

Ecological Inference in Empirical Software Engineering

Daryl Posnett

Department of Computer Science
University of California, Davis
Davis, CA
dpposnett@ucdavis.edu

Vladimir Filkov

Department of Computer Science
University of California, Davis
Davis, CA
vfilkov@ucdavis.edu

Premkumar Devanbu

Department of Computer Science
University of California, Davis
Davis, CA
devanbu@ucdavis.edu

Abstract—Software systems are decomposed hierarchically, for example, into modules, packages and files. This hierarchical decomposition has a profound influence on evolvability, maintainability and work assignment. Hierarchical decomposition is thus clearly of central concern for empirical software engineering researchers; but it also poses a quandary. At what level do we study phenomena, such as quality, distribution, collaboration and productivity? At the level of files? packages? or modules? How does the level of study affect the truth, meaning, and relevance of the findings? In other fields it has been found that choosing the wrong level might lead to misleading or fallacious results. Choosing a proper level, for study, is thus vitally important for empirical software engineering research; but this issue hasn't thus far been explicitly investigated. We describe the related idea of ecological inference and ecological fallacy from sociology and epidemiology, and explore its relevance to empirical software engineering; we also presents some case studies, using defect and process data from 18 open source projects to illustrate the risks of modeling at an aggregation level in the context of defect prediction, as well as in hypothesis testing.

I. INTRODUCTION

Large systems are composed of multiple modules. Modularization [1] is critical to scaling up software engineering projects: without proper modular decomposition, and congruent work assignment [2] large system development would become hopelessly mired in knowledge bottlenecks and coordination overheads. Indeed, as systems scale up to millions of lines of code and beyond, designers seek hierarchical modularization; for example, the code in large software products such as Eclipse exhibit at least 3 hierarchical levels of decomposition, *viz.*, files, packages and plugins/modules. Hierarchical decomposition in software products naturally dovetails (and is often beneficially congruent [3], [4]) with the hierarchical organization of modern software development teams. Indeed, quite often we find software processes are themselves also hierarchical, with steps contained within steps. Thus, we have hierarchical systems developed by hierarchical teams using hierarchical processes.

While hierarchical decomposition is largely an unalloyed blessing for large software products, teams and processes, we argue that it poses risks for empirical software engineering research (ESE). ESE is concerned with observable outcomes such as quality and productivity; such outcomes are subject

to large-sample studies, so that a) statistical methods can be brought to bear for hypothesis testing, and b) automated machine learning and mining methods on past data can be built into tools that support programming tasks. Thus for example, many studies focus on software quality, by choosing the number of defects in an element as a response variable, and (depending on the hypothesis) measures of such factors as the complexity of the element, the number of contributors, the development organization's social and geographic structure, as predictor variables. Likewise, historical defect data is mined and used with machine learning models to automatically predict likely future loci for defects. These ideas animate a large body of ESE research.

Hierarchical decomposition becomes important here. Many papers¹ study phenomena at the level of files [5], [6]; others study them at higher levels of aggregation [7], [8], [9], [10] (*e.g.*, packages or modules).

But what is the right level of study?

The reason why this is an important question, worthy of being underlined and italicized, becomes clear when we ask two derivative questions:

- If we build a statistical model to test a hypothesis at an aggregated level (*e.g.*, packages), do the findings in the model hold at the dis-aggregated level (*e.g.*, files) ?
- If we build automated prediction models at varying aggregated levels (*e.g.*, bug prediction at a module-level, or at a file-level) does the performance at these different levels give equally valid indications of the actual cost-effectiveness of their predictions (*e.g.* when using these prediction models for inspections)?

Ecological inference is the conceit that an empirical finding at an aggregated level (*e.g.*, packages) can apply at the disaggregated level (*e.g.* files). When this inference is mistaken, we have the *ecological fallacy*.

¹There are too many papers to enumerate, we merely present a few representatives

Contributions: In this paper, we make the following contributions.

- 1) We present a detailed conceptual overview of Ecological inference and ecological fallacy: What are they? Why are they relevant in software engineering? What are the specific risks in software engineering?
- 2) We present a theoretical discussion of several factors known to give rise to ecological inference risk: *sample size*, *zonation*, and *class imbalance*.
- 3) We empirically study the incidence of ecological inference in 18 open source projects. We find:
 - Ecological inference risk in defect prediction models: while it may appear from ROC type measures that aggregated, package-level predictions models are similar (or slightly better than) disaggregated, file-level prediction models, in fact, when using cost-effectiveness measures, file-level models are decidedly better.
 - Ecological inference risk in hypothesis testing: using multiple regression, we find quite a number of cases where a null hypothesis is rejected ($p \leq 0.05$) at an aggregated level *cannot* be rejected at the disaggregated level, and vice versa.

The goal of this paper is to lay out a conceptual framework of ecological inference risk in software engineering, and empirically demonstrate the existence of this risk. In future work, we hope to study the effects of the factors (sample size, zonation, and class imbalance) on this risk.

Relevance to Automated Software Engineering: This paper is concerned with the construction of defect prediction models, which provide an automated way to focus quality-control efforts.

II. BACKGROUND AND CONCEPTS

The risks of transferring statistical inferences from aggregated groups to smaller constituent groups were noticed as early as 1950 by Robinson [11]. He observed that at an aggregated level (U.S. States), immigrant status was positively correlated (+0.526) with educational achievement, but at the individual level, it was negatively correlated (-0.118). This discrepancy has been attributed to the tendency of immigrants to congregate in regions with higher levels of education. In this case, the congregational tendency of immigrants is a confounding phenomenon at the aggregated regional level that jeopardized the internal validity of the study at that level. This discrepancy between the findings at the aggregated and disaggregated levels illustrates the *ecological fallacy*.

Detecting a phenomenon at an aggregated level, and then inferring it to apply at a disaggregated level is called *ecological inference*; doing so risks the *ecological fallacy*. For brevity, we use the abbreviation \mathcal{EI} for “ecological inference” and \mathcal{EF} for “ecological fallacy”. While the above (now-classic) example illustrates the ecological fallacy, by no means is it the case that all ecological inferences are subject to ecological fallacies.

