National Research Council Canada
Conseil national de recherches Canada
Institute for Information Technology
Institut de Technologie de l'information

ERB-1076

NRC-CNRC

A Methodology for Validating Software Product Metrics

Khaled El Emam
June 2000

Canada

NRC 44142

National Research Council Canada
Institute for Information Technology

Conseil national de recherches Canada
Institut de Technologie de l'information

A Methodology for Validating Software Product Metrics

Khaled El Emam
June 2000

A Methodology for Validating Software Product Metrics

Khaled El Emam
National Research Council, Canada
Institute for Information Technology
Building M-50, Montreal Road
Ottawa, Ontario
Canada K1A OR6
khaled.el-emam@iit.nrc.ca

1 Introduction

A large number of software product metrics[1] have been proposed in software engineering . Product metrics quantitatively characterize some aspect of the structure of a software product, such as a requirements specification, a design, or source code. They are also commonly collectively known as *complexity metrics*.

While many of these metrics are based on good ideas about what is important to measure in software to capture its complexity, it is still necessary to systematically validate them. Recent software engineering literature has reflected a concern for the quality of methods to validate software product metrics (e.g., see [38][80][106]). This concern is driven, at least partially, by a recognition that: (i) common practices for the validation of software engineering metrics are not acceptable on scientific grounds, and (ii) valid measures are essential for effective software project management and sound empirical research. For example, in a recent paper [80], the authors write: *"Unless the software measurement community can agree on a valid, consistent, and comprehensive theory of measurement validation, we have no scientific basis for the discipline of software measurement, a situation potentially disasterous for both practice and research."* Therefore, to have confidence in the utility of the many metrics that are proposed from research labs, it is crucial that they are validated.

The validation of software product metrics means convincingly demonstrating that:

1. The product metric measures what it purports to measure. For example, that a coupling metric is really measuring coupling.
2. The product metric is associated with some important external metric (such as measures of maintainability or reliability).
3. The product metric is an improvement over existing product metrics. An improvement can mean, for example, that it is easier to collect the metric or that it is a better predictor of faults.

There are two types of validation that are recognized [37]: *internal* and *external*. Internal validation is a theoretical exercise that ensures that the metric is a proper numerical characterization of the property it claims to measure. Demonstrating that a metric measures what it purports to measure is a form of theoretical validation. Typically, one defines the properties of the attribute that is to be measured, for example, the properties of module coupling. Then one demonstrates analytically that the product metric satisfies these properties. External validation involves empirically demonstrating points (2) and (3) above. Internal and external validation are also commonly referred to as *theoretical* and *empirical* validation respectively [80].

The true value of product metrics comes from their association with measures of important external attributes [64]. An external attribute is measured with respect to how the product relates to its

[1] Some authors distinguish between the terms 'metric' and 'measure' [3]. We use the term "metric" here to be consistent with prevailing international standards. Specifically, ISO/IEC 9126:1991 [63] defines a "software quality metric" as a "quantitative scale and method which can be used to determine the value a feature takes for a specific software product".

environment [39]. Examples of external attributes are testability, reliability and maintainability. Practitioners, whether they are developers, managers, or quality assurance personnel, are really concerned with the external attributes. However, they cannot measure many of the external attributes directly until quite late in a project's or even a product's life cycle. Therefore, they can use product metrics as leading indicators of the external attributes that are important to them. For instance, if we know that a certain coupling metric is a good leading indicator of maintainability as measured in terms of the effort to make a corrective change, then we can minimize coupling during design because we know that in doing so we are also increasing maintainability.

Given that there are many product metrics in existence today, it is necessary for a new product metric to demonstrate an improvement over existing metrics. For example, a new coupling metric may be a better predictor of maintainability than existing coupling metrics. Then it can be claimed to be useful. If its predictive power is the same as an existing metric but it is much easier to collect than existing metrics, or can be collected much earlier in the life cycle, then it is also an improvement over existing metrics.

