account primarily for bug fixes. It measures the extent to which lines of code are modified with respect to the overall file/binary size.

Pre-release bugs/Total LOC: This measure serves as a cross check with the Deleted LOC/Total LOC and Modified LOC/Total LOC so that if a bug fix in a particular file was to delete a few lines of code, our predictors are able to withstand the variation in the metric values.

(Added + Modified + Deleted LOC) / (Commits+1): This measure quantifies the extent of overall work done in a file/binary per check-in. It also cross checks the three code churn measures to make sure no single measure inflates the prediction variables.

Cyclomatic complexity/Total LOC: With this measure we get a relative estimate of the complexity of the binary/file. Cyclomatic complexity was used in several previous studies and found to be a strong indicator of code quality [26].

There is an extensive body of knowledge on the metrics discussed above when used for predicting quality [12, 13, 19, 22, 28, 29]. We use the above six metrics for each system to build our prediction models. The dependent variable for all systems is whether a binary, a component, or a file is defect-prone or not.

4. EXPERIMENTS

Our case study in Section 2 showed that Firefox is a strong defect predictor for Internet Explorer and a good example for cross-project prediction. However, the prediction worked only in one direction and Internet Explorer failed to predict defects for Firefox. In this and the following section, we will argue what can make cross-prediction work by running a much larger experiment than the initial Firefox/ Internet Explorer case study.

4.1 Methodology

For the experiment we used the 28 datasets from the 12 products discussed in Section 3.1 and checked for all possible combinations with *unseen* data whether cross-project prediction works. In this context "unseen" means that we considered a combination (A, B) to train a model from A to predict B if and only if the products of A and B are different or B is a later version than A. For example, we used Eclipse 2.0 to predict Eclipse 2.1 and later, but we did not use Eclipse 2.1 to predict Eclipse 2.0. Out of 28×27=756 possible combinations this left 719 combinations for our experiments.

Next we ran cross-project predictions for defect-proneness. Defect-proneness is the probability that a particular software element (such as a binary) will fail in operation in the field (i.e., will have post-release defects). The higher the defect-proneness, the higher is the probability of experiencing a post-release defect. To classify the binaries/components/files in our subject systems into two categories (not defect-prone and defect-prone) and taking a conservative approach, we define a statistical *lower confidence bound* on all post-release defects for each project. Elements with fewer defects than the lower confidence bound are classified as not defect-prone; all other elements are classified as defect-prone.

For each valid combination (A, B), we built a logistic regression model from A and tested how well it classified elements as defect-prone in B. To assess the model we used precision, recall, and accuracy:

- Precision addresses how many of the elements returned by a model were actually defect-prone. The best precision value is 1.0; the higher the precision, the fewer false positives (i.e., elements incorrectly classified as defect-prone).
- Recall addresses how many of the defect-prone elements were actually returned by a model. The best recall value is 1.0; the higher the recall, the fewer false negatives (i.e., elements missed by the model)
- Accuracy is the percentage of correctly classified elements (both as defect-prone and not defect-prone). The best accuracy value is 1.0.

For some project combinations, logistic regression gave the same prediction for all elements of the test projects (either all elements as defect-prone or all elements as defect-free). Such models are unusable in practice. However, they still can yield acceptable precision or recall values. For example, a model, which classifies everything as defect-prone, will have a recall of 1.0 (and very low precision). To avoid any bias in our experiments, we removed all such combinations. Out of the 719 original combinations, this resulted in 622 combinations.

The general form of a logistic regression equation is given in Equation 1,

$$\text{Probability}(\pi) = \frac{e^{(c + a_1X_1 + a_2X_2 + \dots)}}{1 + e^{(c + a_1X_1 + a_2X_2 + \dots)}} \quad (1)$$

where a_1, a_2, … are the logistic regression predicted constants and the X_1, X_2, … are the independent variables used for building the logistic regression model. In our case, the independent variables are the relative measures (churn, complexity, pre-release bugs), which we defined in Section 3.2.

4.2 Cross-project prediction results

We considered a project as a strong predictor for another project, if, and only if, all precision, recall, and accuracy were greater than 0.75. These thresholds are based on our previous (independent) studies of defect predictions for Eclipse [35], Mozilla [17], and Windows [27]. Out of the 622 non-trivial cross-project combinations, only 21 had precision, recall, and accuracy values which satisfied our criteria; an alarmingly low success rate of 3.4%.

Figure 2 shows the outcome of the experiment for the individual projects. Each node corresponds to a project; for example, an edge from a node Derby to a node Clustering means that Derby is a strong predictor for Clustering. The weight of the edge indicates how many Derby versions were strong predictors for Clustering. The color of node tells whether a project is a predictor of other projects (white), can be predicted (black), or both (gray).