This issue has been explored in geographical and epidemiological research. The term MAUP or Modifiable Areal Unit Problem introduced by Openshaw and Taylor [12] addresses the issues of scale and zonation in geographic data. *Scale* refers to the size of the aggregated unit: larger scale means bigger and fewer aggregated units. *Zonation* refers to the manner in which aggregation is performed. A vividly pathological example of zonation is gerrymandering, wherein geographic regions are aggregated in distorted, artificial ways to the deliberate electoral advantage of a political party. MAUP refers to the problem of choosing the proper scale and zonation when studying phenomena that are subject to aggregation, and thus mitigating the risk of ecological fallacy.

In Empirical software engineering (ESE) phenomena are often studied at the aggregated package or module levels [9], [10]. Certainly, assuming that hypotheses supported at the package level hold at the file level is subject to \mathcal{EI} risk. This risk applies to prediction models as well. Consider the use of prediction models—they are intended to be used to focus quality control efforts such as inspection. Inspectors work line by line, whereas prediction models are trained on aggregate metrics at the level of files or packages. It’s possible, and indeed it has been documented, that prediction models that are designed to work well at the level of files *do not* work well at the disaggregated level of lines, which is arguably the right level to determine actual inspection effort! [13], [14]. Although the issue of \mathcal{EI} arises in empirical software engineering, to our knowledge, there has been no explicit discussion of this in the literature.

A. ESE, \mathcal{EI} and \mathcal{EF}

In the rest of this section, we qualitatively consider \mathcal{EI} in the context of ESE, illustrating the issues that can arise using several examples. The discussion below has been informed by similar considerations presented in other fields [15], [16], [17] and is in the spirit of the paper by Briand *et al.* [18] on the application of measurement theory in ESE. First, we discuss the reasons why studies may have to be conducted at higher levels of aggregation (thus risking the ecological fallacy). Next, we discuss construct validity issues that arise directly from aggregation. Next, we consider zonation issues, which can threaten internal validity, and were at the heart of Robinson’s original formulation of the ecological fallacy. Finally, we consider the basic, fundamental issue of sample size that rears its head at increasing levels of aggregation.

B. Reasons for Aggregation

One might reasonably ask, why not avoid ecological fallacies altogether by always conducting studies at the lowest level of aggregation (finest resolution)? There are natural reasons for aggregated studies.

Aggregate Phenomena Relevant phenomena apply only at a higher aggregated level, or differently at different levels. As a simple example, some object-oriented phenomena, such as inheritance and class cohesion [19], apply only at the class level, not at the lower level of methods. Fan-in and fan-out

(number of methods calling, or called-by a method) apply at the method level, not at the level of a line of code. As a more subtle example, certain types of phenomena may only emerge at the team level: some teams may be better managed or more cohesive than others, and as a result, team-related quality and productivity effects may apply equally to members of a team, and differentially across different teams. If teams (as is common) are assigned work based on modular decomposition, then modules will reflect these team effects. In such situations, an aggregated, module-level study may be appropriate; even so, transferring findings from (or prediction models trained at) the aggregated level to the disaggregated level is risky.

Observational Resolution Data may be observable at only certain levels of aggregation. Field defect data may be only available at the binary level, since failure information available to users and customer support staff might not have file-level information. Customer satisfaction ratings might only be available at the level of complete applications. Despite the unavailability of data at lower levels of aggregation, \mathcal{EI} may be desirable, as it may lead to actionable concepts. For example, if statistical findings relating to field defects at the binary level could be transferred to the file level, it may suggest better ways of doing work assignment at the file level, in order to reduce incidence of field defects. Of course, if such inferences are fallacious, the results may not be actionable.

Next we discuss 3 possible pitfalls of aggregated studies in empirical software engineering.

C. Difficulties of Interpreting Aggregated Results

If we model data at an aggregated level, and observe some statistically significant result, what is the meaning of that result at the disaggregated level? Given an aggregated level finding, what action should we take? A measure's interpretation, or content, may not easily transfer from an aggregated level to a disaggregated level; that is, a measure of a property at a coarser level of aggregation might be difficult to interpret and/or act upon at a more fine-grained level. Some epidemiologists (see, e.g., Schwartz [17]) have considered this to be a form of construct validity. In fact, a finding at a module level and a finding at a file level, with the same measure, might indicate different remedies. We now present two *gedanken* examples of this;

Organizational Effects. Organizational effects have been found to affect quality. For example, Nagappan et al [20] report that metrics such as ownership and organizational diameter are excellent predictors of defect occurrence. The study was conducted at the level of binaries in Windows, which are aggregations of files. Consider the simplest metric presented, the number of engineers (NOE) working on a binary. If NOE is strongly correlated with defects at the binary level, two interpretations are possible. Consider a binary with high NOE, and also high defects. First, if the binary has a large number of source files, each allocated to a different developer (thus having overall high NOE), then miscommunication between developers might be leading to errors. In this case, the proper response might be to more carefully define inter-file interfaces

("design rules" [21]). On the other hand, if the binary has a few number of files, but each is maintained by multiple developers (thus also having high NOE), the difficulty might be arising from developers tripping over each other's work; in which case the response might be to allocate fewer developers to each file, or to actually split the files up into smaller work units.

Geographical Issues. Several studies [22], [23], [24] have found that geographical distribution can affect software quality. The same arguments presented above apply here. For example, Ramasubbu & Balan [23] find that geographical distribution at the product level negatively impacts quality and productivity. Suppose that products are composed of binaries, and binaries are composed of files. What is the correct response to this finding? Should we ensure that all developers of each product are in one location? Or that developers of a single binary are not geographically split? Or that developers of single files are not split? Any of these types of splits might have given rise to the quality and productivity impact observed in the study. Interestingly, the other two studies [22], [24] were at binary and file levels, and have yielded different results. Spinellis [24] considers the geographical distribution of committers to files in FreeBSD; Bird et al [22] consider distribution developers at the binary level. These two studies have found no effect of distribution on software quality. The 3 studies were conducted in different settings (outsourced development, open-source, and globalized development, respectively in the order discussed above) and doubtless, the results are influenced by the setting. Still, it is reasonable to speculate whether the discrepancy is a result of the choice of aggregation.