Both types of validation are necessary. Theoretical validation requires that the software engineering community reach a consensus on what are the properties for common software product attributes. This consensus typically evolves over many years. Empirical validation is also time-consuming since many studies need to be performed to accumulate convincing evidence that a metric is valid. Strictly speaking, in a Popperian sense, we can only fail to empirically invalidate a metric [80]. Therefore, if many studies fail to invalidate a metric we have accumulating evidence that it has empirical validity. Furthermore, empirical validity is not a binary trait, but rather a degree. As the weight of evidence increases so does confidence, and increasing confidence means that the software engineering community is reaching a consensus; but validity can never be proven per se.

The above discussion highlights another important point. The validity of a product metric, whether theoretical or empirical, is not a purely objective exercise. The software engineering community must reach common acceptance of the properties of software product attributes. Furthermore, the conduct of empirical studies requires many judgement calls. However, by having rigorous standards one would expect objectivity to be high and results to be repeatable.

This chapter will present a comprehensive methodology for validating software product metrics. Our focus will be limited to *empirical* validation. We will assume that theoretical validation has already been performed, and therefore will not be addressed here.

2 Terminology

We will use the generic term *component* to refer to the unit of observation in a metrics validation study. This may mean a procedure, a file, a package, a class, or a method, to name a few examples. The methodology is applicable irrespective of the exact definition of a component.

When performing data analysis, it is typical to refer to the software product metrics as *independent variables*, and the measure of the external attribute as the *dependent variable*. Any of these variables may be *binary* or *continuous*. A binary variable has only two values. For example, whether a component is faulty or is not faulty. A continuous variable has many values (i.e., it is not limited to two).

We will refer to the individual or team performing the validation of a metric by the generic term *analyst*. The analyst may be a researcher or a practitioner, in academe or industry.

Metrics can be either *static* or *dynamic*. Static metrics can be collected from a static analysis of a software artifact, for example, a static analysis of source code. Dynamic metrics require execution of the software application in order to collect the metric values, which also makes them difficult to collect at early stages of the design. Also, the unit of measurement of a metric can vary. For example, in procedural applications a unit can be a module, a file, or a procedure. A procedure-level metric may be, say, cyclomatic complexity. If a file contains many procedures, then cyclomatic complexity can be defined as the median of the cyclomatic complexity of the procedures in the file, or even their total. Furthermore, metrics can be defined at the whole system level. For object-oriented software, metrics can be defined at the method-level, class-level, or the system level.

It is also important to clarify the terminology for counting faults. We will use the terms in the IEEE Standard Glossary [62]. A *mistake* is a human action that produces an incorrect result. The manifestation of a mistake is a software *fault*, which is an incorrect step, process, or data definition. A fault can result in a *failure*, which is an incorrect result. For instance, during testing the software may exhibit a failure if it produces an incorrect result compared to the specification.

3 The Utility of Validated Product Metrics

Ideally, once the research community has demonstrated that a metric or set of metrics is empirically valid in a number of different contexts and systems, organizations can take these metrics and use them. In practice, many organizations will adopt a set of metrics before adequate theoretical validation and before the convincing evidence has been accumulated. On the one hand this is typical behavior in software engineering, but on the other hand such early adopters are necessary if we ever hope to perform reasonable empirical validations.

Software organizations can use validated product metrics in at least three ways: to identify high risk software components early, to construct design and programming guidelines, and to make system level predictions. These are described further below.

3.1 Identifying Risky Components

The definition of a high-risk component varies depending on the context. For example, a high risk component is one that contains any faults found during testing [11][82], one that contains any faults found during operation [75], or one that is costly to correct after a fault has been found [1][5][12].

Recent evidence suggests that most faults are found in only a few of a system's components [41][67][91][95]. If these few components can be identified early, then an organization can take mitigating actions, such as focus fault detection activities on high-risk components, for example by optimally allocating testing resources [52], or redesigning components that are likely to cause field failures or be costly to maintain.

Predicting whether a component is high risk or not is achieved through a *quality model*. A quality model is a quantitative model that can be used to:

- Predict which components will be high risk. For example, some quality models make binary predictions as to whether a component is faulty or not-faulty [11][30][31][35][75][82].
- Rank components by their risk-proneness (in whatever way risk is defined). For instance, there have been studies that predict the number of faults in individual components (e.g., [72]), and that produce point estimates of maintenance effort (e.g., [66][84]). These estimates can be used for ranking the components.

