Local versus Global Lessons for Defect Prediction and Effort Estimation

Tim Menzies, *Member*, *IEEE*, Andrew Butcher, David Cok, *Member*, *IEEE*, Andrian Marcus, *Member*, *IEEE*, Lucas Layman, *Member*, *IEEE*, Forrest Shull, *Member*, *IEEE*, Burak Turhan, *Member*, *IEEE*, and Thomas Zimmermann

Abstract—Existing research is unclear on how to generate lessons learned for defect prediction and effort estimation. Should we seek lessons that are global to multiple projects or just local to particular projects? This paper aims to comparatively evaluate local versus global lessons learned for effort estimation and defect prediction. We applied automated clustering tools to effort and defect datasets from the PROMISE repository. Rule learners generated lessons learned from all the data, from local projects, or just from each cluster. The results indicate that the lessons learned after combining small parts of different data sources (i.e., the clusters) were superior to either generalizations formed over all the data or local lessons formed from particular projects. We conclude that when researchers attempt to draw lessons from some historical data source, they should 1) ignore any existing local divisions into multiple sources, 2) cluster across all available data, then 3) restrict the learning of lessons to the clusters from other sources that are nearest to the test data.

Index Terms—Data mining, clustering, defect prediction, effort estimation

1 Introduction

Process and product data are used in software engineering (SE) to support a variety of tasks, such as defect prediction, effort estimation, refactoring of source code, determination of the social networks of programmers, learning the expected interface between modules, and so on. Two questions are at the center of much of the research in the field: 1) What data (e.g., which metrics) are best suited to support specific tasks? and 2) what is the best way to reason about SE data? This paper explores the latter question, in the context of:

- *software effort reduction*: finding rules for reducing a project's development time;
- *software defect reduction*: finding rules for reducing a project's defect count.

Our focus in this paper is not on what data (e.g., process or product data) are used for building models for defect prediction or effort estimation, but rather on the source of the data and, implicitly, the applicability of the lessons derived by the models. Software *data* come from some *sources* (e.g., different companies for effort data or different projects for defect data). These data show the defects or effort associated with examples (e.g., projects for effort data or classes for defect data) from that source.

How should we reason about these data? On this point, the literature is contradictory. Some existing work argues that *data from multiple sources* can generate rules that apply in any context (i.e., project or company) [1], [2], [3], [4]. We call these *global lessons* (or rules).

On the other hand, other papers indicate that the best lessons are learned from *within one source* (rather than across all sources), which implies that these lessons are only useful in their context [5], [6], [7]. We call these *local lessons*.

When project managers want to make changes in their projects to minimize the development effort or the rate of defects, they are faced with two options: 1) make changes based on *global lessons* available from existing data, or 2) mine data about the current project and infer *local lessons*. The dilemma of the manager is obvious. In the first case, expensive changes (based on the global lessons) may be undertaken without reaching the desired goal if the global lessons prove to be wrong for this context. In the second case, an upfront investment is needed to collect and analyze data and to generate the local lessons, which may be unnecessary if the global lessons apply to this context.

This paper addresses this dilemma and reports on experiments where:

- Data from different sources are combined.
- Within that combination, automatic tools find clusters of related examples. Note that clusters may contain examples from multiple sources.
- Data mining is then used to learn lessons from the examples in each cluster.
- The generated lessons are compared.

- *T. Menzies and A. Butcher are with the Lane Department of Computer Science and Electrical Engineering, West Virginia University, Morgantown, WV. E-mail: tim@menzies.us, abutcher@afrolegs.com.*
- *D. Cok is with GrammaTech, Ithaca, NY. E-mail: dcok@grammatech.com.*
- *A. Marcus is with Wayne State University, Detroit, MI. E-mail: amarcus@wayne.edu.*
- *L. Layman and F. Shull are with the Fraunhofer Center, University of Maryland, College Park, MD. E-mail: {llayman, fshull}@fc-md.umd.edu.*
- *B. Turhan is with the Department of Information Processing Science, University of Oulu, Oulu, Finland. E-mail: turhanb@computer.org.*
- *T. Zimmermann is with Microsoft Research, Redmond, WA. E-mail: tzimmer@microsoft.com.*

Manuscript received 12 Jan. 2012; revised 16 Aug. 2012; accepted 22 Oct. 2012; published online 27 Nov. 2012.
Recommended for acceptance by R. Jeffery.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2012-01-0013.
Digital Object Identifier no. 10.1109/TSE.2012.83.

Before this experiment, our previous work indicated that local lessons seem to be best for defect prediction and effort estimation [8], [9]. The surprising result of the experiment presented in this paper is that the best lessons for a project from one source come from *neighboring clusters with data from nearby sources, but not inside that source*. We will call such lessons *neighbor lessons*.

We conclude that when project managers attempt to draw lessons (i.e., about defect or effort reduction) from some historical data source, they should: 1) ignore any existing local divisions into multiple sources, 2) cluster across all available data, then 3) restrict the learning of lessons to the clusters that are nearest to the test data (regardless of the source of the data). While the paper focuses on defect prediction and effort estimation data, we believe that our conclusions can translate to other types of data, related to different SE tasks.

This paper extends a prior publication [10] in the following way:

- That paper only explored four datasets. Here, we explore over twice that number of datasets.
- That prior publication only explored local versus global lessons and found evidence that supported local lessons over the global ones. In this paper, we offer new experiments in Section 4 (which have not appeared previously) that explore clusters from multiple sources.
- This paper's literature review is more extensive (see Fig. 2).

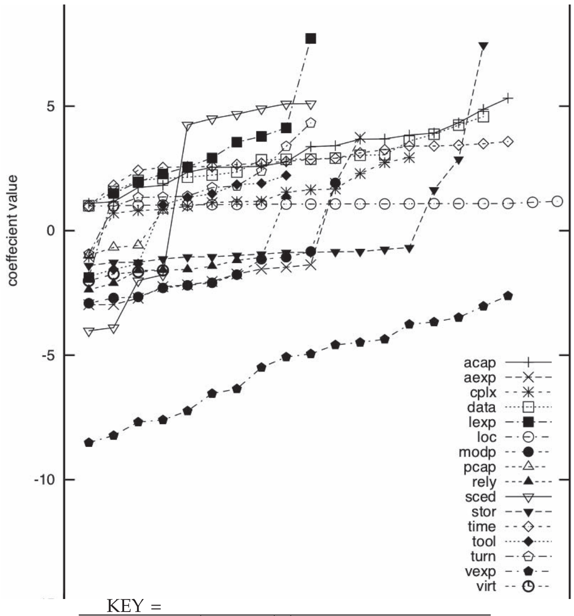
The rest of this paper is structured as follows: First, we explore in Section 2 the literature on defect prediction and effort estimation. This highlights the lack of stable conclusions on what the best way to create predictors is (i.e., how to best learn lessons from the existing data). Section 4 presents two experiments that compare rules for defect prediction and effort estimation generated from global data, from strict local data, and from clusters, whereas Section 3 discusses the details of the clustering and rule learning algorithms used in these experiments. Section 5 discusses the external validity of the conclusions resulting from the two experiments.

2 Literature Review

The main goal of this section is not to provide a generic review of defect prediction and effort estimation work, but to highlight work that documents contradictory results which make it difficult to generalize solutions for defect prediction or effort estimation.

2.1 Effort Estimation

Conclusion instability in effort estimation may be a fundamental property of the datasets we are exploring [11]. For example, Fig. 1 tests the stability of Boehm's COCOMO software development effort estimation model [11]. In that analysis, 20 times, we learned $effort = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots$ using a random $\frac{2}{3}rds$ sample from 93 NASA projects (this subset size was chosen to be similar in size to the 61 projects used to find the original COCOMO coefficients). As shown in Fig. 1, only the coefficient on lines



KEY =
acap = analyst capability
apex = applications experience
cplx = product complexity
data = database size
docu = documentation
lexp = language and tool-set experience
modp = use of modern programing practices
pcap = programmer capability
pcon = personnel continuity
plex = platform experience
pvol = platform volatility
rely = required reliability
ruse = required reuse
sced = dictated development schedule
site = multi-side development
stor = required % of available RAM
time = required % of available CPU
tool = use of software tools
turn = turnaround time
vexp = virtual machine experience
virt = machine volatility

Fig. 1. Sorted β_i values from local calibration on 20^*(66 percent) samples of NASA93 data (from [11]). Coefficients learned using Boehm's recommended methods [12]. A greedy backward selection removed attributes with no impact on estimates (so some attributes have less than 20 results).

of code (loc) was stable. The observed ranges on the other β_i coefficients were very large (e.g., $-8 \leq vexp \leq -3.5$). In fact, the signs of five coefficients even changed from positive to negative (see stor, aexp, modp, cplx, sced). This coefficient instability is particularly troubling because we know of NASA project managers who have made acquisition decisions worth tens of millions of dollars based on the COCOMO coefficients (i.e., they decided to acquire the technologies that had the most impact on the variables with largest coefficients).