D. \mathcal{EI} Risk factor: Zonation

Another issue to consider is the possibility of confounding factors in *zonation*, the way in which smaller units are aggregated into larger units. These can threaten internal validity. Internal validity questions can arise when an observed relationship between independent and dependent variables might be an artifact of confounding variables. Aggregation *per se* can lead to confounding, and thus threaten internal validity. In Robinson's example quoted above, the tendency of immigrants to move to areas where established residents are more educated is confounding.

E. \mathcal{EI} Risk Factor: Sample Size

Another issue to consider in aggregation is sample size. It may be possible to conduct a study at the level of methods, classes, files, packages, modules, or even products. Suppose that in the study, the phenomena of interest are modeled by a set of independent variables and a dependent variable, and furthermore, that this set of variables can be reasonably aggregated at multiple levels for study. As the level of aggregation increases, there will necessarily be fewer and fewer samples. The lowest levels of aggregation, with the large sample sizes, will yield the highest statistical power, the higher levels will have lower power, and greater risk of over-fit models. On the other hand, it is possible that some of the variables are only available at higher levels of aggregation, for reasons discussed above.

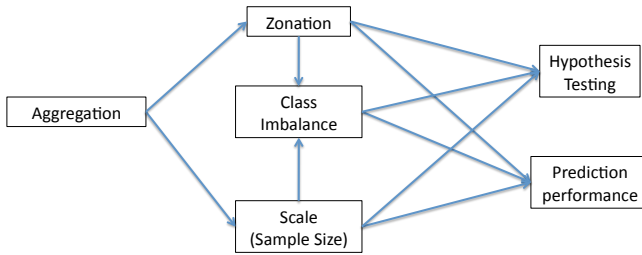


Fig. 1: Conceptual Framework

F. \mathcal{EI} Risk Factor: Class Imbalance

Defect prediction data often suffers from the class imbalance problem. Entities are labeled as defective if they contain at least one defect and not defective if they contain no defects; this *class labeling* is seldom balanced in that the majority of the entities contain no defects, and yet, it is the few that do that we wish to identify. The resulting class imbalance is mediated both by zonation and sample size; since the number of defects remains almost constant across levels². Even if only defective entities aggregate only with other defective entities, the number of aggregated entities is often sufficiently smaller than the disaggregated entities that class imbalance is reduced. Consequently, inferences drawn from the aggregated data may not hold at the disaggregated level if class imbalance significantly affects modeling efforts.

G. Summary

Figure 1 summarizes the discussion above, and presents our preliminary conceptual framework of the way to approach \mathcal{EI} in the context of empirical software engineering work. We consider two end-goals: prediction models, which aim to focus human effort, and hypothesis testing, which aims to find statistical evidence in data in support or reject claims.

The figure illustrates how scaling and zonation arise out of aggregation. At the simplest level, scaling affects sample size (§ II-E), and can thus affect the power of statistical models built with aggregated data; small sample sizes can also make the performance of prediction models built with aggregated data unpredictable. Aggregation naturally also affects zonation. Both sample size and zonation can affect class imbalance. For example, considering the aforementioned “defective” and “non-defective” class labels; class imbalance at the file level might very well be reduced by aggregation into packages. Furthermore, the manner in which zonation groups non-defective files along with defective files can also affect imbalance.

All 3 factors, sample size, zonation, and class imbalance, affect the quality of the statistical models that are built to do defect prediction, or hypothesis testing. Sample sizes have well-known effects on the validity and stability of models. Zonation can create internal-validity threats as a result of latent choices made in the way samples are grouped into aggregates,

²In some cases a defect may be associated with more than one file within the same module causing a reduction in the number of defects in the process of aggregation.

as shown by Robinson. Finally, class imbalance as discussed above also influences model quality.

III. RELATED WORK

ESE researchers have investigated a wide diversity of phenomena, including various aspects of quality and productivity. Ecological inference, and the attendant risks, are likely to be relevant in this broad context. Our study is focused on research into software quality, specifically into techniques that build statistical models of defect occurrence, either for hypothesis testing or defect prediction; our survey of related work is thus largely confined to statistical and predictive models of defect occurrence.

A. Modeling and Predicting Defects

The core idea in this very popular area of research is to consider the number of defects as a response variable, and choose predictors based on intuitions about factors that contribute to defects, such as complexity, design attributes, personnel attributes, programming practices, team structures, and so on. Regression models and correlation studies have been used to gauge the explanatory power of different variables, and thus test hypotheses concerning the etiology of defects. Another line of research, exemplified by the annual PROMISE³ meetings at ICSE, attempts to build accurate, reliable prediction models to predict where defects might occur next, using past data for training. This research is opportunistic and eclectic, leveraging a wide set of statistical and machine-learning techniques to improve prediction performance.

This research area is too rich and varied to survey and cite in detail. Our interest here is in aggregated modeling of defects, for prediction or hypothesis testing, so we present some examples of this approach.

Aggregated Defect Models Aggregation is common in defect modeling research; a common motivation is *observational resolution* (See § II-B), since field defect data is often reported at aggregated levels, *e.g.*, at the level of binaries [22], [8], or even complete projects [23]. Koru *et al.* [7] recommend aggregating for a different reason: improving model performance. They sum method-level measures into class-level measures, then use these aggregated measures along with metrics of class-level properties to improve model performance. They report that this approach overcomes problems arising from the skewed distribution of defects in the NASA KC2 dataset, and defend their approach, arguing that the source dataset is large enough to obtain statistically meaningful results.

Ramler *et al.* [25] also address how to build a quality defect model and reiterate the findings of Koru *et al.* [7]. In a study aimed at predicting the likelihood (count) of post release failures at the module level, Nagappan *et al.* aggregate per/class and per/function metrics by sum and maximum and combine with module level metrics [26]. Schröter *et al.* use imports gathered at the file level to predict failures for both files and packages. Imports are aggregated to the package level

³Predictor Models in Software Engineering

from the subordinate classes [9]. The authors assert “Intuitively, predicting for a coarse granularity is easier while using fine-grained input features yields better results.” Zimmerman *et al.* [10] report file and package level correlations between defects and file level and aggregate code metrics. In their study the package level correlations are consistently higher than the file level correlations. Similarly models built from package level variables showed a constantly higher R^2 value. In a later study, Zimmerman *et al.* use function, class, and file level metrics aggregated to the binary level to predict failure prone binaries [27].