An overview of a quality model is shown in Figure 1. A quality model is developed using a statistical or machine learning modeling technique, or a combination of techniques. This is done using historical data. Once constructed, such a model takes as input the values on a set of metrics for a particular component ($M_1 \ldots M_k$), and produces a prediction of the risk (say either high or low risk) for that component.

A number of organizations have integrated quality models and modeling techniques into their overall quality decision making process. For example, Lyu et al. [87] report on a prototype system to support developers with software quality models, and the EMERALD system is reportedly routinely used for risk assessment at Nortel [60][61]. Ebert and Liedtke describe the application of quality models to control the quality of switching software at Alcatel [27].

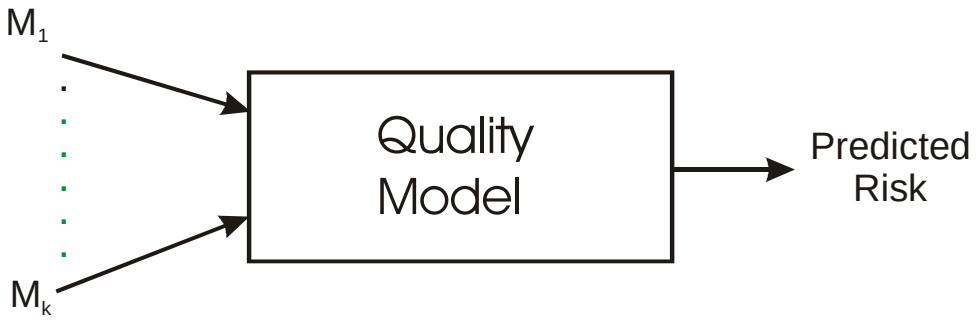

Figure 1: Definition of a quality model.

3.2 Design and Programming Guidelines

An appealing operational approach for constructing design and programming guidelines using software product metrics is to make an analogy with conventional statistical quality control: identify the range of values that are acceptable or unacceptable, and take action for the components with unacceptable values [79]. This means identifying thresholds on the software product metrics that delineate between 'acceptable' and 'unacceptable'. In summarizing their experiences using software product measures, Szentes and Gras [115] state "the complexity measures of modules may serve as a useful 'early warning' system against poorly written programs and program designs. .... Software complexity metrics can be used to pinpoint badly written program code or program designs when the values exceed predefined maxima or minima." They then argue that such thresholds can be defined subjectively based on experience. In addition to being useful during development, Coallier et al. [22] present a number of thresholds for procedural measures that Bell Canada uses for risk assessment during the *acquisition* of software products. The authors note that their thresholds identify 2 to 3 percent of all the procedures and classes for manual examination. Instead of experiential thresholds, some authors suggest the use of percentiles for this purpose. For example, Lewis and Henry [83] describe a system that uses percentiles on procedural measures to identify potentially problematic procedures. Kitchenham and Linkman [79] suggest using the 75th percentile as a cut-off value. More sophisticated approaches include identifying multiple thresholds simultaneously, such as in [1][5].

In an object-oriented context, thresholds have been similarly defined by Lorenz and Kidd as [86] "heuristic values used to set ranges of desirable and undesirable metric values for measured software." Henderson-Sellers [54] emphasizes the practical utility of object-oriented metric thresholds by stating that "An alarm would occur whenever the value of a specific internal metric exceeded some predetermined threshold." Lorenz and Kidd [86] present a number of thresholds for object-oriented metrics based on their experiences with Smalltalk and C++ projects. Similarly, Rosenberg et al. [102] have developed thresholds for a number of popular object-oriented metrics that are used for quality management at NASA GSFC. French [42] describes a technique for deriving thresholds, and applies it to metrics collected from Ada95 and C++ programs. Chidamber et al. [21] state that the premise behind managerial use of object-oriented metrics is that extreme (outlying) values signal the presence of high complexity that may require management action. They then define a lower bound for thresholds at the 80th percentile (i.e., at most 20% of the observations are considered to be above the threshold). The authors note that this is consistent with the common Pareto (80/20) heuristic.