In Figure 2 we depict the results of our experiments:

- The OSS projects in our study (Firefox, Derby, Tomcat, and Eclipse) are strong predictors for closed-source projects but do not predict the other OSS.
- The OSS projects cannot be predicted by any of the projects in our study.
- On the closed-source side, we see projects such as File System, Printing, Clustering, and IIS that can predict other closed-source projects. However, we also see projects such as Internet Explorer, Kernel, and Direct-X that do not predict other systems at all.
- And we find systems that are active and passive predictors, i.e., predict each other, such as Printing, Clustering, and IIS.

The database management system Derby has a very machine-oriented programming level and therefore may be a predictor for Clustering and Kernel, which share similar characteristics. Eclipse is well-known for its API, which might explain why it predicts Direct-X and Kernel, again two projects with a fairly large API. Printing, Clustering and IIS seem to be in a near-perfect "magic triangle" of mutual prediction. We identified several possible reasons for this. One of them is the age of the code base. All three projects have been reengineered at the same time and therefore their code bases have a similar age, possibly this has impact on the defect prediction across them. Another reason is that the three projects have similar relative code churn profiles, which correlate equally to the number of defects in their respective training sets.

From Figure 2 it is not obvious which projects are good predictors for other project. Also, there are many different characteristics that seem to influence whether cross-project predictions work. In the next section, *we present a systematic approach to assess the effect of certain characteristics on cross-project predictions.*

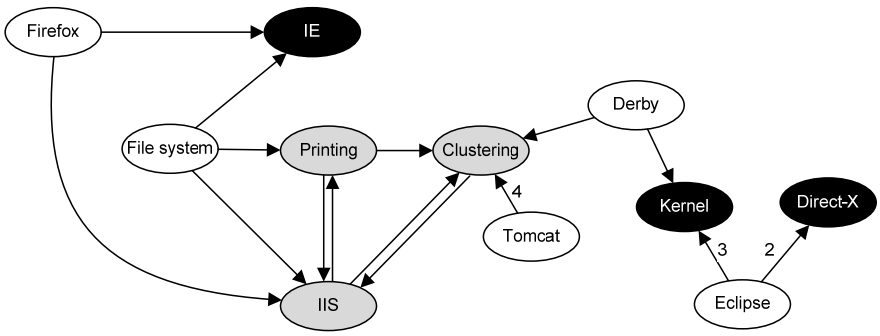

Figure 2. Results from 622 cross-project defect predictions. For example, Firefox data can predict IE. The color tells whether a project predicts other projects (white), can be predicted (black), or both (gray).

5. CROSS-PROJECT PREDICTABILITY

The previous section showed that for a successful cross-project defect prediction, the training project needs to be chosen very carefully and that the choice is often not obvious. Given that only 3.4% of our cross-project experiments actually worked, a random or arbitrary choice of the training project will very likely result in poor predictions and lead managers to wrong decisions. Ideally one would have a set of guidelines and rules that enable assessing the chances of success before any cross-project prediction is attempted. In this section, we describe a technique to derive such guidelines. The approach consists of the following steps.

1.) Describe each project p (the set of all projects is P) with a set of characteristics C. The characteristics of a project p is a vector of values and denoted as $c(p)$.

In total we used 40 characteristics, which described the domain, process, and data of each project. (Section 5.1)

2.) Compare all projects pair-wise with respect to their characteristics. For each two projects p_i and p_j, the result is a similarity vector $s(p_i,p_j)$, which describes whether characteristics are the same or different. For example, to describe similarity for characteristic "Project does code reviews" we use the levels "Both no", "Both yes", and "Different" (Section 5.2).

We get 622 similarity vectors. The next two steps analyze how similarity of characteristics affects the precision, recall, and accuracy values from Section 4.

3.) For each characteristic and its levels in the similarity vector, we check with a Welch t-test whether the level affects precision, recall, and/or accuracy (Section 5.3).

For the 40 characteristics, we have 125 levels in the similarity vectors. For each level, we run three tests to assess effect on precision, recall, and accuracy (in total 375 tests). We applied a Bonferroni correction to account for multiple hypothesis testing.

4.) Build decision trees from the similarity data to describe precision, recall, and accuracy (Section 5.4).

While the previous step analyzed similarity levels independently, a decision tree also looks at interactions. In addition, it provides a handy tool to estimate precision, recall, and accuracy beforehand.

5.1 Describing projects

Based on Goal Question Metric (GQM) [2] we defined

Goal: Identify the best predictor for code quality

Question: What characteristics differ between projects used for building predictors?

Metric: The 40 characteristics that we used are explained below in the following format.

Name of characteristic: (values it takes): Description

Domain: (Name): To assess whether products of the same domain predict each other. For example Firefox and IE, Derby and SQL belong to the same domain of web browsers and databases, respectively.