Other papers also report contradictory findings about effort estimation. Jorgensen [41] reviewed 15 studies comparing model-based to expert-based methods. Five of those studies favored expert-based methods, five found no difference, and five favored model-based methods. Kitchenham et al. [42] reviewed studies that checked if data imported from other organizations were as useful as local data (for the purposes of building effort models). From a total of seven studies, three found that models from other organizations were not significantly worse than those based on local data, while four found that they were significantly different (and worse). MacDonell and Shepperd [43] also

ref	cbo	rfc	lcom	dit	noc	wmc	#projects	size	type
[13]	+	+	+	-	-	+	6	95-201 classes	6 versions of rhino (java)
[14]	+	+	+	-	-	+	12	86 classess (3-12kloc)	student
[15]	+	+	-				1	1700 classes (110kloc)	commercial telecom
[16]	+	+	-	+	+	+	8	113 classes	student
[17]	+	+	-	+	+	+	8	114 classes	student
[18]	+	+	+	+	-		1	83 classes	commercial: lalo (c++)
[19]				+	+		1	32 classes	commercial: telecom c++
[20]				+	-		1	42-69 classes	commercial java word proc.
[21]	+	-	-	-	-	-	1	85 classes	telecom c++
[22]	-	+		-	-	+	3	92 classes	3 c++ subsystems,commercial
[23]	+	+	+	-	+	+	1	123 classes (34kloc)	java commercial
[24]	+			+		+	1	706 classes	commercial c++ and java
[25]	+	+	+	-	+	+	1	145 classes	kc1-nasa
[26]	+	+	+	+	-	+	1	3677 classes	open source:mozilla
[27]	+	+	+			+	1	?	java (sap) commercial
[28]	+	+	+	+	+	+	3	?	eclipse 2.0, 2.1, 3.0
[29]	-	+	+	-	-	+	8	113 classes	student
[30]		+	+	+	+		2	64 classes	sales and cd-selection system
[31]		-		-	-	-	1	3344 modules (2mloc)	commercial telecom c++
[32]	+	+	+	-	-	+	5	395 classes	commercial telecom c++
[33]	+	+		-	-	+	1	1412 classes	open source:jdt
[34]	+	+		-	-	+	2	9713 classes	eclipse 2.0, 2.1
[35]	+	+	-	-	-	+	1	145 classes	kc1-nasa
[36]				+	-		1	145 classes	commercial java xml editor
[37]	-	-	-	-	-	-	1	174 classes	commercial telecom c++
[38]	-					-	0	50 classes	student
[39]	+	+	-	-	-	+	1	145 classes	kc1-nasa
[40]		+		+	+		2	294 classes	commercial c++
total +	18	20	11	11	8	17			
total -	4	3	7	14	16	4			

Total percents: "*" denotes majority conclusion in each column

	cbo	rfc	lcom	dit	noc	wmc
+	* 64%	* 71%	* 39%	39%	29%	* 61%
-	14%	11%	25%	* 50%	* 57%	14%

KEY:
- Strong consensus (over 2/3rds)
- Some consensus (less than 2/3rds)
- Weak consensus (about half)
- No consensus

Fig. 2. Contradictory conclusions from OO-metrics studies for defect prediction. Studies report significant ("+") or irrelevant ("-") metrics verified by univariate prediction models. Blank entries indicate that the corresponding metric is not evaluated in that particular study. Colors comment on the most frequent conclusion of each column. CBO = coupling between objects; RFC = response for class (#methods executed by arriving messages); LCOM = lack of cohesion (pairs of methods referencing one instance variable, different definitions of LCOM are aggregated); NOC = number of children (immediate subclasses); WMC = #methods per class.

performed a review on the value of local versus global effort estimation models through a replication of [42]. From a total of 10 studies, two were found to be inconclusive, three supported global models, and five supported local models. Similarly, Mair and Shepperd [44] compared regression to analogy methods for effort estimation and found conflicting evidence. From a total of 20 empirical studies, seven favored regression, four were indifferent, and nine favored analogy.

2.2 Defect Prediction

In the area of defect prediction, there are also many contradictory findings. For example, Zimmermann et al. [45] learned defect predictors from 622 pairs of projects $\langle project_1, project_2 \rangle$. In only 4 percent of pairs did the defect predictors learned in $project_1$ work in $project_2$. Similar findings (of contradictory conclusions) concern the OO metrics as well. Fig. 2 lists 28 studies that offer contradictory conclusions regarding the effectiveness of OO metrics for predicting defects (exception: response for class is often a useful indicator of defects). To create Fig. 2, we:

1. Used our domain knowledge to pick three high-impact seed articles [19], [20], [21].
2. Used Google Scholar[1] to find 500+ relevant studies that cited any of those seed articles.
3. Removed false positives by scanning titles and abstracts. This reduced the 500+ articles to 86.
4. Applied the following relevancy rule to reduce the 86 studies to 28: Reject all papers that do not offer a univariate predictive analysis for the validation of the metric(s) under investigation.
5. Checked the literature reviews of important papers in this field [38], [46] for papers not in our sample.

For the manager of a software project Fig. 2 is particularly troubling. Each study makes a clear, but usually different, conclusion. Hence, it is difficult for a manager to make a clear decision about, for example, the merits of a proposed coding standard, where maximum depth of inheritance is required to be less than some expert-specified threshold.

As to the root cause of the instabilities of Fig. 2, we offer the following conjecture. We showed above in Section 2.1 that models learned from different regions within effort data can have very different properties. If defect data were as varied as effort data, then we would naturally expect that different samples of different projects would yield different models (e.g., as seen in Fig. 2) due to *dataset shift* [47].

If this conjecture is correct, then we would expect that clusters within the data should produce different models. This paper is a test of that conjecture. In short, we show that different regions of the data generate different models. Further, the models built from specialized regions within the dataset *perform better than those learned across all data*.

1. http://scholar.google.com.

	dimension	attribute	project1	project2	project3
independent	1	afp	1587	260	152
	2	input	774	9	5
	3	output	260	4	3
	4	enquiry	340	3	2
	5	file	128	193	42
	6	interface	0	41	35
	7	added	1502	51	16
	8	changed	0	138	0
	9	deleted	0	61	0
	10	pdr_afp	4.7	16	4.4
	11	pdr_ufp	5	16.6	4.1
	12	npdr_afp	4.7	16	4.4
	13	npdu_ufp	5	16.6	4.1
	14	resource	4	2	1
	15	dev.type	0	0	0
	16	duration	4	17	9
dependent	1	effort	7490	5150	668

Fig. 3. Example data. Three examples from the CHINA effort estimation dataset.

	dimension	attribute	project1	project2	project3
independent	1	afp	hi	lo	lo
	2	input	hi	lo	lo
	3	output	hi	lo	lo
	4	enquiry	hi	lo	lo
	5	file	hi	hi	lo
	6	interface	lo	hi	hi
	7	added	hi	lo	lo
	8	changed	lo	hi	lo
	9	deleted	lo	hi	lo
	10	pdr_afp	lo	hi	lo
	11	pdr_ufp	lo	hi	lo
	12	npdr_afp	lo	hi	lo
	13	npdu_ufp	lo	hi	lo
	14	resource	hi	lo	lo
	15	dev.type	lo	lo	lo
	16	duration	lo	hi	lo
dependent	1	effort	hi	hi	lo

Fig. 4. Example data of Fig. 3, all data categorized as "hi" or "lo" depending on whether it is above or below the mean value for each row, respectively.

3 Inside the Learners

Before we describe our experiments, this section reviews the technical details of the clustering algorithm and the rule learner used in these experiments. Note that our algorithms are agnostic with respect to the semantics of their input data, which means that we use the same algorithms for defect data and effort data.

To better relate the algorithms to our context (i.e., defect prediction and effort estimation) and to better understand the experiments of the subsequent sections, Figs. 3 and 4 provide examples of instances from one of the actual effort estimation datasets used in the experiments. In this dataset, each project is a point in a 16-dimensional space of "independent" attributes. Each point also has one "dependent" attribute (in Fig. 3 it is the development effort associated with each project). The attribute names in this figure concern function points of these projects and are defined in Fig. 5.

Fig. 4 shows the same data as Fig. 3, but each value has been categorized into "hi" or "lo," depending on whether it falls above or below the mean value for each row. This figure classifies our projects into two "hi" effort projects (project1 and project2) and one "lo" effort project (project3).