3.3 Making System Level Predictions

Typically, software product metrics are collected on individual components for a single system. Predictions on individual components can then be aggregated to give overall system level predictions. For example, in two recent studies using object-oriented metrics, the authors predicted the proportion of faulty classes in a whole system [31][46]. This is an example of using predictions of fault-proneness for each class to draw conclusions about the overall quality of a system. One can also build prediction models of the total number of faults and fault density [36]. Similarly, another study used object-oriented metrics to predict the effort to develop each class, and these were then aggregated to produce an overall estimate of the whole system's development cost [16].

4 A Metrics Validation Methodology

The methodology that is described here is based on actual experiences, within software engineering and other disciplines where validation of measures is a common endeavor. The presentation below is intended to practical, in that we explain how to perform a validation and what are the issues that arise that must be dealt with. We provide guidance on the most appropriate ways to address common difficulties. In many instances we present a number of reasonable methodological options, discuss their advantages and disadvantages, and conclude by making a recommendation.

We will assume that the data analysis technique will be statistical, and will be a form of regression, e.g., logistic regression, or ordinary least squares regression. Many of the issues that are raised are applicable to other analysis techniques, in particular, machine learning techniques such as classification and regression trees; but not all. We focus on statistical techniques.

4.1 Overview of Methodology

The complete validation methodology is summarized below, with references to the relevant sections where each step is discussed. As can be seen, we have divided the methodology into three phases, planning, modeling, and post modeling. While the methodology is presented as a sequence of steps, in reality it is rarely a sequence of steps, and there is frequent iteration. We also assume that the analyst has a set of product metrics that need to be validated, rather than only one. If the analyst wishes to validate only one metric then the section below on variable selection would not be applicable.

Planning		
	Measurement of Dependent Variable	Section 4.3
	Selection of Data Analysis Technique	Section 4.4
	Model Specification	Section 4.5
	Specifying Train and Test Data Sets	Section 4.6
Statistical Modeling		
	Model Building	Section 4.7
	Evaluating the Validity of a Product Metric	Section 4.8
	Variable Selection	Section 4.9
	Building and Evaluating a Prediction Model	Section 4.10
	Making System Level Predictions	Section 4.11
Post Modeling		
	Interpreting Results	Section 4.12
	Reporting the Results of a Validation Study	Section 4.13

4.2 Theoretical Justification

The reason why software product metrics can be potentially useful for identifying high risk components, developing design and programming guidelines, and making system level predictions is exemplified by the following justification for a product metric validity study “There is a clear intuitive basis for believing that complex programs have more faults in them than simple programs” [94]. In general, however, there has not been a strong theoretical basis driving the development of traditional software product metrics. Specifically, Kearney et al. [68] state that “One of the reasons that the development of software complexity measures is so difficult is that programming behaviors are poorly understood. A behavior must be understood before what makes it difficult can be determined. To clearly state what is to be measured,

we need a theory of programming that includes models of the program, the programmer, the programming environment, and the programming task."

A theory is frequently stated at an abstract level, relating internal product *attributes* with external attributes. In addition, a theory should specify the *mechanism* which explains why such relationships should exist. To operationalize a theory and test it empirically the abstract attributes have to be converted to actual metrics. For example, consider Parnas' theory about design [96] which states that high cohesion within modules and low coupling across modules are desirable design attributes, in that a software system designed accordingly is easier to understand, modify, and maintain. In this case, it is necessary to operationally define coupling, understandability, modifiability, and maintainability into actual metrics in order to test the theory.

Recently, an initial theoretical basis for developing quantitative models relating product metrics and external quality metrics has been provided in [15], and is summarized in Figure 2. There, it is hypothesized that the internal attributes of a software component (i.e., structural properties), such as its coupling, have an impact on its cognitive complexity. Cognitive complexity is defined as the mental burden of the individuals who have to deal with the component, for example, the developers, testers, inspectors, and maintainers. High cognitive complexity leads to a component exhibiting undesirable external qualities, such as increased fault-proneness and reduced maintainability. Further detailed elaboration of this model has been recently made [32].