Company: (Mozilla Corp/Microsoft/Eclipse community/Apache foundation): to assess whether projects from the same company predict each other.

Product: (Name): to assess if products in the same family predict each other; for example, does Eclipse 2.0 predicts Eclipse 2.1?

Open source: (Yes/No): to distinguish between open source and closed source systems.

Global development: (Yes/No): to assess whether systems developed by teams distributed across the world predict each other.

Code reviews: (Yes/No): to consider code quality processes such as whether code reviews were employed as part of the development process.

Static checkers: (Yes/No): to take into account code quality checks such as whether static code checkers were used during the development of the system.

Intended audience: (End user/Developer): to assess whether systems built for interaction with end users predict systems built for professionals like software engineers, e.g. Tomcat.

Operating system: (Windows/Multiple): to take into account whether a system runs only on Windows or on multiple operating systems such as Suse Linux, Mac OS, Windows, etc.

Type of user interface: (Graphical/Toolkit/Non-interactive): this measure is also to describe the type of the system, e.g. Firefox has a UI, Printing is a toolkit, Tomcat is not interactive etc.

Product uses database: (Yes/No): to specify whether a system uses a database.

Product is localized: (Yes/No): to address whether the systems are available in one language (say English) or are available in several international languages (like IE and Firefox are in English, Chinese, German, Russian, etc.).

Programming language: (C and C++/CSHARP/Java): to track all the programming languages used in the system development.

Single programming language: (Yes/No): to reflect if the system uses only one programming language throughout the system.

Project uses C/C++: (Yes/No): self explanatory.

Project uses C#: (Yes/No): self explanatory.

Project uses Java: (Yes, No): self explanatory.

First version: (Yes/No): Is this the first version of a system.

Level of analysis: (File/Binary or Plug-in or Component): This indicates the lowest level at which metrics are tracked by the project. For example: Eclipse is plug-ins (or components) whereas Firefox does not follow this model and the level is therefore files.

Total number of lines of code: (Numerical): self explanatory.

Number of developers: (Numerical): self explanatory.

Number of observations: (Numerical): the number of files/ components/binaries in a project

Median, maximum and standard deviation: (Numerical): For each metric that we used for the prediction models (Section 3.2) we computed median, maximum and standard deviation to describe the input data that a model can expect. This results in a total of 18 different characteristics that describe input data.

5.2 Similarity between projects

For two projects p_i (train) and p_j (test) we create the similarity vector $s(p_i,p_j)$ as follows from the characteristic vectors $c(p_i)$ and $c(p_j)$. With $v[n]$ we describe the value of n in a vector v.

For each characteristic n in C, we do the following.

- If n is "Domain", "Product", "Programming languages", or "Level of analysis" and
  - $c(p_i)[n] == c(p_j)[n]$ then $s(p_i,p_j)[n]$:= "Same"
  - $c(p_i)[n] <> c(p_j)[n]$ then $s(p_i,p_j)[n]$:= "Different"

- If n is a nominal characteristic (e.g., "Open Source") and
  - $c(p_i)[n] == c(p_j)[n]$ then $s(p_i,p_j)[n]$:= "Both" + $c(p_i)[n]$
  - $c(p_i)[n] <> c(p_j)[n]$ then $s(p_i,p_j)[n]$:= "Different"

  For example, if a characteristic is "Yes" for both projects the similarity level will be set to "Both Yes". If one project has "Yes" and the other "No" the similarity level will be "Different".

- If n is a numerical characteristic (e.g., "LOC") and
  - $0.9 \times c(p_i)[n] > c(p_j)[n]$ then $s(p_i,p_j)[n]$:= "Less"
  - $1.1 \times c(p_i)[n] < c(p_j)[n]$ then $s(p_i,p_j)[n]$:= "More"
  - Else $s(p_i,p_j)[n]$:= "Same"

  The factors 0.9 and 1.1 are used to allow for small deviations between projects. For example, if two projects have 1.0 MLOC and 1.05 MLOC we consider the characteristic "LOC" as the same.

As an example take Firefox as the training project p_i and Internet Explorer (IE) as the test project p_j. For the example we consider only a subset of characteristics that is displayed in the table below. Both projects are browsers, thus the "Domain" characteristic is set to "Same" in the similarity vector. Firefox is open source, IE is not, thus "Open source" is set to "Different". Both projects do code reviews ("Both Yes"). IE is smaller than Firefox in terms of lines of code, thus "LOC" is set to "Less".

	Characteristics			
Project	Domain	Open source	Code reviews	LOC
Train: Firefox (p_i)	Browser	Yes	Yes	3.2M
Test: IE (p_j)	Browser	No	Yes	2.3M
Similarity $s(p_i,p_j)$	Same	Different	Both Yes	Less

We computed similarity vectors for each of the 622 combination of projects that we used in Section 4.