Contrast sets list the differences between classes. To find a contrast set, we look for an attribute where 1) all the values from one class are similar, but 2) those values differ in different classes. The only such contrast set in Fig. 4 is the row marked in gray which denotes the function points of internal logical files. Using this contrast set, we would say that "lo" file function points is the factor that most selects for low effort projects.

While the rules generated with contrast sets are simple, they are quite powerful and our choice is not accidental. We have previously conducted extensive evaluations of contrast set learning with other learners on SE data [48]. We found that these succinct rules outperformed more complex models generated by standard classifier or optimization algorithms [49]. Also, using stochastic sampling, contrast set learning can process very large datasets in linear time,

afp	adjusted function points	adjusted size by the standard value adjustment factor (vaf)
input	function points (ufp) of input	
output	function points (ufp) of external output	
enquiry	function points (ufp) of external enquiry	
file	function points (ufp) of internal logical files or entity references	
interface	function points (ufp) of external interface added	function points (ufp) of new or added functions
changed	function points (cfp) of changed functions	
deleted	function points (cfp) of deleted functions	
pdr_ufp	normalized level 1 productivity delivery rate	norm. level 1 effort (for development team) divided by functional size (unadjusted function points).
npdr_afp	normalized productivity delivery rate	normalized effort divided by functional size (unadjusted function points).
npdu_ufp	productivity delivery rate (adjusted function points)	summary work effort divided by adjusted function point count.
resource	team type	1 = development team effort (e.g., project team, project management); 2 = development team support (e.g., database administration, data administration, quality assurance); 3 = computer operations involvement (e.g., information center support, computer operators, network administration); 4 = end users or clients (e.g., user liaisons, user training time)
dev.type	development type	1= new development, 2= enhancement; 3= redevelopment.
duration	total elapsed time for the project	in calendar months.
effort	summary work effort	provides the total effort in hours recorded against the project.

Fig. 5. Function point measures used in the CHINA effort estimation dataset. The last line shows the dependent attribute (total effort in hours).

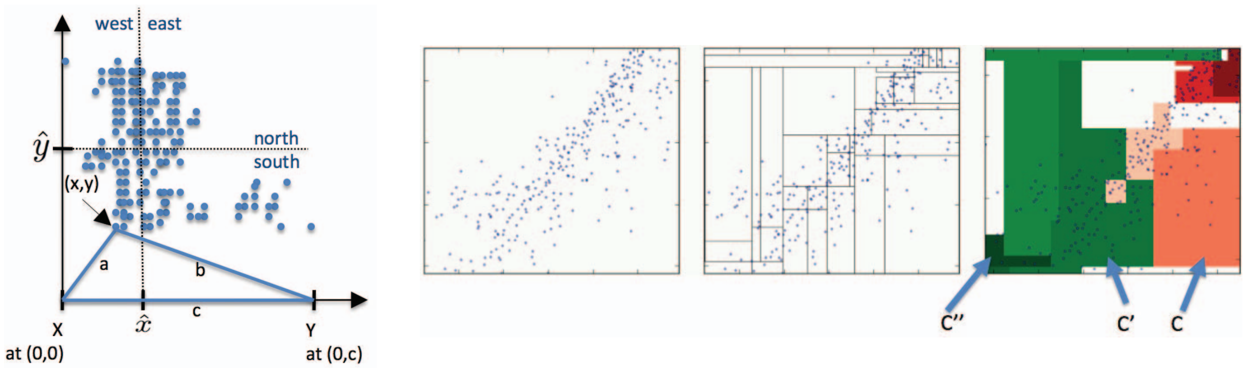


Figure 6.a: *all data presented in 2 dimensions found by FASTMAP.* **Figure 6.b**: *raw data* **Figure 6.c**: *FASTMAP's leaf clusters* **Figure 6.d**: *after grid clustering*

Fig. 6. Each dot is a D-dimensional instance from the CHINA dataset mapped into two dimensions (for an example of three such dots, see Fig. 3). From that data, a new dimension is synthesized on a line between X (at the origin) and the most remote instance Y (at $0, c$)—see Fig. 6a . Each dot has distance a from the origin and b from the most remote point. The median points on the x- and y-axes are $\hat{x}$ and $\hat{y}$, respectively. The algorithm then recurses on each quadrant to generate grids. Leaf pruning then combines the smaller clusters into the colored regions shown in Fig. 6d.

which is an advantage when the datasets are large. Further, a major advantage of contrast set learning for SE is that it generates very succinct rules. For example, the single rule generated by this example is

$$if\ file = lo\ then\ effort = lo.$$

In [50], we argued that such brevity was important when explaining data mining results to business users and we prefer such rules (when possible) over more complex ones, which tend to be harder to understand.

To test this rule, we run a *selection study*, where we look at the class distribution of all projects where $file = lo$. In this case, 100 percent of projects with $file = lo$ have "lo" effort. Note that for this test to be valid, it should be conducted on data not used for learning the rule.

3.1 Clustering Data with WHERE

Our clustering algorithm, named WHERE, assumes that *the dimensions of most interest are the dimensions of greatest variability*. This assumption is shared by other researchers, such as those using feature weighting based on variance [51] or principal component analysis (PCA), e.g., [4].

Matrix factorization methods like PCA take polynomial time to execute [52]. We focus in this paper on defect and effort data, but our long term view is that these techniques can be used for other types of SE data. Hence, we adopt a more efficient solution for our tools. Faloutsos and Lin [53] offer a linear-time heuristic for generating these dimensions, which we use in our work. Given N instances, their "FASTMAP" heuristic finds the dimension of greatest variability to a line drawn between the two furthest points. These two points are found in linear time, as follows: *First* select any instance Z at random, *second* find the instance X that is furthest away from Z, *third* find the instance Y that is furthest away from X. The line $\overline{XY}$ is an approximation to the first component found by PCA.

As shown in Fig. 6a, an orthogonal dimension to $\overline{XY}$ can be found by declaring that the line $\overline{XY}$ is of length c and runs from point $(0, 0)$ to $(0, c)$. Each instance now has a distance a to the origin (instance X) and distance b to the most remote point (instance Y). From the Pythagorean Theorem and cosine rule, each instance is at the point $x = (a^2 + c^2 - b^2)/(2c)$ and $y = \sqrt{a^2 - x^2}$. Fig. 6a shows four quadrants defined by the median values of each dimension $(\hat{x}, \hat{y})$: *NorthWest, NorthEast, SouthWest, and SouthEast.*

WHERE constructs Fig. 6a using a standard euclidean distance operator, then it recurses on each quadrant. This generates a balanced tree of quadrants, stopping when a subquadrant has less than a *minimum number instances* (currently the square root of the total number of instances). The resulting tree of quadrants is then pruned from the leaves back to the root: Leaf quadrants with *similar density* (currently, within 50 percent) are grouped together.

Fig. 6b shows the CHINA effort estimation dataset from the PROMISE repository mapped onto the axes found by FASTMAP. Each dot describes one project using multiple independent attributes and one dependent attribute showing the development effort (in months). For an example of one of those dots, see Fig. 3.

Fig. 6c shows the leaf quadrants found by WHERE's recursive exploration of the NorthWest, NorthEast, SouthWest, and SouthEast quadrants.

Fig. 6d shows the results of leaf pruning: Those clusters are colored to show the median intracluster development effort (dark red = highest effort while dark green = lowest effort).

Now consider the three clusters labeled C, C', C''. Suppose a manager of a project in the orange cluster C is considering how to decrease the development effort of that project (of all the neighbors of that cluster, the green cluster C' has the lowest development effort). Accordingly, that manager would learn rules over the C' data to find treatments that convert projects of type C to C' (note that such a strategy is *not* available to the manager of projects in the dark green cluster C'' because no neighbor of C'' has a shorter development effort, so there we would advise to maintain the status quo).

3.2 Learning Rules with WHICH

A vital requirement for this work is that whatever data miner is used, it can generate rules that can be compared

About	System	Description	Size
effort	CHINA	Function points	499 projects
	NASACOC	COC81 + NASA93	156 projects
defects	JEDIT	Text editor	306 classes
	LUCENE	Text search engine	428 classes
	SYNAPSE	Enterprise service bus	256 classes
	TOMCAT	Apache servlet container	858 classes
	VELOCITY	Template language engine	229 classes
	XALAN	XSLT processor	875 classes
	XERCES	XML processor	329 classes

Fig. 7. Data from http://promisedata.googlecode.com/ for this study.

with the general truisms in the field. Therefore, we eschew learners that use statistical distributions and probability calculations to generate models which, even if they work successfully, are opaque to a human reader. Hence, we do not use Bayes classifiers [54] or neural networks [55]. For the same reason, we also choose not to use learners that generate numeric combinations of project influences, e.g., linear regression, logistic regression, simulated annealing [56], model trees [57], or support vector machines. Finally, we avoid learners that produce large and hard to read theories, e.g., genetic programming algorithms [58] that can learn large and intricate rules.