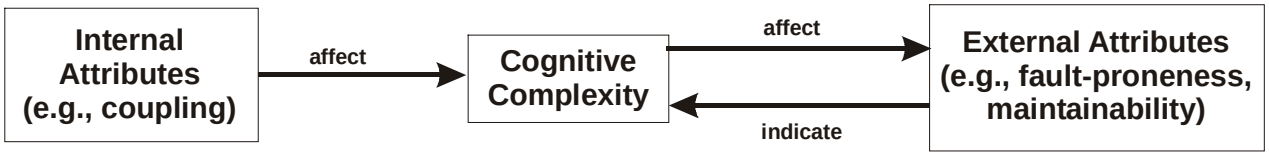

Figure 2: Theoretical basis for the development of object oriented product metrics.

Hatton has extended this general theory to include a mechanism for thresholds. He argues that Miller [90] shows that humans can cope with around 7 +/- 2 pieces of information (or chunks) at a time in short-term memory, independent of information content. He then refers to [56], where they note that the contents of long-term memory are in a coded form and the recovery codes may get scrambled under some conditions. Short-term memory incorporates a rehearsal buffer that continuously refreshes itself. He suggests that anything that can fit into short-term memory is easier to understand. Pieces that are too large or too complex overflow, involving use of the more error-prone recovery code mechanism used for long-term storage. In a subsequent article, Hatton [53] also extends this model to object-oriented development. Based on this argument, one can hypothesize threshold effects for many contemporary product metrics. El Emam has elaborated on this mechanism for object-oriented metrics [32].

It must be recognized that the above cognitive theory suggests only one possible mechanism of what would impact external metrics. Other mechanisms can play an important role as well. For example, some studies showed that software engineers experiencing high levels of mental and physical stress tend to produce more faults [43][44]. Mental stress may be induced by reducing schedules and changes in requirements. Physical stress may be a temporary illness, such as a cold. Therefore, cognitive complexity due to structural properties, as measured by product metrics, can never be the reason for all faults. However, it is not known whether the influence of software product metrics dominates other effects. The only thing that can reasonably be stated is that the empirical relationships between software product metrics and external metrics are not very likely to be strong because there are other effects that are not accounted for, but as has been demonstrated in a number of studies, they can still be useful in practice.

4.3 Measurement of Dependent Variables

Dependent variables in product metric validation studies are either *continuous* or *binary*.

In software product metric validation continuous variables are frequently counts. A count is characterized by being a non-negative integer, and hence is a discrete variable. It can be, for example, the number of faults found in the component or the effort to develop the component[2]. One can conceivably construct a variable such as fault density (number of faults divided by size), which is neither of the above. However, in such cases it is more appropriate to have size as an independent variable in a validation model and have the number of defects as the dependent variable.[3]

Binary variables are typically not naturally binary. For example, it is common for the analyst to designate components that have no faults as not-faulty and those that have a fault as faulty. This results in a binary variable. Another approach that is used is to dichotomize a continuous dependent variable around the median [29], or even on the third quartile [74].

4.3.1 Dichotomizing Continuous Variables

In principle, dichotomizing a continuous variable results in the loss of information in that the continuous variable is demoted to a binary one. Conceivably, then, validation studies using such a dichotomization are weaker, although it has not been demonstrated in a validation context how much weaker (it is plausible that in almost all situations the same conclusions will be drawn).

Generally, it is recommended that when the dependent variable is naturally continuous, that it remains as such and not be dichotomized. This will alleviate any lingering doubts that perhaps the results would have been different had no dichotomization been performed. The only reasonable exception in favor of using binary variables is related to the reliability of data collection for the system being studied. We will illustrate this through a scenario.

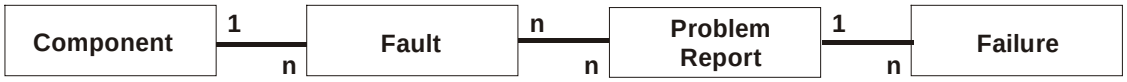

Figure 3: ER model showing a possible relationship between failures and components. The acronym PR stands for *Problem Report*.

In many organizations whenever a failure is observed (through testing or from a customer) a Problem Report (PR) is opened. It is useful to distinguish between a failure and a failure instance. Consider Figure 3. This shows that each failure instance may be associated with only a single PR, but that a PR can be associated with multiple failure instances. Multiple failure instances may be the same failure, but detected by multiple users, and hence they would all be matched to a single PR. A fault occurs in a single component, and each component may have multiple faults. A PR can be associated with multiple faults, possibly in the same component or across multiple components.