However, aggregation has not been without controversy. Ambros *et al.* [28] criticize the use of aggregate level metrics for defect model evaluation. They claim that “Predictions at the package-level are less helpful since packages are significantly larger” and that “the review of a defect-prone package requires more work than a class.” In addition they assert that “classes are the building blocks of object-oriented systems, and are self-contained elements from the point of view of design and implementation” and that “package-level information can be derived from class level information, while the opposite is not true.”

Class Imbalance As previously discussed, defect data frequently suffers from class imbalance. This problem has been studied both generally, and within the context of software defect data. One simple technique to correct for class imbalance is to bias the sampling process. Undersampling and oversampling are opposite procedures that bias the sampling procedure to either favor the minority class (oversampling), or penalize the majority class (undersampling). Drummond and Holte showed that when using their cost-sensitive evaluation technique, *viz.* the cost of misclassification is taken into account, that undersampling outperforms oversampling [29]. Other more elaborate techniques have also been proposed. Gu *et al.* propose a technique that minimizes the impact of problem instances, *viz.*, those instances whose predictors cause class overlap [30]. Menzies *et al.* apply micro-sampling, which combines undersampling and data reduction, to defect data [31].

In conclusion: previous efforts at statistical modeling of defect occurrence have given some consideration to aggregation and class imbalance issues. However, our specific concern here is with the validity of \mathcal{EI} : whether models built at the aggregated level apply at disaggregated levels. With defect models, the question concerns both the hypothesis testing and prediction performance. The former is addressed in § V; before we address the latter, we first discuss how prediction performance is evaluated.

B. Evaluating Prediction Models

There are several approaches to defect prediction model evaluation: we begin with the simplest, *precision/recall*, and work up to the more sophisticated, *cost-effectiveness curves*.

Precision/Recall Defect prediction can be viewed as a binary classification problem. Entities are classified as defect prone or not based on predictors such as complexity, size, ownership,

and pre-release defects. An entity predicted as defect-prone is considered a true positive (TP) if it actually contains a post-release defect, and false positive (FP) if it does not; entities predicted as not defect-prone that contain defects are false negatives (FN) and the rest are true negatives (TN).

Well-known prediction performance measures ensue from these counts. *Accuracy*, computed as $\frac{TP+TN}{TP+FN+FP+TN}$ yields the chance that the total number of modules will be predicted correctly. *Precision* is the ratio of correct predictions to the total predicted as defective $\frac{TP}{TP+FP}$. *Recall* is the ratio of correct predictions to the actual number of defective entities $\frac{TP}{TP+FN}$. A good model should achieve both high precision, and high recall, but there is a well-known trade-off between the two. The F-Measure takes the harmonic mean of the two, and has been used to measure overall prediction performance. Most classifiers do not produce a hard yes/no decision, however. The classifier output is typically a probability that must then be compared to a threshold to obtain a classification decision. Significant work has addressed how meaningful model evaluation criteria are with respect to defect prediction models. Ma *et al.* [32] criticize accuracy as it ignores the data distribution and cost information. Lessmann *et al.* [33] argue that the requirement of defining a threshold is reason enough not to use such simple static measures in a defect prediction context.

ROC An established method of evaluating classifiers independently of any particular threshold is *Receiver Operating Characteristic* (ROC) analysis. An ROC curve represents a family of precision/recall pairs generated from varying the threshold value between 0 and 1 and plotting the False Positive Rate $FPR = \frac{FP}{FP+TN}$ on the x -axis and the True Positive Rate $TPR = \frac{TP}{TP+FN}$ on the y -axis. All such curves pass through the points (0, 0) and (1, 1). The point (0, 1) represents perfect classification and points on the ROC curve close to (0, 1) represent high quality classifiers. A common way to evaluate the overall quality of the classifier is to compute the area beneath its ROC curve, which we denote AUCROC; this has a value between 0 and 1.

A limitation of ROC curves, and by extension all of the aforementioned measures, is that they value all entities the same. Classification of an entity as defect prone is not the only, or even the primary, goal of defect prediction. Ideally, the goal of such efforts would be to aid in efficiently guiding corrective maintenance. Thus the classifier tells us where to find defects, *viz.*, approximately where to look to take corrective action. We can view inspection effort as roughly proportional to lines of code, and so it makes sense that the value of inspecting a class depends on its bug density. Ma *et al.* address this issue and recommend careful evaluation of ROC curves at meaningful performance points [32], Arishom *et al.* [34] address it somewhat differently by defining a metric, cost effectiveness, more appropriate to defect prediction models motivated by code inspection.

Cost Effectiveness Suppose defects were uniformly distributed through the source code. If an inspection budget allows inspection of 10% of the source code then we might

Metric	description
LOC	Source lines of code
Lines	Total lines in file/package
# Developers	Number of developers who have edited this file/package
# Active Developers	Number of developers on this file/pkg in current release
Churn	Number of added changed lines
Commits	Count of commits to file/pkg
Features	Number of new features as identified by issue tracker
Improvements	Number of improvements as identified by issue tracker

TABLE II: Metrics gathered and their description.

choose 10% of the lines at random and might reasonably expect to find about 10% of the defects. This scheme requires minimal work and no expertise in data gathering and defect modeling. Therefore, it is reasonable to expect that any useful defect prediction method should be able to improve on this result. This is the basis for the cost effectiveness (CE) metric defined by Arisholm *et al.* [34]. The CE curve plots percentage of identified faults found against the number of lines of code accumulated by entities considered. To use a prediction model, we use the model to compute a predicted defect probability for each entity. Entities are ordered by decreasing order of fault probability and increasing size. A successful model is then one that predicts a greater percentage of faults found than the percentage of lines of code inspected, hence, a curve that lies, at least in part, above the line $y = x$. The Arisholm *et al.* formulation computes only the area above the line $y = x$ as contributing to the CE measure. We simply take the area under the CE curve (denoted AUCCE) as our measure as we are not evaluating practical models, rather, we are looking at the range of cost effectiveness achieved by models in different settings. Arisholm *et al.* expand on their work on CE in a systematic investigation of methods used in building defect prediction models considering issues of ROC vs cost effectiveness [13]. We use the AUCCE measure to evaluate how prediction models perform primarily with respect to code inspections. We argue that a independent of prediction performance, the model that facilitates the identification of the greatest number of defects after inspecting the fewest lines of code is of greater practical use.