For this research, we use the WHICH contrast rule learner [59]. WHICH was informally introduced at the start of Section 3. More formally, we say that WHICH learns rules of the form

$$if\ R_x\ then(change = \epsilon_1/\epsilon_0 * support).$$

Here, R_x is a *treatment* containing a set of attribute value pairs a_v; ϵ_0 is the median score for instances in the untreated population. Referring back to Fig. 4, we say that the untreated population is all the test data (i.e., all of Fig. 4) and the treated population are the rows found by the selection study (i.e., all the examples that do not conflict with the treatment). ϵ_1 is the median score for the population subset selected by the rule. For effort and defect prediction, the ratio ϵ_1/ϵ_0 is smaller if the treatment selects for *better* instances. For an example of such a rule, see the start of this section.

WHICH builds these rules by looping over attribute value combinations, combining those that look most promising (for the AI-literate reader, we note that WHICH is a fuzzy beam search). Continuous attributes are first discretized to a few discrete values. A stack is created containing one item for every attribute value. The items in that stack are sorted using $\epsilon_1/\epsilon_0 * support$ (where *support* is the percent of the data selected by that rule). WHICH generates new rules as follows: Several times, 1) select two items at random, favoring those with better $\epsilon_1/\epsilon_0 * support$; 2) combine the pair into a new item and score it. The new rules are then sorted back into the stack. This process repeats until no new improvements are seen at the top-of-stack. WHICH returns the rule at the top-of-stack.

4 Experiments

We ran a set of experiments that compare the results of defect prediction and effort estimation obtained from learning rules from global data, local data, and clusters, respectively.

The objects of our experiments are the nine datasets shown in Fig. 7. Each of the nine datasets used in the experiments was scored with their project effort or number of defects. There are two datasets used for effort estimation and seven used for defect prediction. The projects have three sets of attributes:

1. NASACOC contains the independent attributes from Fig. 1. Its dependent attribute is development effort, measured in terms of calendar months (at 152 hours per month, including development and management hours).
2. CHINA contains the independent attributes of Fig. 5 and the dependent attribute shown in the last row of that table (summary work effort).
3. The other seven datasets contain the independent attributes of Fig. 8 and the dependent attribute of defect counts (as seen in a postrelease bug tracking system).

NASACOC and CHINA are effort estimation datasets, while the other seven are defect prediction datasets. Thanks to the researchers who shared the data via the PROMISE repository [60], we were able to experiment with a diverse set of data which remains available for future replications.

The main research question we are addressing is how to generate lessons that lead to rules for minimizing effort and defects. On one hand, we generated lessons from global data and applied them to individual project data. On the other hand, we clustered the data and for each cluster, we applied lessons from the best neighboring cluster (i.e., with better defect or effort values). We present an informal example, followed by the formal description of the treatments we chose.

Assume that we have data from two sources (e.g., effort or defect data from different companies) A and B:

- Dataset $\mathrm{A} = \{\mathrm{Xa}, \mathrm{Ya}, \mathrm{Za}\}$
- Dataset $\mathrm{B} = \{\mathrm{Xb}, \mathrm{Yb}, \mathrm{Zb}\}$.

We combine the data $\{\mathrm{Xa}, \mathrm{Ya}, \mathrm{Xb}, \mathrm{Yb}, \mathrm{Za}, \mathrm{Zb}\}$ and cluster it. We obtain three clusters: $\mathrm{C} = \mathrm{Xa}, \mathrm{Xb}$, $\mathrm{C}' = \mathrm{Ya}, \mathrm{Yb}$, and $\mathrm{C}'' = \mathrm{Za}, \mathrm{Zb}$. We want to see what treatments to each cluster of data result in lowering the defect (or effort) values. The question we are addressing is how to infer the rules (using WHICH). We have the following options:

1. *Global learning*. For cluster C, WHICH learns the rules from all the data (minus what is in the cluster—$\{\mathrm{Ya}, \mathrm{Yb}, \mathrm{Za}, \mathrm{Zb}\}$) and test it on the data in C, i.e., $\{\mathrm{Xa}, \mathrm{Xb}\}$.
2. *Cluster learning*. For cluster C, WHICH learns the rules on the data from cluster C', i.e., $\{\mathrm{Ya}, \mathrm{Yb}\}$, (we assume here that C' is the neighbor cluster to C with the best defect/effort values) and we test it on the data in C, i.e., $\{\mathrm{Xa}, \mathrm{Xb}\}$. We further refine the cluster learning:

   a. *Neighbor learning (cross)*. For the data in C from one source (e.g., Xa), WHICH learns on the data from C' that does not come from the same source (i.e., Yb in this example). Note that this is an overfitting-avoidance strategy because it somewhat increases the difference of the training data from the test data. More details on this issue are provided later in this section.
   b. *Local learning (within)*. For the data in C from one source (e.g., Xa), WHICH learns on the data

amc	average method complexity	e.g. number of JAVA byte codes
avg_cc	average McCabe	average McCabe's cyclomatic complexity seen in class
ca	afferent couplings	how many other classes use the specific class.
cam	cohesion amongst classes	summation of number of different types of method parameters in every method divided by a multiplication of number of different method parameter types in whole class and number of methods.
cbm	coupling between methods	total number of new/redefined methods to which all the inherited methods are coupled
cbo	coupling between objects	increased when the methods of one class access services of another.
ce	efferent couplings	how many other classes is used by the specific class.
dam	data access	ratio of the number of private (protected) attributes to the total number of attributes
dit	depth of inheritance tree	
ic	inheritance coupling	number of parent classes to which a given class is coupled (includes counts of methods and variables inherited)
lcom	lack of cohesion in methods	number of pairs of methods that do not share a reference to an instance variable.
locm3	another lack of cohesion measure	if m, a are the number of $methods, attributes$ in a class number and $\mu(a)$ is the number of methods accessing an attribute, then $lcom3 = ((\frac{1}{a}\sum_j^a \mu(a_j)) - m)/(1-m)$.
loc	lines of code	
max_cc	maximum McCabe	maximum McCabe's cyclomatic complexity seen in class
mfa	functional abstraction	number of methods inherited by a class plus number of methods accessible by member methods of the class
moa	aggregation	count of the number of data declarations (class fields) whose types are user defined classes
noc	number of children	
npm	number of public methods	
rfc	response for a class	number of methods invoked in response to a message to the object.
wmc	weighted methods per class	
defects	defects	number of defects per class, seen in post-release bug-tracking systems.

Fig. 8. OO measures used in the LUCENE, XALAN, JEDIT, VELOCITY, SYNAPSE, TOMCAT, and XERCES defect datasets. The last line shows the dependent attribute (defects reported to a postrelease bug-tracking system).

from C′ that also comes from the same source (i.e., Ya in this example).

We conducted the experiment in two stages: one for steps 1 and 2 above and one for steps 2a and 2b. Before we describe the experiments more formally, we need to introduce some notation:

- Let *all* refer to all examples in C.
- Let *treated* refer to the examples of C selected by the rules predicting for lower class values ($treated \subseteq all$). In the case of the defect datasets, the class is "number of defects" while in the case of the effort datasets, the class is "development effort." Note that in both cases, we wish to *minimize* the class value.
- Let $D_{treated}$ be the distribution of class variables seen in *treated*.
- Let D_{all} be the distribution of the class variables seen in *all* and *max* be the maximum value of that distribution (i.e., the worst-case result). Normalize all values in D_{all} by expressing them as a percent of the *max* worst case. For example, in $D_{treated}$, a value of 50 would denote a class value that is half of the maximum value seen in the raw data D_{all}.
- Find the 25th, 50th, 75th, 100th percentile normalized value in $D_{treated}$ and D_{all}. For both distributions, let the *median*, *stability*, and *worst case* values to be the 50th, 75th-25th, and 100th percentile values, respectively.

Formally, in each case, the following steps were done:

1. Combine all the data from all sources; e.g., for defect reduction, combine together data from S_1 = JEDIT , S_2 = XERCES, S_3 = XALAN, and so on.
2. Cluster the combined data using WHERE. Note that each cluster may now contain examples from multiple sources.
3. For each cluster C, find C' (the neighboring cluster with the best median score of its dependent variable).
4. Apply WHICH to C' to learn some rules. Then apply those learned rules, i.e., that predicts for lower defects, to C.
5. Report the *median*, *stability*, and *worst case* values in *all* and *some*.