Counting inaccuracies may occur when a PR is associated with multiple faults in the same component. For example, say because of schedule pressures, the developers fix all faults due to the same PR at the same time in one delta. Therefore, when data is extracted from a version control system it appears as one fault. Furthermore, it is not uncommon for developers to fix multiple faults due to *multiple* PRs in the same delta. In such a case it is not known how many faults were actually fixed since each PR may have been associated to multiple faults itself.

The above scenarios illustrate that, unless the data collection system is fine grained and followed systematically, one can reliably say that there was at least one fault, but not the exact number of faults that were fixed per component. This makes an argument for using a binary variable such as faulty/not-faulty.

---

[2] Although effort in principle is not a discrete count, in practice it is measured as such. For example, one can measure effort at the granularity of minutes or hours.

[3] Fault density may be appropriate as a descriptive statistic, however.

4.3.2 Surrogate Measures

Some studies use surrogate measures. Good examples are a series of studies that used code churn as a surrogate measure of maintenance effort [55][84][85]. Code churn is the number of lines added, deleted, and modified in the source code to make a change. It is not clear whether code churn is a good measure of maintenance effort. Some difficult changes that consume a large amount of effort to make may result in very few changes in terms of LOC. For example, when performing corrective maintenance one may consume a considerable amount of time attempting to isolate the cause of the defect, such as when tracking down a stray pointer in a C program. Alternatively, making the fix may consume a considerable amount of effort, such as when dealing with rounding off errors in mathematical computations (this may necessitate the use of different algorithms altogether which need to be implemented). Furthermore, some changes that add a large amount of code may require very little effort (e.g., cloning error-checking functionality from another component, or deletion of unused code). One study of this issue is presented in the appendix (Section 6). The results suggest that using code churn as a surrogate measure of maintenance effort is not strongly justifiable, and that it is better not to follow this practice.

Therefore, in summary, the dependent variable ought not to be dichotomized unless there is a compelling reason to do so. Furthermore, analysts should be discouraged from using surrogate measures, such as code churn, unless there is evidence that they are indeed good surrogates.

4.4 Selection of Data Analysis Technique

The selection of a data analysis technique is a direct consequence of the type of the dependent variable that one is using. In the discussion that follows we shall denote the dependent variable by y and the independent variables by x_i.

4.4.1 A Binary Dependent Variable

If y is binary, then one can construct a logistic regression (LR) model. The form of a LR model is:

$$\pi = \frac{1}{1 + e^{-\left(\beta_0 + \sum_{i=1}^{k} \beta_i x_i\right)}} \qquad \textbf{Eqn. 1}$$

where π is the probability of a component being high-risk, and the x_i's are the independent variables. The β parameters are estimated through the maximization of a log-likelihood [59].

4.4.2 A Continuous Dependent Variable That is Not a Count

As noted earlier, a continuous may or may not be a count variable. If y is not a count variable, then one can use the traditional ordinary least squares (OLS) regression model[4]. The OLS model takes the form:

$$y = \beta_0 + \sum_{i=1}^{k} \beta_i x_i \qquad \textbf{Eqn. 2}$$

4.4.3 A Continuous Dependent Variable That is a Count

If y is a (discrete) count variable, such as the number of faults or development effort, it may seem that a straight forward approach is to also build an OLS model as in Eqn. 2. This is not the case.

In principle, using an OLS model is problematic. First, an OLS model can predict negative values for faults or effort, and this is clearly meaningless. For instance, this is illustrated in Figure 4, which shows the relationship between the number of methods (NM) in a class and the number of post-release faults for the Java system described in [46] as calculated using OLS regression. As can seen, if this model is used

[4] Of course the analyst should ensure that its assumptions are met. But it would be a reasonable starting point.

to predict the number of faults for a class with two methods the number of faults predicted will be negative.