5.3 Effect of similarity on predictions

For each similarity level (e.g., "Both Yes") of each characteristic (e.g., "Open Source"), we did three Welch t-tests to compare the mean precision, recall, and accuracy for the level against its complement level (e.g., in the case of "Both Yes", the complement level is the combination of "Both No" and "Different").

For the tests in this section we used an overall p-value of 0.05. Because we tested multiple hypotheses (125×3=375 tests) we applied Bonferroni correction, i.e., a test result has to be significant at p=0.05/375=0.000133 before it is included in this section.

Table 2 shows the results for nominal characteristics and Table 3 the results for numerical characteristics. The rows correspond to the characteristics, and the columns to the effect on precision, recall, and accuracy. In Table 2 the levels are prefixed to each row and in Table 3, the levels "Less", "Same", and "More" each group together three columns. Each level can increase (UP), decrease (DOWN), or have no statistically significant effect (— or omitted) on precision, recall, and accuracy.

For example, explaining the first row in Table 2, we can observe that the having same domain increases accuracy of cross-project predictions, but has no impact on precision and recall. At the same having a different domain decreases accuracy with no effect on precision or recall. When projects of different companies are used the precision decreases, an observation that has also been made by Turhan et al. [32]. A result that matches with our observation that Internet Explorer fails to predict Firefox is that when the number of observations is "More" for the test projects (Firefox has more files than Internet Explorer has binaries), the precision of cross-project predictions decreases.

Several similarity levels increase precision and recall, but decrease accuracy. We believe that this is an instance of the accuracy paradox, which states "that predictive models with a given level of accuracy may have greater predictive power than models with higher accuracy." [34]

In Table 3 it is noteworthy that higher medians of the code measures (i.e., higher churn, higher complexity, and more pre-release bugs) in the test project seem to increase precision and recall. This indicates that the success of cross-project defect prediction is largely data driven. However, we can also see from Tables 2 and 3 that there are many other driving factors, which often have opposite effects. For example, a project might have both more observations (files/binaries) and more pre-release defects than another project. In this case, more observations *decrease* precision, while more pre-release defects *increase* precision.

Next, we will look at the interaction of characteristics and show how decision trees can help to estimate precision, recall, and accuracy of a cross-project prediction.

Table 2. Nominal characteristics and how they influence precision, recall, and accuracy.

Factor	Both	Precision	Recall	Accuracy		Precision	Recall	Accuracy
Domain	Same:	—	—	UP	Different:	—	—	DOWN
Company	Apache: Microsoft:	— UP	DOWN —	— DOWN	Different:	DOWN	—	—
Product								
Open source	Yes: No:	— UP	DOWN —	UP DOWN	Different:	—	UP	DOWN
Global development	Yes: No:	— UP	DOWN —	UP —	Different:	—	UP	DOWN
Code reviews	Yes: No:	UP —	UP DOWN	DOWN UP				
Static checkers	Yes: No:	UP —	— DOWN	DOWN UP	Different:	—	UP	DOWN
Intended audience	Developer: End-user	DOWN UP	DOWN UP	— —				
Operating system	Multi: Windows:	— UP	DOWN —	UP DOWN	Different:	—	UP	DOWN
Type of user interface								
Product uses database	No:	UP	UP	—	Different:	DOWN	—	—
Product is localized	Yes:	UP	—	DOWN				
Programming languages	Same:	—	DOWN	UP	Different:	—	UP	DOWN
Project uses a single programming language	Yes: No:	— UP	DOWN —	— —				
Project uses C/CPP	Yes: No:	UP —	UP DOWN	DOWN UP				
Project uses C#	No:	—	DOWN	UP	Different:	—	UP	DOWN
Project uses Java	Yes: No:	— UP	DOWN UP	UP DOWN				
First version	Yes: No:	UP —	— DOWN	— UP	Different:	—	UP	DOWN
Level of analysis	Same:	—	DOWN	UP	Different:	—	UP	DOWN

Table 3. Numerical characteristics and how they influence precision, recall, and accuracy.