IV. EXPERIMENTAL METHODS

We now present our findings illustrating the risks of ecological inference, in the two settings commonly used in empirical software engineering, defect prediction and hypothesis testing. These findings support the following claims:

- 1) *Prediction models are subject to ecological inference risk.* Models built using aggregated data, even when they show reliable performance at the disaggregated level, can have less reliable performance at the aggregated level.
- 2) *Hypothesis testing is subject to ecological inference risk.* Strong, practically significant relationships observed at

the aggregated level may weaken, at the disaggregated level

But first, some details of our experimental approach.

Data Gathering We extracted data from the JIRA defect tracking system and associated git repositories for 87 distinct versions of 18 different ASF (*Apache Software Foundation*) projects described in Table I. For each release we extract the JIRA RSS feed, an XML report of JIRA issues. We then crawl the associated JIRA webpage for each issue found in the XML report and extract the commits related to that issue. We then link this data to the git log to determine which commits, and consequently which files and packages, are associated with defects. A file associated with a closed defect in a commit is considered to be a partial repair for that defect in that commit and the file is labeled as defective within that release. For each file in a release we gather the size in LOC (*lines of code*) using the *SLOCcount* tool and aggregate these counts to the package level [35]. The number of developers associated with each file and package is identified from the unique author names in the git log and counted directly for both files and packages. We distinguish between the number of developers who have ever touched a file or package and active developers, *viz.*, the count of those actively working on the package in the current release. Similarly, we count unique defect IDs in each file and package to identify the number of defects associated with each release. We count *improvements*, the number of code improvements identified by the JIRA tracking system as well as *features*, the number of new features.

We would certainly have liked to run our experiments with existing data used by other researchers, *e.g.*, such as can be found in the promise repository [36]. We note, however, the vast majority of such data is measured at a single aggregation level; it is, therefore, impossible to *post facto* accurately perform set unions to aggregate defect and developer counts to a higher level. A single defect, as an example, associated with each of three classes in a package contributes only a single defect to the package. Since most available data contains only counts and not issue identifiers, it cannot be properly aggregated.

Modeling Defects

We use logistic regression to classify files and packages as defective using the metrics referenced in Table II as predictors and the existence (0 or 1) of a post release bug in the entity as a response. For each predictor the *z-test statistic* is computed by dividing the estimated value of the parameter by its standard error and used to assess the significance of the variable. This statistic is a measure of the likelihood that the actual value of the parameter is not zero: the larger the absolute value of the statistic, the less likely that the actual value of the parameter could be zero. For each package/file variable pair, changes in the *p-value* associated with the *z-test* and an *alpha* value of 0.05 to decide if a parameter can be judged to have a significant effect.

Given the large number of projects and releases we used an automated model selection technique to identify models.

Project	Releases	Description	# Releases	# Files	# Packages
Abdera	1.0.1, 1.1	Atom (XML Syndication) implementation	2	672-680	112-113
Cassandra	0.6.0 - 0.6.8	Distributed Database	9	314-332	31-33
Cayenne	3.0, 3.0.1	Java Object Relational Mapping framework	2	2763-2764	160-162
CXF	2.11-2.3.1	Services Framework	17	3086-4097	491-598
HttpCore	4.0.1,4.1	Http Core Library	1	451-451	29-29
Ivy	2.0.0 - 2.2.0	Agile Dependency Manager	3	481-498	65-67
IvyDE	2.0.0, 2.1.0	Eclipse plugin for Ivy	2	118-95	20-23
James	2.3.0 - 3.0	Java Apache Mail Enterprise Server	5	375-477	39-85
Lucene	1.9.1 - 3.0.3	Text search engine library	7	1010-957	102-85
Mahout	0.4	Machine Learning framework	1	1119-1119	147-147
Nutch	1.1, 1.2	Web search software	2	446-453	89-91
ODE	1.2-1.3.4	Business process executor	7	1034-954	122-99
OpenEJB	3.0 - 3.1.3	Enterprise Java Beans	9	2191-2949	124-191
Pluto	2.0.0, 2.0.1	Java Portlet reference implementation	2	370-371	44-44
Shindig	2.0.0, 2.0.1	OpenSocial application	2	811-812	75-75
Solr	1.3.0 -1.4.0	Lucene search server container	2	542-749	33-36
Wicket	1.2.7 - 1.3.7	Web Application Framework	9	1776-1947	240-249
XercesJ	2.7.1 - 2.11.0	Java XML parser	5	740-827	67-71

TABLE I: Apache projects and their description

We enumerated all combinations of at most six predictors from those presented in Table II. It has been shown that size often follows a log-normal distribution [37]. Consequently, we include the log transformations of size variables in addition to their untransformed counterparts allowing the model selection process to choose between them. Log transform data can increase central tendency and reduce heteroskedasticity, however, in some cases untransformed data may yield a better fitting model [38]. We rank the models by *Akaike Information Criterion* (AIC) score and select the model with the lowest AIC value. Using AIC alone, however, may yield models with high multicollinearity amongst some of the variables. Consequently, we check for this by rejecting models with VIF *Variance Inflation Factor* higher than 5. Without specific reasons for including a high VIF variable in a model, the value 5, is generally considered to be the maximum acceptable value [38]. By rejecting models with high VIF and sorting by how well the models fit the available data, this process yields models that are similar to what a researcher might choose if she were manually identifying models. In addition, it does not constrain our process to choose the same model for each revision and project.