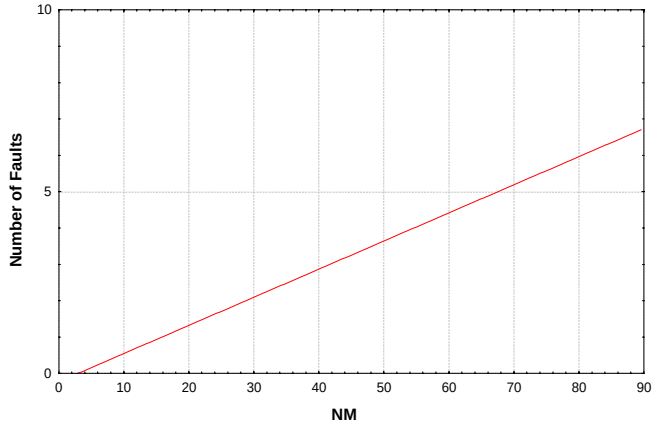

Figure 4: Relationship between the number of methods in a class and the number of faults.

Furthermore, such count variables are decidedly non-normal, violating one of the assumptions used in deriving statistical tests for OLS regression. Marked heteroscedasticity is also likely which can affect the standard errors: they will be smaller than their true value, and therefore t-tests for the regression coefficients will be inflated [45]. King [77] notes that these problems lead to coefficients that have the wrong size and may even have the wrong sign.

Taking the logarithm of y may alleviate some of these problems of the OLS model. Since y can have zero values, one can add a small constant, typically 0.01 or 0.5: $\log(y+c)$ [77]. An alternative is to take the square root of y [65][77]. The square root transformation, however, would result in an awkward model to interpret. With these transformations it is still possible to obtain negative predictions. This creates particular problems when the square root transformation is used. For example, Figure 5 shows the relationship between NM and the square root of faults for the same system. In a prediction context, this OLS model can predict negative square root number of faults for very small classes.

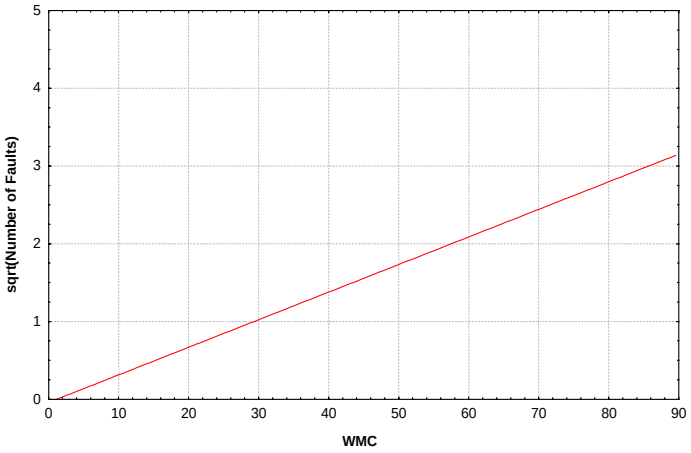

Figure 5: Relationship between the number of methods and the square root of faults.

Also, in many cases the resulting distribution still has its mode[5] at or near the minimal value after such transformations. For example, Figure 6 shows the original fault counts found during acceptance testing for one large system described in [35].[6] As can be seen, the mode is still at the minimal value even after the logarithmic or square root transformation. Another difficulty with such transformations occurs when making system level predictions. The Jensen inequality (see [112]) states for a convex function $g(y)$ that $E(g(y)) \geq g(E(y))$. Given that the estimates from OLS are expected values conditional on the x values, for the above transformations the estimates produced from the OLS regression will be systematically biased.

In addition, when y is a count of faults, an OLS model makes the unrealistic assumption that the difference between no faults and one fault is the same as the difference between say 10 faults and 11 faults. In practice, one would reasonably expect a number of different effects:

- The probability of a fault increases as more faults are found. This would mean that there is an intrinsic problem with the design and implementation of a component and therefore the incidence of a fault is an indicator that there are likely to be more faults. Furthermore, one can argue that as more fixes are applied to a component, new faults are introduced by the fixes and therefore the probability of finding a subsequent fault increases. Evidence of this was presented in a recent study [9] whereby it was observed that components with more faults pre-release also tend to have more faults post-release, leading to the conclusion that the number of faults already found is positively related of the number of faults to be found in the future. This is termed *positive contagion*.
- The probability of a fault decreases as more faults are found. This would be based on the argument that there are a fixed number of faults in a component and finding a fault means that there are fewer faults left to find and these are likely to be the more difficult ones to detect. This is termed *negative contagion*.