	Factor is Less			Factor is the Same			Factor is More		
Factor	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy
Number of lines of code	DOWN	—	—	—	—	—	—	—	—
Number of developers	—	—	—	—	—	—	—	—	—
Number of observations	UP	—	—	—	—	—	DOWN	—	—
median_added_rel	—	—	—	—	—	—	—	UP	DOWN
median_average_churn	DOWN	—	—	—	—	—	UP	—	—
median_bugs_rel	DOWN	—	DOWN	—	DOWN	UP	UP	UP	DOWN
median_cyclomatic_rel	—	DOWN	UP	—	—	—	—	UP	DOWN
median_deleted_rel	DOWN	—	—	—	DOWN	UP	UP	UP	DOWN
median_modified_rel	DOWN	—	—	—	—	—	UP	UP	—
sd_added_rel	—	DOWN	—	—	—	—	—	UP	—
sd_average_churn	DOWN	—	—	—	—	—	UP	—	—
sd_bugs_rel	—	DOWN	—	—	—	—	—	UP	—
sd_cyclomatic_rel	—	—	—	—	—	—	—	—	—
sd_deleted_rel	—	—	—	—	—	—	DOWN	UP	—
sd_modified_rel	—	—	—	—	—	—	—	—	—
max_added_rel	UP	DOWN	—	—	—	—	DOWN	UP	—
max_average_churn	—	—	—	—	—	—	—	—	—
max_bugs_rel	UP	DOWN	—	—	—	—	DOWN	UP	—
max_cyclomatic_rel	UP	—	—	—	—	—	DOWN	—	—
max_deleted_rel	UP	DOWN	—	—	—	—	DOWN	UP	—
max_modified_rel	UP	—	—	—	—	—	DOWN	—	—

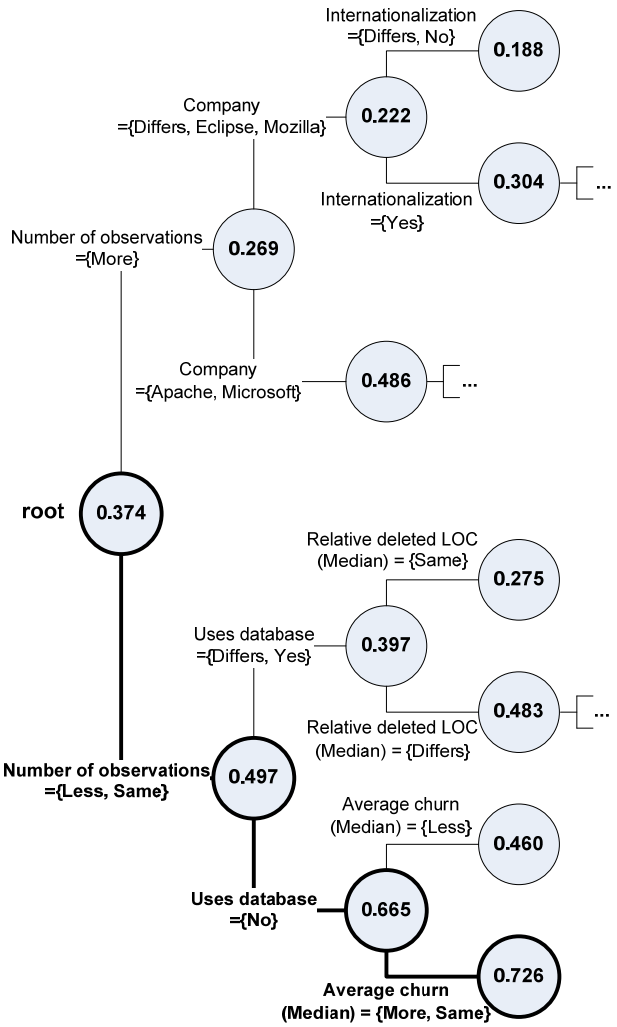

Figure 3. Decision tree for precision.

5.4 Estimate precision, recall, and accuracy

To find out how interactions of characteristics influence cross-project predictions, we computed three decision trees [15], one for precision, recall, and accuracy (dependent variable). As independent variables we used the similarities of the characteristics, which we defined in Section 5.1 and 5.2.

Figure 3 shows the decision tree for precision (only the top three levels and nodes with at least 60 data points). For the root node, which corresponds to all 622 cross-project predictions, the average precision is 0.374. The first level splits based on the *number of observations* (files/binaries/components). If the test project has the same or fewer observations the precision increases to 0.497, if it has more, the precision decreases to 0.269. Depending on the branch, the second level is then split based on company (upper branch) or the usage of a database (lower branch). This process continues until a leaf node is reached for which no further splits are possible. In Figure 3 the precision is below 0.500 for most leaves, only for "fewer or same number of observations", "no database" and "greater or equal median of average churn" the precision reaches a value of 0.726 (path is highlighted in bold).

Such a decision tree helps managers to decide whether they will attempt cross-project predictions. For example they can compare projects at hand to identify the similarities and then use the decision trees to estimate precision. Based on Figure 3, with IE as the test project and Firefox as the training project ("Number of observations" is "Less") a manager could expect a precision of 0.497. However, for the reverse direction ("More") the expected precision is only 0.269. This could explain why Firefox can predict IE, but not the other way around. Note that even though we show only average values in Figure 3, it is fairly straightforward to derive a confidence interval (say at 95%) for a node.