V. FINDINGS

If we build a prediction model that performs well on an aggregated measure, how does it perform on a disaggregated measure? To address this question we build models at two levels and explore the relationship between them. For the aggregated level, we use files and packages; a prediction model works well at this level if it accurately predicts which files (or packages) have defects, with as few false positives and false negatives as possible. As a measure of performance at these aggregated levels, we use a ROC chart, as measured by AUCROC. For the disaggregated level, we use a cost-effectiveness lift-chart, as measured by AUCCE.

We built prediction models at two levels: at the base level of files, and then an aggregated level, where files are aggregated into packages. While most files contain a single Java class, some files may contain more than one; however, each file belongs to at most one package. Of the 87 available versions of the 18 projects discussed in Section IV, we use only the 68 versions that meet minimal sample-size and events per variable requirements considered appropriate for use with maximum likelihood estimation approach used in logistic regression. [39]. The minimum is required for both number of packages and number of files and is computed as follows. Let p be the minimum class percentage, *e.g.* for 100 files with 10 defective, $p = 0.1$. Let k be the number of predictor variables. Then the minimum number of samples is $N = 10 \frac{k}{p}$. We thus reject datasets where there are not only insufficient numbers of datapoints, but also insufficient numbers of datapoints in each class.

A. Aggregation Effects on Model Quality

In the first study, we sought to evaluate how the level of modeling affects the quality of the models. We used 55 of the available revisions from 15 projects based on the criteria that each dataset used needed a revision on which to train, and at least one subsequent revision on which to evaluate the predictive power of the trained model. Each of the 55 prediction models was evaluated using two measures. First at both levels, we calculated the area under the ROC curve (AUCROC) for each of the models. Thus, *e.g.* at the *package* level, AUCROC measures how well the package-level model predicts defective *packages*, and so also respectively the file-level model. Second, for both levels, we calculate the AUCCE measure for each of the models; *viz.* at the package level, we plot the cost-effectiveness curve, and calculate the AUCCE; this is repeated for the file level.

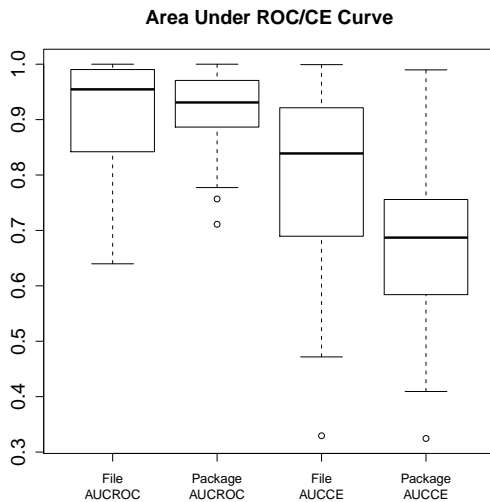


Fig. 2: Comparing AUCROC and AUCCE for packages and classes. AUCROC is not significantly different across aggregation levels by a two-sided Wilcoxon rank sum test with $p\text{-value}=0.51$. AUCCE is significantly lower for package level results by Wilcoxon rank sum test with $p\text{-value}=2.462e-05$.

Even though package and file level predictor models are often indistinguishable by their AUCROC measures, file level models show clearly better AUCCE measures. This can be observed in Figure 2, showing that the AUCROC’s are sometimes higher at the package level. Thus, if we build a model at a *package* level, and *evaluate only at the same level*, package level models may appear to perform better.

... *but file level models are actually better*: This phenomenon can be observed from Figure 2, showing that the AUCCE’s are generally higher at the file level. Thus, if we build a model at the level of files or packages, but evaluate at the fine-grained level (note that AUCCE considers lines of code level), rather than at the aggregated level, we find that file-level models perform better.

If an unwary investigator built a prediction model at the package level, and also at the file level, and compared them using same-level AUCROC, he might falsely conclude that the package level model is giving comparable (or, in some cases, better) performance, whereas in fact, it is performing *worse* in terms of the more demanding and realistic AUCCE measure. From these observations, we draw the following rather sobering conclusion.

Aggregated prediction models, when evaluated purely at the aggregated level, can look better than they really are.

B. Ecological Inference Risk

Next, we study the risks of Ecological Inference to ESE. Conceptually, the risk arises in this setting: one builds a statistical model at an aggregated level, uses it to test hypothesis, and *simply assumes that the results also hold at*

the disaggregated setting, without testing the same model at the disaggregated setting. We show that modeling defect occurrence at two different aggregation levels can lead to substantially different fits of model parameters to the data, and consequently to very different statistical inferences.

We used model selection based on AIC and VIF to select the best multiple regression model at the aggregated (package) level, and used the same set of variables from this best aggregated model to build a model at the disaggregated (file) level. This gives a matched pair of aggregated-disaggregated models for all project revisions (18 projects, 68 versions total).

In order to evaluate the hypothesis that the corresponding model variable affects defects, it is standard practice to use the $p\text{-value}$ of a coefficient within each model. If the $p\text{-value}$ is below a threshold, say 0.05, that model variable has a significant effect on defects; if not, the null hypothesis (that it does not affect defects) cannot be rejected. We follow that practice here in selecting the significant variables from our models. Furthermore, for a particular variable, we check if the models at both levels give the same results (i.e. if it is significant at both levels or insignificant at both levels). Thus, for example, we find that the *number of active developers* tends to be significant at both levels of aggregation in many of the 68 models.

Major changes in a variable’s significance can result in substantially different conclusions from the inferences. To evaluate $\mathcal{E}\mathcal{I}$ risk, we studied the change in the significance of a variable in a model between the aggregated-disaggregated pair. Specifically, for a given level of aggregation (file or package), and a given project version (18 projects, 68 versions total), we noted two properties of each model variable: its fitted parameter’s sign (positive or negative) and its significance to the model (significant or insignificant).

We compared the corresponding parameters between the two levels, and focused on three outcomes indicative of $\mathcal{E}\mathcal{I}$ risk. The first is a change in sign (from + to - or vice versa) between the two levels of modeling; we did not observe any such changes in our data. The second is *gain of significance, or GS*, where the variable gains significance when aggregated from file to package. The last is *loss of significance, or LS*, where the variable is significant at the package level, but not at the file level ⁴.