For experiments 2a and 2b, step 3 becomes:

3ab. For data in C from source S_i, find examples in C' from other sources S_j, where $i \neq j$.

For experiment 2a, step 4 becomes:

4a. Apply WHICH to the S_j data from C'. In this experiment, the S_i examples in C are the *all* set while *some* are the examples selected by the rules learned from other sources S_j in C'.

For the experiment 2b, step 4 becomes:

4b. The S_i examples in C are treated with the rules learned from the same sources S_i in C'.

Next, we discuss the details of the experiments separately along with their results.

4.1 Experiment #1: Global Rules Are Suboptimal

In the first experiment, we compared *global lessons* with *cluster lessons*. We used WHERE to cluster all the available data separately for effort estimation and defect prediction. Our clustering algorithm takes care to skip the dependent variable during clustering. That is, when computing the

	effort		defect						
cluster	NASACOC	CHINA	LUCENE	XALAN	JEDIT	VELOCITY	SYNAPSE	TOMCAT	XERCES
global	kloc=1	afp=1	rfc=2	loc=1	rfc=2	cam=7	amc=1	loc=2	cbo=1
C0									
C1	rely=n	added=4	amc=7	amc=1	ic=7	noc=1	dit=4	cbm=1	dit=1
C2	prec=h	deleted=1	ca=1	cam=2	noc=1	dam=1 or 5		dam=1	dam=1
C3		deleted=1	dam=5	cam=3	amc=6	avg_cc=4		noc=1	ca=1 or 7
C4			mfa=1	dit=2 or 4	noc=1	moa=1		rfc=5	**cbo=1**
C5			moa=1	**loc=1**				lcom3=5	
C6				**loc =1 or 2**				max_cc=1	
C7				moa=1				cbm=1	

Fig. 9. *Global* and *cluster* rules for each cluster C_i. In this study, all the numerics were discretized into seven equal frequency bins so, e.g., $\mathrm{kloc} = 1$ should be read as "set kloc to its minimal value." The **underlined bold entries** denote the rules that were the same globally as at cluster level. Note that there are very few rules that are the same globally as at cluster level.

distance between examples (i.e., projects for effort estimation and classes for defect prediction), the defect or effort values in those examples are ignored. The dependent variable is used only after clustering.

Once WHERE divides the data, WHICH is applied to each cluster C to find the $cluster$ rules for selecting lower defect/effort values. Next, the dependent values are used to score each cluster (median defect/effort values of projects in that cluster). For each cluster C, we search its neighbors for the cluster $C' \neq C \wedge adjacent(C, C') \wedge (score(C') < score(C))$ with the best score (lowest median defect/effort values). The rule $C'.cluster$ is then applied to C to find $treated \subseteq C$. See above the definition for $treated$ and *score*.

For every cluster C, we compared the two treated sets:

- one using $C'.cluster$ and
- the other using rules generated globally across all the data (by applying WHICH to all the datasets, rather than to a single cluster).

We found that the rules generated from cluster lessons were better and different than those learned globally. Fig. 9 shows the rules learned in different datasets. Each dataset generated between two (*SYNAPSE*) to eight (*XALAN*) clusters. One cluster always had the best score (lowest effort or defects) and this cluster was labeled $C0$. No rules were learned for this cluster because our recommendation for projects in $C0$ is to "maintain the status quo"—i.e., we do not know how to improve the median value of the dependent variable (defect or effort) for this cluster, given the current data. Lines $C_i \in C1..C7$ show the cluster rules learned from C_i's best neighbor. The underlined cluster rules are those that are same as the global rules (these appear in the results for XALAN and XERCES). Note that there are very few cluster rule sets that are the same as the global rules.

The effects of applying these rules are shown in Fig. 10. These results are expressed in terms of percentile bands: *median*, *stability*, and *worst case*. All values are expressed as ratios of maximum values seen in the untreated dataset (e.g., "50" means the middle value of the untreated data).

The first thing to note in Fig. 10 is that our rule learning method is effective: The median and stability values of the above are small percentages of original data. Indeed, in five of the seven defect prediction experiments, the cluster rules selected projects with zero defects.

The second important conclusion of Fig. 10 is that the cluster rules performed better than the global rules:

- The *median* performance of cluster rules is significantly better than the median performance of the global rules (Wilcoxon, 95 percent confidence).
- The *stability* around the median is greater with cluster rules than with global rules, i.e., if we use the global rules, then we will be *less confident* in our predictions on the effects of those rules.
- The *worst case* results with global rules are far worse than the worst case results of cluster rules. In all cases, the worst case performance of the global rules is the same as the untreated data (see the second last row of the table). On the other hand, in seven results, the worst case performance of the cluster rules is one-third or less of the maximum in the untreated data.

Why is the performance of the global rules suboptimal? Our hypothesis is that software construction is such a diverse task that any global conclusion that holds *across all*

		effort		defect							
		NASACOC	CHINA	LUCENE	XALAN	JEDIT	SYNAPSE	VELOCITY	TOMCAT	XERCES	$\frac{\sum global}{\sum cluster}$
median	global	17	4	3	12	0	0	0	0	0	0.64
(50th percentile)	cluster	7	3	0	12	0	0	0	0	1	
stability	global	16	7	10	12	0	11	8	0	1	0.37
(75th-25th percentile)	cluster	6	6	3	0	0	0	8	0	1	
worst-case	global	100	100	100	100	100	100	100	100	100	0.39
(100th percentile)	cluster	9	100	23	62	8	33	50	33	32	

Fig. 10. Global versus cluster reasoning. In this figure, *smaller* values are *better*. All values are percentages of the maximum effort or defect values seen in the untreated datasets so, e.g., a median global value of "17" in NASACOC is 17 percent of the maximum effort value in the NASACOC datasets.

		VELOCITY1.4 POI1.5, ANT1.6	**poi-3.0**, VELOCITY1.6, XERCES1.3	**ANT1.5**, XERCES1.4, VELOCITY1.4	**POI2.5**, JEDIT3.2, LUCENE2.4	**ANT1.4**, IVY1.4, XERCES1.2	**IVY1.4**, ANT1.4, JEDIT4.3	**POI3.0**, JEDIT4.0, ivy-1.4	**POI2.5**, XERCES1.3, ANT1.5	$\frac{\sum cross}{\sum within}$
median	within	5	0	2	4	0	0	0	0	0.4
(50th percentile)	cross	0	0	2	2	0	0	0	0	
stability	within	4	0	2	0	0	0	0	0	1.3
(75th-25th percentile)	cross	0	0	0	0	0	0	4	8	
worst case	within	10	32	15	16	33	67	9	23	0.3
(100th percentile)	cross	0	11	2	11	33	0	4	3	

Fig. 11. Experiment 2. Using groups of three projects, S_j (the test cluster) is picked at random (marked in **bold underline**).

projects may be somewhat different from the conclusions learned from *individual projects*.

In summary, the data we obtained are not supportive of the claim that global lessons are the best for defect prediction or effort estimation. However, it would be a mistake to conclude that this experiment supports delphi localisms. As shown in our next experiment, the best way to divide the data is not via delphi localizations (into, say, just the JEDIT or LUCENE datasets) but by automatic localizations that cross dataset boundaries.

4.2 Experiment #2: Local Rules Are Suboptimal

If a data miner is overly zealous, then it could read too much into the training data. In this situation, the learner may *overfit* its models to spurious details in the training data. Hence, many data mining algorithms employ overfitting-avoidance strategies to prune away needless elaborations. For example:

- Early versions of decision tree learners produced very big and bushy decision trees. One of the key innovations of C4.5 [61] was *leaf pruning*, which prunes leaves until the error rate in the pruned tree starts increasing.
- After the INDUCT rule learner builds a rule condition, it runs a greedy back-select algorithm that checks if any condition can be removed while preserving the accuracy of that rule [62].

Accordingly, the following experiment checked if WHERE and WHICH can be improved by an overfitting-avoidance strategy. Specifically, we selected the data used for training in each cluster, based on which source they come from (as explained in the beginning of this section—see step 3ab). This resulted in two ways of learning the rules.

We call the steps selecting data from different sources the *cross* treatment because it ensures that data from one source are treated with rules learned from data from other sources. We call the generated rules *neighbor rules* because the source of the data is not random as it comes from a neighboring cluster. The second treatment is referred to as *within* treatment and the rules are called *local rules*, as the data are treated with the rules learned from the same sources (caveat: provided that the data falls into separate neighboring C, C' clusters). It is important to note that in both treatments, rules are applied to the same subset S_i of C and that these rules select only some subset of that data.