We also computed decision trees for recall and accuracy, which we omit because of limited space. For *recall,* the highest observed value was 0.728 for global development (differs or both no), median of relative number of pre-release bugs (more for test project), and intended audience (different or both end-user). For accuracy, the highest observed value was 0.843 for median of relative number of pre-release bugs (same), operating system (both support multiple systems), and the median of relative added LOC (fewer or same in the test project). We also computed a decision tree for *precision, recall, and accuracy all being greater than 0.75*. Here the highest observed success rate for cross-project prediction was 0.324 for median of relative number of pre-release bugs (more in test project), operating system (different or both Windows), and standard deviation of the relative cyclomatic complexity (higher in test project)

An alternative to decision trees can be case-based reasoning [15]. In a database of past cross-project prediction results, each characterized by a similarity vector, a manager could search for the same or similar vectors and use their precision/recall/accuracy values to estimate the success of a planned cross-project prediction.

6. THREATS TO VALIDTY

As stated by Basili et al., drawing general conclusions from empirical studies in software engineering is difficult because any process depends on a potentially large number of relevant context variables [4]. For this reason, we cannot assume a priori that the results of a study generalize beyond the specific environment and projects for which it was conducted.

Four our study we have analyzed 12 large and long-lived applications. The data extraction has been done systematically on a release by release basis using automated and proven data extractors. Out of the huge amount of metrics one could collect, we selected a subset that was reasonable based on previous experiments [23, 24, 26]. Other researchers could have made a different choice. Another possible threat is that the metrics collection might be slightly different between projects. This is alleviated to a large degree by using automated measurement and infrastructure tools, for open-source and commercial systems.

The size of the code bases and development teams from the 12 projects in our study will be different to many other commercial and open-source products. Thus it might be possible that our results do not generalize to other projects. This is often misunderstood as a criticism of empirical studies. This study shows surprising results (cross-project prediction is a serious challenge) to build up a body of empirical evidence (characteristics that influence precision and recall), which should encourage more researchers to run similar studies and deepen the understanding of the field. Ideally, our study would be replicated with more projects, different metrics, and more project characteristics.

7. RELATED WORK

In this section we discuss related work in the area of defect prediction (Section 7.1) and cross-project predictions, which has been mostly for effort estimation (Section 7.2).

7.1 Defect prediction

In this subsection we primarily focus on studies that have involved making predictions of defects/failures *using multiple versions of the same system*, in line with the broad objectives of our own study. Structural object-orientation (OO) measurements, such as those in the CK OO metric suite [9], have been used to evaluate and predict fault-proneness [3, 8].

Mockus et al. [20] predict the customer perceived quality using logistic regression for a commercial telecommunications system (of seven million LOC) by utilizing external factors like hardware configurations, software platforms, amount of usage and deployment issues. They observed an increase in probability of failure of twenty times by accounting for such measures in their predictions.

Ostrand et al. [29] use code measures in a negative binomial regression equation to predict the number of faults in a multiple release software system (size of the last release was 538 KLOC). The top 20% of the files so identified as fault-prone for fifteen consecutive releases representing four years of field usage and contained between 71% and 93% (average 84%) of the total faults in each release [29]. Nagappan et al. [25] used code churn (added, modified, deleted) LOC, code coverage and code complexity data from Windows XP-Service Pack 1 to successfully predict failures in Windows Server 2003.

Denaro et al. [11] used data from the open source Apache 1.3 to and Apache 2.0 projects. Using PCA, logistic regression models were built using the data from the Apache 1.3 project to predict defects successfully against the Apache 2.0 project. Gyimothy et al. [14] mined the CK metrics data from Mozilla versions (1.0-1.6) and investigated their relationship and ability to predict fault classes. Their results indicated that CBO was the best predictor of fault-prone classes; DIT was untrustworthy and NOC could not be used to predict fault-prone classes.

Ekanayake et al. [11] observed concept drift in defect prediction, i.e., models become unsuitable as influencing features change. As features they found the number of authors editing a file and the number of defects fixed by them. Bird et al. studied bias in bug datasets and found that prior experience as well as severity can influence how well bugs are linked to version archives [5].

7.2 Cross-project predictions

In this subsection we primarily discuss research that has used metrics from one project to predict characteristics of metrics in a different project (i.e. not belonging to the same family of versions). Most of the research in this area has primarily focused on cost estimation. For example, cost estimation models such as COCOMO [6], SLIM [30], Function Points[1], or ESTIMACS [31] have provided us with general purpose models that can be applied to arbitrary projects. However, some studies, for example by DeMarco [10], have argued for single company estimation models. As a consequence, studies have been undertaken to address both cross-company and within-company effort estimation. Results do not show a conclusive picture so far.