We discuss some observed examples of LS and GS below. For example, the *number of commits* is an “LS” variable in the Abdera project: significant at the aggregated package level for the Abdera project, but insignificant at the file level. Thus, if one were to conclude from the package-level model that *code churn* affected defects, and then unwarily used ecological inference to conclude that files that were subjected to more commits are more defective, that would be fallacious in this case; committing additional resources to inspect files would probably be unwise. Likewise, in the case of Ivy, the *number of active developers* is a “GS” variable, insignificant at the

⁴There can be two other possible observations, when the coefficients are either *both significant, or SS* or *both insignificant, II*, but they carry no inference risk.

Type	Predictor	# Releases	# Significant Releases	# Projects	# Significant Projects	Projects
LS	commits	5	26	2	12	abdera, cxf
LS	activedevs	2	32	2	11	abdera, wicket
LS	improvements	5	15	2	6	cxf, openejb
LS	devs	2	16	2	7	cxf, openejb
LS	lines	3	1	2	1	cxf, wicket
LS	features	5	9	4	6	cxf, james, nutch, ode
LS	added	1	8	1	4	cxf
LS	loc	2	1	2	1	cxf, openejb
GS	commits	6	26	5	12	cassandra, ivy, nutch, openejb, wicket
GS	activedevs	4	32	4	11	cassandra, cxf, ivy, wicket
GS	improvements	4	15	3	6	cxf, openejb, wicket
GS	devs	2	16	2	7	ivy, openejb
GS	lines	5	1	4	1	cxf, wicket, abdera, mahout
GS	features	2	9	1	6	cxf
GS	added	1	8	1	4	wicket
GS	loc	4	1	4	1	lucene, cxf, ode, xercesj

TABLE III: Counts of gains and loss of significance in inference models. As an example (first row), the predictor *commits* showed loss of significance (LS) in 5 different releases in 2 different projects, *abdera* and *cxf*; the variable was actually significant ($p \leq 0.05$) in 26 releases in 12 projects in at least one level of aggregation.

package level, but gaining significance at the file level. Thus, an unwary researcher might conclude from a package-level model that this variable has no influence at the file-level, whereas in fact it does.

In summary, we found that out of a total of 108 variables used in the 68 models, we found 28 instances of GS, and 25 instances of LS. Table III summarizes our findings for the variables that showed gain and loss of significance. Overall, the table shows a worrying number of cases of GS and LS. Certain variables, such as *number of commits* and *number of improvements*, show either GS or LS in quite a number of cases, and illustrate the potential for ecologically fallacious inferences when hypotheses are tested at the aggregated level, and the results of these tests are inferred to apply at the disaggregated level. Especially notable is the *number of new features* variable. It shows both gain and loss of significance. For it, we observe GS in 2 revisions, on 1 different projects, and LS in 5 revisions across 4 different projects. These numbers should be considered relative to the total number of revisions (and projects) where the variables were found to be significant at least at one level. Thus, *the number of new features* was significant in 9 revisions in 6 different projects. This gives an indication of the chances of running into $\mathcal{E}\mathcal{I}$ risk.

Inferences drawn from models built at aggregated levels such as models, packages, or binaries, may not transfer to disaggregated components (e.g. files) used to build the aggregations.

VI. THREATS TO VALIDITY

Perry *et al.* [40] identify three forms of validity that must be addressed in research studies. We now examine threats to each form of validity in our study and the methods used to mitigate these threats where possible.

Construct validity attempts to reconcile measured properties with the concepts they are believed to represent. Files often

contain more than one class, hence, they are themselves aggregations of classes. It could be argued that a better comparison is classes vs. package. Files, however, are the lowest unit of aggregation for which we had bug linking data so we feel that the aggregation is justified in this case.

Internal validity is the ability of a study to establish a causal link between independent and dependent variables regardless of what the variables are believed to represent. We rely on linking data to identify defects, hence, our models cannot capture defects not linked by developers. This approach, however, is widely employed in this area. Further, this same limitation applies to *improvement* and *new feature* annotations of JIRA issue tickets. We were careful to address multicollinearity issues as part of our automated model selection process.

External validity refers to how these results generalize. The threats here are common to studies of this type. We use only open source data and the consider only source written in the Java language. Our case study is limited to a single source of projects, the Apache Software Foundation, which may affect to what extent our results can generalize. However, we study a large number of revisions, collected from 18 different projects.

VII. DISCUSSION & CONCLUSION

We have discussed and illustrated the risks of ecological inference in software engineering. However, it is important to keep in mind that it is difficult, indeed unwise to always attempt to avoid this risk. Software is inherently hierarchical. Certainly, products are hierarchical, (as discussed above with files, packages, and plugins). In addition, teams are hierarchical, with hierarchical management structures. Development processes can also be viewed hierarchically, with minor steps within major steps, and iteration loops nested within other loops.

Because of *aggregate phenomena*, and *observational resolution*, as discussed above in Section II-B, it is often necessary to study phenomena and/or gather data at aggregated levels of products, teams, or processes. It is also possible that the resulting findings are only actionable at the disaggregated

level, *viz.*, at the level of files, individual people, or steps of a process. Therefore, it is unlikely that ecological inference, risks notwithstanding, can be completely avoided in empirical software engineering. When making the inference, however, the risk of ecological fallacy needs to be considered and discussed. As we have argued above, there are *construct validity* issues; it is not always clear how to translate a finding relating to an aggregated metric into a concrete action that can be applied to a disaggregated product, process, or team. There are also *internal validity* issues, as we discussed above; factors that influence aggregation, such as intentional or unintentional assortativity (in our case studies) can confound the results, and threaten internal validity when ecological inferences are made. Of course, one must always be mindful while aggregating of the loss of statistical power due to reduction in sample size.

In conclusion, we hope to have convinced the reader that a) ecological inference is often unavoidable in software engineering research, and b) that managing the resulting risks ecological fallacy, is a ripe area for study in our field. We hope others will join us to explore these issues in other settings and other datasets, investigate specifically the effects of *sample size*, *zonation* and *class imbalance* on \mathcal{EI} risk, and also consider \mathcal{EI} risk in their future work.