As before, our results are expressed in terms of *median*, *stability*, and *worst case*. We express those percentiles as a ratio of the maximum value seen in the untreated S_i data of C (i.e., the maximum value seen before any rule selects some subset). That is to say, all our results are expressed as values ranging from 0 to 100 percent.

The results are shown in Fig. 11. To ensure external validity, the above procedure is repeated multiple times. The results of each repetition are shown in different columns of Fig. 11. Each repetition combines together the data from three sources. Within each group of three, one source is the designated test cluster S_j (again, selected at random). While the datasets were picked at random from the PROMISE data repository, all had to conform to the same ontology (in practice, that meant that Experiment #2 was conducted on datasets of the OO ontology of Fig. 8 since this is the most frequently shared ontology in PROMISE).

One issue that arises in these results is that the *within* treatment is so effective that it is sometimes hard to distinguish further improvements. For example, at first glance, the *stability* results of *cross* seem worse than *within*. However, the raw values for *stability* are so small that a single value (the "8" in the last column) can throw off the results. The same effect (that the baseline *within* results are very small) also confuses an analysis of the *median* results.

However, when we turn to the *worst case* results, the numbers are large enough to allow for a differentiation of the *within* and *cross* results. We observe that the *worst case* results of the *cross* study are much better than the *within*.

In summary, based on an analysis of rules learned from neighboring clusters in different sources, we conclude that it is suboptimal to learn purely local rules from the clusters within the same source as the test data. This is the standard overfitting-avoidance result [54]: It is best not to learn too much from local data. Rather, it is better for a learner to step back a little and train from related concepts (rather than concepts that are too similar).

We conjecture that the diverse nature of software construction means that even for projects built within the same organization, it is more useful to chase external data sources than just to use the local historical data. However, when using data from other sources, it is best to cluster that data and just use a small portion of data from the nearest cluster.

5 External Validity

External validity is the issue of the generality of the conclusions of a study to data not used in that particular study. Within the specific context of effort estimation and defect prediction, the central claim of this paper is that any standard discussion on external validity cannot be trusted for SE data. That is, the methodology of this paper precludes a claim of external validity about specific conclusions (e.g., the relationship of inheritance depth to defects).

But this paper is not a counsel of despair. An essential feature of our work is that the same algorithms were used to generate recommendations for *both* defect reduction *and* effort reduction. This makes this paper somewhat unique since, in the literature, effort estimation and defect prediction are usually explored by different research teams[2] and techniques. That is:

- While this paper has shown that any *specific conclusion* about reducing effort or defects may not be externally valid (since they are project dependent) ...
- ... our externally valid *metaconclusion* is that there are general techniques (i.e., WHERE+WHICH) for finding local conclusions in different projects.

In other words, while we do not provide general lessons that work for defect prediction or effort estimation on any project, we provide instead a technique (using WHERE and WHICH) that can be used on any project with the available data.

To disprove the external validity of this metaconclusion, a research team would need to demonstrate the stability of conclusions across multiple projects. We would propose one sanity check for that demonstration: The project-independent conclusion must be "significant."

For examples of less-than-significant conclusions, we refer the reader to the *global* results shown in the first line of Fig. 9. Here, we read that effort or defects can be reduced by:

- Minimizing lines of code: See the $kloc = 1$ results for NASACOC and $loc = 1$ for XALAN;
- Minimizing function points: See the $afp = 1$ result from CHINA.

That is, those global conclusions are just trite statements that if programmers write less code, they can do so in less time (and introduce fewer defects). In most development projects, programmers do not have the option of writing less code. What makes WHICH's local conclusions nontrite is that most often they are not about merely reducing the size of a system. Rather, as seen in Fig. 9, they are about system reorganization (such as reorganizing the class hierarchy).

Note that, in terms of the current literature, current results support the external validity of the conclusions of this paper. Inspired by our earlier work [10], Bettenburg et al. [6] recently repeated our experiment #1 using different clustering tools. That study found the same conclusion as ours, i.e., that cluster rules do better than rules learned across the whole data.

2. Exception—see the work of Martin Shepperd [63], [64] who explores both areas.

6 Conclusion and Future Work

This paper has discussed ways to collect training data for learning lessons from SE projects and to generate rules for reducing the number of defects or the development effort.

At issue here is how the chief information officers should set policies for their projects. Should they demand that all their projects rigorously apply all the advice from the standard SE textbooks or other literature? Or should they devote resources to a "local lessons team" that explores local data to find the best local practices?

The results of this paper strongly endorse the creation of the local lessons team. However, that team should apply automatic algorithms to build clusters from all available data. The best cluster is the one that is nearby (see Experiment #1) but not from the same source as the test data (see Experiment #2). While we experimented only on defect prediction and effort estimation data, our procedure is data agnostic and we believe it would apply to other kinds of data, for other tasks.

More generally, in terms of SE research, the experiments of this paper show that an SE project is an intricate entity that is best described in terms of a complex combination of multidimensional factors. Hence:

- Trite global rules are not sufficient for controlling such complex entities, at least when it comes to defect prediction and effort estimation.
- Neither is it sufficient to characterize the data with simple divisions of data into local contexts. Before researchers attempt to draw lessons from some historical data source, they should:
  - ignore any existing local divisions into multiple sources,
  - cluster across all available data, then
  - restrict the learning of lessons to the clusters from other sources that are nearest to the test data.

As to future work, it is important to check how often in other datasets cluster rules are better than global rules.

Other future work might include improving WHICH and WHERE. WHICH contains numerous design options that deserve closer attention. Like any beam search, WHICH only stores the top ($N = 50$) rules in its stack. Also, prior to building rules, WHICH discretizes numeric data into $B = 7$ equal frequency bins. It is possible that different values of N and B would result in better rules.

More fundamentally, WHICH is one of a large class of contrast set learners. Other candidate contrast set learners which might do better than WHICH are MINWAL [65], STUCCO [66], and others [67].

Another task deserving future research is to explore multigoal optimization. In the results of Fig. 10, we tried to minimize effort in the NASACOC and CHINA datasets while minimizing defects in the remaining seven datasets. A more challenging goal would be to reduce defects and effort at the same time. The search-based SE literature [68] lists many techniques that might be useful in this regard, including tabu search [69], ant algorithms [70], and particle swarm optimization [71].

ACKNOWLEDGMENTS

The work was funded by US National Science Foundation (NSF) grants CCF:1017330 and CCF: 1017263, the Qatar/WVU research grant NPRP 09-12-5-2-470, partly funded by the Finnish Funding Agency for Technology and Innovation (TEKES) under the Cloud Software Program and the Academy of Finland with Grant Decision No. 260871. Also, the research reported in this document was performed in connection with contract/instrument W911QX-10-C-0066 with the US Army Research Laboratory. The views and conclusions contained in this document/presentation are those of the authors and should not be interpreted as presenting the official policies or position, either expressed or implied, of the US Army Research Laboratory or the US Government unless so designated by other authorized documents. Citation of manufacturers or trade names does not constitute an official endorsement or approval of the use thereof.