Most research so far has addressed effort estimation in cross-company models [7, 18, 33]. In a recent survey, Kitchenham et al. [16] found inconclusive results: although some organizations would benefit from cross-company benchmarks, there is no clear indicator of when it works most effectively. In their survey, seven out of ten studies provided evidence that cross-company effort prediction is viable. A point to be noted is that this is survey data as opposed to actual statistical experiments on data. They also suggested testing specific hypotheses about the conditions that favor the use of cross-company estimation models [16]. This is what our study provides: first, we deal with an independent data set within large companies (or organizations) like Microsoft, Apache foundation, Mozilla Corporation etc. Second, we focus on the domain of a software project to test the favor of cross- and within-company prediction models.

For their study Turhan et al. [32] analyzed 12 NASA projects (mostly in C++) which they considered cross-company because they were all developed by contractors under the umbrella of NASA. However, all projects had to follow stringent ISO-9001 industrial practices imposed by NASA, so it is unclear to what extent the data can be actually considered cross-company. Their study shows that cross-company data dramatically increases the probability of failure detection but it also dramatically increases the false positives rate. In cross-company failure prediction, the false positive rate increased up to 73% (with a median of 52%), which shows that there is a large drawback for cross-company prediction models.

Our study in comparison to Turhan et al. [32] addresses defect prediction across projects based on static code measures such as code churn (code modified, added, deleted) and code complexity, metrics. Our data originates from within-company projects of Microsoft, Apache Foundation but also from *true* cross-company data such as Firefox. Furthermore, we address how the domain and the process influence cross-project predictions.

8. CONCLUSION AND CONSEQUENCES

Cross-project defect prediction is important for projects with little or insufficient data to build prediction models. However, until now research has paid little attention to this problem.

For this paper we ran 622 cross-project predictions and found that only 3.4% actually worked. We also tested the influence of several factors on the success of cross-project prediction. Here, we found that data and process seemed to be crucial factors. However, there was no single factor that led to success. To accommodate for the combination of factors, we used decision trees, which can help managers to estimate precision, recall, and accuracy before attempting a prediction across projects.

Our main contributions in this paper are the following.

1. Empirical evidence that cross-project prediction is a serious problem. That is, simply using projects in the same domain does not work to build accurate prediction models. Process, data and domain need to be quantified, understood and evaluated before prediction models are built and used.

2. An approach for identifying factors that influence the success of cross-project predictions. Software engineers can this technique to locate projects for building cross-project predictors. Though our study uses 12 different real world applications, as with all empirical studies, our study will need to be replicated with more projects, different metrics, and more project characteristics.

For future work we envision the following.

- Out of the factors that we analyzed, domain was surprisingly not very significant. Possibly, this is because it is the most difficult to describe and the measures that we use for domain were rather simplistic. More research is needed to find out how to best describe the domain of a software automatically.
- Our study also leads to many follow-up questions: Why is defect-prediction not transitive? For example, File system predicted Printing and Printing predicted Clustering; however File system did not predict Clustering? Can we find a transformation that makes prediction transitive? For our experiments, we used fixed sets of measures. However, it could well be that Internet Explorer predicts Firefox, but only with a different set of metrics. How can we find the right set of metrics to use so that a project predicts another project?

A consequence for research is that rather than increasing the precision and recall of models by some small percentage, it should focus on how to make defect prediction work across projects and relevant for a wide audience. We believe that this will be an important trend for software engineering in general. Learn from one project, to improve another.

Acknowledgements. We thank Rebecca Aiken, Martin Robillard, and the anonymous ESEC/FSE reviewers for feedback on earlier versions of this paper.

9. REFERENCES

[1] A. J. Albrecht and J. R. Gaffney, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," *IEEE Transactions in Software Engineering,* vol. 9, pp. 639-648, 1983.

[2] V. Basili, G. Caldiera, and D. H. Rombach, "The Goal Question Metric Paradigm," in *Encyclopedia of Software Engineering*. vol. 2: John Wiley and Sons, Inc., 1994, pp. 528-532.

[3] V. R. Basili, L. C. Briand, and W. L. Melo, "A Validation of Object Orient Design Metrics as Quality Indicators," *IEEE Transactions on Software Engineering,* vol. 22, pp. 751-761, 1996.

[4] V. R. Basili, F. Shull, and F. Lanubile, "Building Knowledge Through Families of Experiments," *IEEE Transactions on Software Engineering,* vol. 25, pp. 456 - 473, 1999.

[5] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu, "Fair and Balanced? Bias in bug-fix Datasets," in *European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2009.

[6] B. W. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece, *Software Cost Estimation with COCOMO II*. Upper Saddle River, NJ: Prentice Hall, 2000.

[7] L. Briand, T. Langley, and I. Wieczorek, "A Replicated Assessment of Common Software Cost Estimation Techniques," in *International Conference on Software Engineering*, 2000, pp. 377-386.

[8] L. C. Briand, J. Wuest, S. Ikonomovski, and H. Lounis, "Investigating quality factors in object-oriented designs: an industrial case study," in *ICSE*, 1999, pp. 345-354.