REFERENCES

- [1] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Commun. ACM*, vol. 15, no. 12, 72.
- [2] M. Cataldo, J. D. Herbsleb, and K. M. Carley, "Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity," in *ESEM '08: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*. New York, NY, USA: ACM, 2008, pp. 2–11.
- [3] M. Conway, "How do committees invent," *Datamation*, vol. 14, no. 4, pp. 28–31, 1968.
- [4] J. D. Herbsleb and R. E. Grinter, "Splitting the Organization and Integrating the Code: Conway's Law Revisited," in *ICSE*, 1999.
- [5] E. J. Weyuker, T. J. Ostrand, and R. M. Bell, "Do too many cooks spoil the broth? using the number of developers to enhance defect prediction models," *Empirical Softw. Engg.*, vol. 13, no. 5, pp. 539–559, 2008.
- [6] A. Meneely, L. Williams, W. Snipes, and J. Osborne, "Predicting failures with developer networks and social network analysis," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. ACM, 2008.
- [7] A. Koru and H. Liu, "Building effective defect-prediction models in practice," *IEEE Software*, vol. 22, no. 6, pp. 23–29, 2005.
- [8] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Proceedings of the 27th international conference on Software engineering*. ACM, 2005, p. 292.
- [9] A. Schröter, T. Zimmermann, and A. Zeller, "Predicting component failures at design time," in *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*. ACM, 2006, p. 27.
- [10] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *PROMISE*, 2007.
- [11] W. Robinson, "Ecological correlations and the behavior of individuals," *International journal of epidemiology*, vol. 15, no. 3, pp. 351–357, 1950.
- [12] P. Openshaw, S. and Taylor, "A million or so correlation coefficients: three experiments on the modifiable areal unit problem," *Statistical Methods in the Spatial Sciences*, pp. 127–144, 1979.
- [13] E. Arisholm, L. Briand, and E. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *Journal of Systems and Software*, vol. 83, no. 1, pp. 2–17, 2010.
- [14] 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 Engineering*, pp. 1–33.
- [15] D. Freedman, "Ecological inference and the ecological fallacy," *International encyclopedia of the social and behavioral sciences*, pp. 4027–4030, 2004.
- [16] S. Piantadosi, D. Byar, and S. Green, "The ecological fallacy," *American Journal of Epidemiology*, vol. 127, no. 5, p. 893, 1988.
- [17] S. Schwartz, "The fallacy of the ecological fallacy: the potential misuse of a concept and the consequences," *American journal of public health*, vol. 84, no. 5, p. 819, 1994.
- [18] L. Briand, K. Emam, and S. Morasca, "On the application of measurement theory in software engineering," *Empirical Software Engineering*, vol. 1, no. 1, pp. 61–88, 1996.
- [19] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Trans. Softw. Eng.*, vol. 22, no. 10, 1996.
- [20] N. Nagappan, B. Murphy, and V. Basili, "The influence of organizational structure on software quality: an empirical case study," in *ICSE 2008*, 2008.
- [21] C. Baldwin and K. Clark, *Design Rules: Vol 1*. MIT Press, 2000.
- [22] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy, "Does distributed development affect software quality?: an empirical case study of Windows Vista," *CACM*, vol. 52, no. 8, pp. 85–93, 2009.
- [23] N. Ramasubbu and R. Balan, "Globally distributed software development project performance: an empirical analysis," in *ESEC/FSE*. ACM, 2007.
- [24] D. Spinellis, "Global software development in the FreeBSD project," in *Proceedings of the 2006 international workshop on Global software development for the practitioner*. ACM, 2006, p. 79.
- [25] R. Ramler, K. Wolfmaier, E. Stauder, F. Kossak, and T. Natschläger, "Key questions in building defect prediction models in practice," *Product-Focused Software Process Improvement*, pp. 14–27, 2009.
- [26] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *ICSE*, 2006.
- [27] T. Zimmermann, N. Nagappan, and A. Zeller, "Predicting Bugs from History," *Software Evolution*, pp. 69–88, 2008.
- [28] M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches," in *MSR*, 2010.
- [29] C. Drummond and R. Holte, "C4. 5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling," in *Workshop on Learning from Imbalanced Datasets II*. Citeseer, 2003.
- [30] J. Gu, Y. Zhou, and X. Zuo, "Making class bias useful: A strategy of learning from imbalanced data," *Intelligent Data Engineering and Automated Learning-IDEAL 2007*, pp. 287–295, 2007.
- [31] T. Menzies, Z. Milton, A. Bener, B. Cukic, G. Gay, Y. Jiang, and B. Turhan, "Overcoming Ceiling Effects in Defect Prediction," *Submitted to IEEE TSE*, 2008.
- [32] Y. Ma and B. Cukic, "Adequate and precise evaluation of quality models in software engineering studies," in *International Workshop on Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007*, 2007, pp. 1–1.
- [33] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, p. 485, 2008.
- [34] E. Arisholm, L. Briand, and M. Fuglerud, "Data mining techniques for building fault-proneness models in telecom java software," in *Proceedings of the The 18th IEEE International Symposium on Software Reliability*. IEEE Computer Society, 2007, pp. 215–224.
- [35] D. Wheeler, "SLOCCount," Available from <http://www.dwheeler.com/sloccount>.
- [36] G. Boetticher, T. Menzies, and T. Ostrand, "PROMISE Repository of empirical software engineering data," *West Virginia University, Department of Computer Science*, 2007.
- [37] H. Zhang and H. B. K. Tan, "An empirical study of class sizes for large java systems," in *Software Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific*, dec. 2007, pp. 230–237.
- [38] J. Cohen, *Applied multiple regression/correlation analysis for the behavioral sciences*. Lawrence Erlbaum, 2003.
- [39] P. Peduzzi, J. Concato, E. Kemper, T. Holford, and A. Feinstein, "A simulation study of the number of events per variable in logistic regression analysis* 1," *Journal of clinical epidemiology*, vol. 49, no. 12, pp. 1373–1379, 1996.
- [40] D. Perry, A. Porter, and L. Votta, "Empirical studies of software engineering: a roadmap," in *Proceedings of the conference on The future of Software engineering*. ACM New York, NY, USA, 2000, pp. 345–355.