REFERENCES

[1] C. Jones, *Estimating Software Costs,* second ed. McGraw-Hill, 2007.
[2] B. Boehm, E. Horowitz, R. Madachy, D. Reifer, B.K. Clark, B. Steece, A.W. Brown, S. Chulani, and C. Abts, *Software Cost Estimation with Cocomo II.* Prentice Hall, 2000.
[3] B.A. Kitchenham, T. Dyba, and M. Jorgensen, "Evidence-Based Software Engineering," *Proc. 26th Int'l Conf. Software Eng.,* pp. 273-281, 2004.
[4] N. Nagappan, T. Ball, and A. Zeller, "Mining Metrics to Predict Component Failures," *Proc. 28th Int'l Conf. Software Eng.,* pp. 452-461, 2006.
[5] D. Posnett, V. Filkov, and P. Devanbu, "Ecological Inference in Empirical Software Engineering," *Proc. IEEE/ACM Int'l Conf. Automated Software Eng.,* 2011.
[6] N. Bettenburg, M. Nagappan, and A.E. Hassan, "Think Locally, Act Globally: Improving Defect and Effort Prediction Models," *Proc. Ninth IEEE Working Conf. Mining Software Repositories,* pp. 60-69, 2012.
[7] Y. Yang, L. Xie, Z. He, Q. Li, V. Nguyen, B.W. Boehm, and R. dValerdi, "Local Bias and Its Impacts on the Performance of Parametric Estimation Models," *Proc. Seventh Int'l Conf. Predictive Models in Software Eng.,* 2011.
[8] K. Lum, J. Hihn, and T. Menzies, "Studies in Software Cost Model Behavior: Do We Really Understand Cost Model Performance?" *Proc. ISPA Conf.,* 2006.
[9] T. Menzies, J. Greenwald, and A. Frank, "Data Mining Static Code Attributes to Learn Defect Predictors," *IEEE Trans. Software Eng.,* vol. 33, no. 1, pp. 2-13, Jan. 2007.
[10] T. Menzies, A. Butcher, A. Marcus, T. Zimmermann, and D. Cok, "Local vs Global Models for Effort Estimation and Defect Prediction," *Proc. 26th IEEE/ACM Int'l Conf. Automated Software Eng.,* 2011.
[11] T. Menzies, Z. Chen, D. Port, and J. Hihn, "Simple Software Cost Estimation: Safe or Unsafe?" *Proc. Workshop Predictive Models in Software Eng. Workshop,* 2005.
[12] B. Boehm, *Software Engineering Economics.* Prentice Hall, 1981.
[13] H. Olague, L. Etzkorn, S. Gholston, and S. Quattlebaum, "Empirical Validation of Three Software Metrics Suites to Predict Fault-Proneness of Object-Oriented Classes Developed Using Highly Iterative or Agile Software Development Processes," *IEEE Trans. Software Eng.,* vol. 33, no. 6, pp. 402-419, June 2007.
[14] K. Aggmakarwal, Y. Singh, A. Kaur, and R. Malhotra, "Empirical Analysis for Investigating the Effect of Object-Oriented Metrics on Fault Proneness: A Replicated Case Study," *Software Process: Improvement and Practice,* vol. 14, no. 1, pp. 39-62, Jan. 2009.
[15] E. Arisholm and L. Briand, "Predicting Fault Prone Components in a JAVA Legacy System," *Proc. ACM/IEEE Int'l Symp. Empirical Software Eng.,* pp. 8-17, 2006.
[16] V. Basili, L. Briand, and W. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators," *IEEE Trans. Software Eng.,* vol. 22, no. 10, pp. 751-761, Oct. 1996.
[17] L. Briand, J. Wust, J. Daly, and D.V. Porter, "Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems," *J. Systems and Software,* vol. 51, no. 3, pp. 245-273, 2000.
[18] L. Briand, J. Wust, and H. Lounis, "Replicated Case Studies for Investigating Quality Factors in Object-Oriented Designs," *Empirical Software Eng.,* vol. 6, no. 1, pp. 11-58, 2001.
[19] M. Cartwright and M. Shepperd, "An Empirical Investigation of an Object-Oriented Software System," *IEEE Trans. Software Eng.,* vol. 26, no. 8, pp. 786-796, Aug. 2000.
[20] K.el Emam, W. Melo, and J. Machado, "The Prediction of Faulty Classes Using Object-Oriented Design Metrics," *J. Systems and Software,* vol. 56, no. 1, pp. 63-75, 2001.
[21] K.el Emam, S. Benlarbi, N. Goel, and S. Rai, "A Validation of Object-Oriented Metrics," *Nat'l Research Council of Canada, NRC/ERB,* vol. 1063, 1999.
[22] M. Tang, M. Kao, and M. Chen, "An Empirical Study on Object-Oriented Metrics," *Proc. Sixth Int'l Software Metrics Symp.,* pp. 242-249, 1999.
[23] P. Yu, T. Systa, and H. Muller, "Predicting Fault-Proneness Using OO Metrics an Industrial Case Study," *Proc. Sixth European Conf. Software Maintenance and Reeng.,* pp. 99-107, 2002.
[24] R. Subramanyam and M.S. Krishnan, "Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects," *IEEE Trans. Software Eng.,* vol. 29, no. 4, pp. 297-310, Apr. 2003.
[25] Y. Zhou and H. Leung, "Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults," *IEEE Trans. Software Eng.,* vol. 32, no. 10, pp. 771-789, Oct. 2006.
[26] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction," *IEEE Trans. Software Eng.,* vol. 31, no. 10, pp. 897-910, Oct. 2005.
[27] T. Holschuh, M. Pauser, K. Herzig, T. Zimmermann, R. Premraj, and A. Zeller, "Predicting Defects in SAP Java Code: An Experience Report," *Proc. 31st Int'l Conf. Software Eng.—Companion Volume,* pp. 172-181, 2009.
[28] R. Shatnawi and W. Li, "The Effectiveness of Software Metrics in Identifying Error-Prone Classes in Post-Release Software Evolution Process," *J. Systems and Software,* vol. 81, no. 11, pp. 1868-1882, 2008.
[29] F. Fioravanti and P. Nesi, "A Study on Fault-Proneness Detection of Object-Oriented Systems," *Proc. Fifth European Conf. Software Maintenance and Reeng.,* pp. 121-130, 2001.
[30] M. Thongmak and P. Muenchaisri, "Predicting Faulty Classes Using Design Metrics with Discriminant Analysis," *Proc. Int'l Conf. Software Eng. Research and Practice,* pp. 621-627, 2003.
[31] G. Denaro, L. Lavazza, and M. Pezze, "An Empirical Evaluation of Object Oriented Metrics in Industrial Setting," *Proc. Fifth CaberNet Plenary Workshop,* 2003.
[32] A. Janes, M. Scotto, W. Pedrycz, B. Russo, M. Stefanovic, and G. Succi, "Identification of Defect-Prone Classes in Telecommunication Software Systems Using Design Metrics," *Information Sciences,* vol. 176, no. 24, pp. 3711-3734, 2006.
[33] M. English, C. Exton, I. Rigon, and B. Cleary, "Fault Detection and Prediction in an Open-Source Software Project," *Proc. Fifth Int'l Conf. Predictor Models in Software Eng.,* nos. 1-11, 2009.

[34] R. Shatnawi, "A Quantitative Investigation of the Acceptable Risk Levels of Object-Oriented Metrics in Open-Source Systems," *IEEE Trans. Software Eng.*, vol. 36, no. 2, pp. 216-225, Mar./Apr. 2010.

[35] Y. Singh, A. Kaur, and R. Malhotra, "Empirical Validation of Object-Oriented Metrics for Predicting Fault Proneness Models," *Software Quality J.*, vol. 18, no. 1, pp. 3-35, 2010.

[36] D. Glasberg, K.el Emam, W. Memo, and N. Madhavji, "Validating Object-Oriented Design Metrics on a Commercial JAVA Application," NRC 44146, 2000.

[37] K.el Emam, S. Benlarbi, N. Goel, and S. Rai, "The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics," *IEEE Trans. Software Eng.*, vol. 27, no. 7, pp. 630-650, July 2001.

[38] M. Thapaliyal and G. Verma, "Software Defects and Object Oriented Metrics—An Empirical Analysis," *Int'l J. Computer Applications*, vol. 9/5, p. 41, 2010.

[39] J. Xu, D. Ho, and L. Capretz, "An Empirical Validation of Object-Oriented Design Metrics for Fault Prediction," *J. Computer Science*, pp. 571-577, July 2008.

[40] G. Succi, "Practical Assessment of the Models for Identification of Defect-Prone Classes in Object-Oriented Commercial Systems Using Design Metrics," *J. Systems and Software*, vol. 65, no. 1, pp. 1-12, Jan. 2003.

[41] M. Jorgensen, "A Review of Studies on Expert Estimation of Software Development Effort," *J. Systems and Software*, vol. 70, no. 1/2, pp. 37-60, 2004.

[42] B. Kitchenham, E. Mendes, and G.H. Travassos, "Cross versus Within-Company Cost Estimation Studies: A Systematic Review," *IEEE Trans. Software Eng.*, vol. 33, no. 5, pp. 316-329, May 2007.

[43] S.G. MacDonell and M.J. Shepperd, "Comparing Local and Global Software Effort Estimation Models—Reflections on a Systematic Review," *Proc. First Int'l Symp. Empirical Software Eng. and Measurement*, pp. 401-409, 2007.

[44] C. Mair and M. Shepperd, "The Consistency of Empirical Comparisons of Regression and Analogy-Based Software Project Cost Prediction," *Proc. Int'l Symp. Empirical Software Eng.*, p. 10, 2005.

[45] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-Project Defect Prediction," *Proc. Seventh Joint Meeting European Software Eng. Conf. and ACM SIGSOFT Symp. Foundations of Software Eng.*, Aug. 2009.

[46] L.C. Briand and J. Wuest, "Empirical Studies of Quality Models in Object-Oriented Systems," *Advances in Computers*, vol. 56, pp. 98-167, 2002.