[9] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering,* vol. 20, pp. 476-493, 1994.

[10] T. DeMarco, *Controlling Software Projects: Management Measurement and Estimation*: Yourdon Press, 1982.

[11] J. Ekanayake, J. Tappolet, H. C. Gall, and A. Bernstein, "Tracking Concept Drift of Software Projects Using Defect Prediction Quality," in *IEEE Working Conference on Mining Software Repositories*, 2009.

[12] N. E. Fenton and N. Ohlsson, "Quantitative analysis of faults and failures in a complex software system," *IEEE Transactions on Software Engineering,* vol. 26, pp. 797-814, 2000.

[13] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy, "Predicting Fault Incidence Using Software Change History," *IEEE Transactions on Software Engineering,* vol. 26, pp. 653-661, 2000.

[14] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Transactions in Software Engineering,* vol. 31, pp. 897 - 910 2005.

[15] J. Han and M. Kamber, *Data Mining Concepts and Techniques*: Elsevier, 2006.

[16] B. Kitchenham, E. Mendes, and G. H. Travassos, "Cross- vs. within-company cost estimation studies: A systematic review," *IEEE Transactions in Software Engineering,* vol. 33, pp. 316-329, 2007.

[17] P. Knab, M. Pinzger, and A. Bernstein, "Predicting defect densities in source code files with decision tree learners," in *Mining Software Repositories (MSR 06)*, 2006, pp. 119-125.

[18] E. Mendes and B. Kitchenham, "Further Comparison of Cross-Company and Within-Company Effort Estimation Models for Web Applications," in *IEEE International Symposium on Software Metrics* 2004, pp. 348-357.

[19] T. Menzies, J. Greenwald, and A. Frank, "Data Mining Static Code Attributes to Learn Defect Predictors," *IEEE Transactions in Software Engineering,* vol. 33, pp. 2-13, 2007.

[20] A. Mockus, P. Zhang, and P. Li, "Drivers for customer perceived software quality," in *International Conference on Software Engineering (ICSE 05)*, St. Louis, MO, 2005, pp. 225-233.

[21] J. Munson and T. Khoshgoftaar, "The Detection of Fault-Prone Programs," *IEEE Transactions on Software Engineering,* vol. 18, pp. 423-433, 1992.

[22] J. C. Munson and S. Elbaum, "Code Churn: A Measure for Estimating the Impact of Code Change," in *IEEE International Conference on Software Maintenence*, 1998, pp. 24-31.

[23] N. Nagappan and T. Ball, "Use of Relative Code Churn Measures to Predict System Defect Density," in *International Conference on Software Engineering (ICSE)*, St. Louis, MO, 2005, pp. 284-292.

[24] N. Nagappan and T. Ball, "Using Software Dependencies and Churn Metrics to Predict Field Failures: An Empirical Case Study," in *International Symposium on Empirical Software Engineering and Measurement*, 2007, pp. 364-373.

[25] N. Nagappan, T. Ball, and B. Murphy, "Using Historical In-Process and Product Metrics for Early Estimation of Software Failures," in *International Symposium on Software Reliability Engineering*, 2006, pp. 62-74.

[26] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *International Conference on Software Engineering*, 2006, pp. 452-461.

[27] N. Nagappan, B. Murphy, and V. Basili, "The Influence of Organizational Structure on Software Quality: An Empirical Case Study," in *Int. Conference on Software Engineering*, 2008, pp. 521-530.

[28] M. C. Ohlsson, A. von Mayrhauser, B. McGuire, and C. Wohlin, "Code Decay Analysis of Legacy Software through Successive Releases," in *IEEE Aerospace Conference*, 1999, pp. 69-81.

[29] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Where the Bugs Are," in *International Symposium on Software Testing and Analysis (ISSTA)*, 2004, pp. 86-96.

[30] L. Putnam and A. Fitzsimmons, "Estimating software costs," *Datamation,* vol. 25, 1979.

[31] H. A. Rubin, "Macroestimation of software development parameters:The Estimacs system," in *SOFTFAIR Conference on Software Development Tools, Techniques and Alternatives* 1983, pp. 109-118.

[32] B. Turhan, T. Menzies, A. B. Bener, and J. D. Stefano, "On the relative value of cross-company and within-company data for defect prediction " *Empirical Software Engineering,* DOI: 10.1007/s10664-008-9103-7, 2009.

[33] I. Wieczorek and M. Ruhe, "How Valuable Is Company-Specific Data Compared to Multi-Company Data for Cost Estimation?," in *International Symposium on Software Metrics*, 2002, pp. 237-246.

[34] X. Zhu, *Knowledge Discovery and Data Mining: Challenges and Realities*: IGI Global, 2007.

[35] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting Defects for Eclipse," in *Third International Workshop on Predictor Models in Software Engineering* 2007.