[47] B. Turhan, "On the Dataset Shift Problem in Software Engineering Prediction Models," *Empirical Software Eng.*, vol. 17, pp. 62-74, 2012.

[48] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, "Defect Prediction from Static Code Features: Current Results, Limitations, New Approaches," *Automated Software Eng.*, vol. 17, no. 4, pp. 375-407, Dec. 2010.

[49] G. Gay, T. Menzies, M. Davies, and K. Gundy-Burlet, "Automatically Finding the Control Variables for Complex System Behavior," *Automated Software Eng.*, vol. 17, no. 4, pp. 439-468, Dec. 2010.

[50] T. Menzies and Y. Hu, "Data Mining for Very Busy People," *Computer*, vol. 36, no. 11, pp. 22-29, Nov. 2003.

[51] D. Baker, "A Hybrid Approach to Expert and Model-Based Effort Estimation," master's thesis, Lane Dept. of Computer Science and Electrical Eng., West Virginia Univ., https://eidr.wvu.edu/etd/documentdata.eTD?documentid=5443, 2007.

[52] Q. Du and J.E. Fowler, "Low-Complexity Principal Component Analysis for Hyperspectral Image Compression," *Int'l J. High Performance Computing Applications*, vol. 22, no. 4, pp. 438-448, Nov. 2008.

[53] C. Faloutsos and K.-I. Lin, "Fastmap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 163-174, 1995.

[54] I.H. Witten and E. Frank, *Data Mining*, second ed. Morgan Kaufmann, 2005.

[55] D. Rumelhart, G. Hinton, and R. Williams, "Learning Representations by Back-Propagating Errors," *Nature*, vol. 323, pp. 533-536, 1986.

[56] S. Bouktif, H. Sahraoui, and G. Antoniol, "Simulated Annealing for Improving Software Quality Prediction," *Proc. Eighth Ann. Conf. Genetic and Evolutionary Computation*, pp. 1893-1900, 2006.

[57] J.R. Quinlan, "Learning with Continuous Classes," *Proc. Fifth Australian Joint Conf. Artificial Intelligence*, pp. 343-348, 1992.

[58] G. Boetticher, "When Will It Be Done? The 300 Billion Dollar Question, Machine Learner Answers," *IEEE Intelligent Systems*, vol. 18, no. 3, pp. 48-50, May/June 2003.

[59] E. Kocaguneli, G. Gay, T. Menzies, Y. Yang, and J.W. Keung, "When to Use Data from Other Projects for Effort Estimation," *Proc. IEEE/ACM Int'l Conf. Automated Software Eng.*, 2010.

[60] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan, "The Promise Repository of Empirical Software Engineering Data," http://promisedata.googlecode.com, June 2012.

[61] R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufman, 1992.

[62] B. Gaines and P. Compton, "Induction of Ripple Down Rules," *Proc. Fifth Australian Conf. Artificial Intelligence*, pp. 349-354, 1992.

[63] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, "A General Software Defect-Proneness Prediction Framework," *IEEE Trans. Software Eng.*, vol. 37, no. 3, pp. 356-370, http://dx.doi.org/10.1109/TSE.2010.90, May/June 2011.

[64] M. Jorgensen and M. Shepperd, "A Systematic Review of Software Development Cost Estimation Studies," *IEEE Trans. Software Eng.*, vol. 33, no. 1, pp. 33-53, Jan. 2007.

[65] C. Cai, A. Fu, C. Cheng, and W. Kwong, "Mining Association Rules with Weighted Items," *Proc. Int'l Database Eng. and Applications Symp.*, Aug. 1998.

[66] S. Bay and M. Pazzani, "Detecting Change in Categorical Data: Mining Contrast Sets," *Proc. Fifth Int'l Conf. Knowledge Discovery and Data Mining*, 1999.

[67] P.K. Novak, N. Lavrač, and G.I. Webb, "Supervised Descriptive Rule Discovery: A Unifying Survey of Contrast Set, Emerging Pattern and Subgroup Mining," *J. Machine Learning Research*, vol. 10, pp. 377-403, June 2009.

[68] M. Harman and B. Jones, "Search-Based Software Engineering," *J. Information and Software Technology*, vol. 43, pp. 833-839, Dec. 2001.

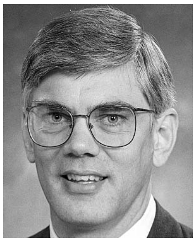
[69] F. Glover and M. Laguna, "Tabu Search," *Modern Heuristic Techniques for Combinatorial Problems*, C. Reeves, ed., Blackwell Scientific Publishing, 1993.

[70] M. Zlochin, M. Birattari, N. Meuleau, and M. Dorigo, "Model-Based Search for Combinatorial Optimization: A Critical Survey," *Annals of Operations Research*, vol. 131, pp. 373-395, 2004.

[71] H. Pan, M. Zheng, and X. Han, "Particle Swarm-Simulated Annealing Fusion Algorithm and Its Application in Function Optimization," *Proc. Int'l Conf. Computer Science and Software Eng.*, pp. 78-81, 2008.

Tim Menzies received the PhD degree from the University of New South Wales. He is a professor in computer science and electrical engineering at West Virginia University (WVU) and the author of more than 200 referred publications. At WVU, he has been a lead researcher on projects for the US NSF, NIJ, DoD, NASA's Office of Safety and Mission Assurance, as well as SBIRs and STTRs with private companies. He teaches data mining and artificial intelligence and programming languages. He is the cofounder of the PROMISE conference series devoted to reproducible experiments in software engineering. He is an associate editor of the *IEEE Transactions on Software Engineering* and the *Automated Software Engineering Journal*. In 2012, he served as a cochair of the program committee for the IEEE Automated Software Engineering conference. He is a member of the IEEE.

Andrew Butcher is an active member of the open source community and a software engineer at Redhat (East Carolina). For further information, see his Git repositories https://github.com/abutcher or his home page http://abutcher.org/.



David Cok graduated with honors from Calvin College with AB degrees in physics and mathematics, and received the AM and PhD degrees in physics from Harvard University. He is an associate vice president of technology at GrammaTech. His career includes research in static analysis with an emphasis on practical application to industrial software development. Before joining GrammaTech, he was a senior research associate and leader of research and product development groups at Eastman Kodak Company. He is a member of the IEEE.

Andrian Marcus received the prior degrees from The University of Memphis, Tennessee, and Babes-Bolyai University, Cluj-Napoca, Romania, respectively, and the PhD degree in computer science from Kent State University. He is an associate professor in the Department of Computer Science at Wayne State University, Detroit, Michigan, where he has been a member of the faculty since 2003. His current research interests are in software engineering, with focus on software evolution and program comprehension. He is best known for his work on using information retrieval and text mining techniques for software analysis to support comprehension during software evolution. He is a member of the Steering Committee of the IEEE International Conference on Software Maintenance and he served as the conference's general chair and program cochair. He is a member of the IEEE.

Lucas Layman received the BS degree in computer science from Loyola College in 2002, and the PhD degree in computer science from North Carolina State University in 2009. His past research includes reliability and productivity prediction at Microsoft Corporation, safety measurement at NASA, and agile process engineering and evaluation at the US Department of Defense, IBM, and Sabre Airline Solutions. In addition to empirical studies on process and process improvement, he has investigated human factors in software development and authored a number of papers on computer science education. In his current position, he is a research scientist at the Fraunhofer Center at the University of Maryland. He is a member of the IEEE.

Forrest Shull is a senior scientist at the Fraunhofer Center at the University of Maryland, a nonprofit research and tech transfer organization, where he leads the Measurement and Knowledge Management Division. At Fraunhofer CESE, he has been a lead researcher on projects for NASA's Office of Safety and Mission Assurance, the NASA Safety Center, the US Department of Defense, the US National Science Foundation, the US Defense Advanced Research Projects Agency, and companies such as Motorola and Fujitsu Labs of America. He is an adjunct professor at the University of Maryland College Park, and the editor-in-chief of *IEEE Software*. He is a member of the IEEE.

Burak Turhan is a Tutkijatohtori (Doctor researcher) in the Department of Information Processing Science, University of Oulu in Finland, and a member of the PROMISE steering committee. For further information, see his website: http://turhanb.net. He is a member of the IEEE.

Thomas Zimmermann received the PhD degree from Saarland University in Germany. He is a researcher in the Research in Software Engineering group at Microsoft Research Redmond, an adjunct assistant professor at the University of Calgary, and an affiliate faculty member at the University of Washington. His research interests include empirical software engineering, mining software repositories, computer games, recommender systems, development tools, and social networking. He is best known for his research on systematic mining of version archives and bug databases to conduct empirical studies and to build tools to support developers and managers.