[5] The mode is the most frequent category [121].

[6] In the study described in [35] only a subset of the data was used for analysis. The histograms shown in Figure 6 are based on the complete data set consisting of 11012 components.

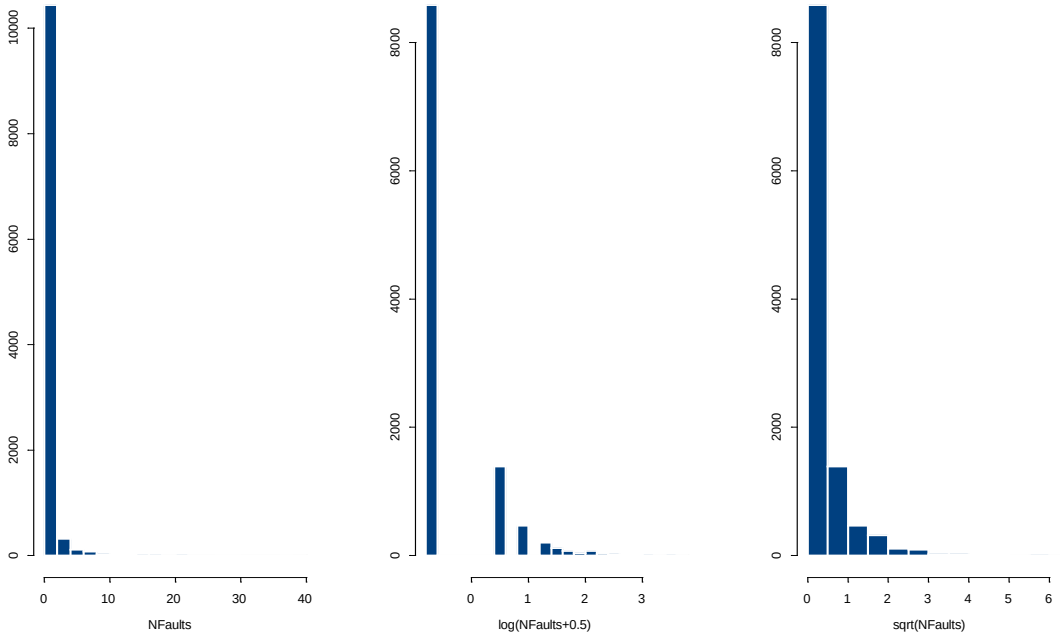

Figure 6: The effect of different transformations of the number of faults dependent variable. The x-axis shows the value of faults or transformed faults, and the y-axis shows the frequency.

A more principled approach is to treat the y variable as coming from a Poisson distribution, and perform a Poisson regression (PR) [123]. For example, Evanco presents a Poisson model to predict the number of faults in Ada programs [36], and Briand and Wuest use Poisson models to predict development effort for object-oriented applications [16].

One can argue that when the y values are almost all large the Poisson distribution becomes approximately normal, suggesting that if the Poisson distribution is appropriate for modeling the count variable, then OLS regression will do fine. King [77] defined large as being almost every observation having a count greater than 30. This is unlikely to be true for fault counts, and for effort Briand and Wuest [16] have made the explicit point that large effort values are rare.

A Poisson distribution assumes that the variance is equal to the mean – *equidispersion*. At least for object-oriented metrics and development effort, a recent study has found overdispersion [16] – the variance is larger than the mean. Parameter estimates when there is overdispersion are known to be inefficient and the standard errors are biased downwards [19]. Furthermore, the Poisson distribution is non-contagious, and therefore would not be an appropriate model, a priori, for fault counts.

The negative binomial distribution allows for overdispersion, hence the conditional variance can be larger than the conditional mean in a negative binomial regression (NBR) model. Furthermore, NBR can model positive contagion. An important question is whether dependent variables better fit a negative binomial distribution or a Poisson distribution. To answer this question we used the number of faults data for all 11012 components in the procedural system described in [35], and the 250 classes for the object-oriented system described in [46]. Using the actual data set, maximum likelihood estimates were derived for a Poisson and a negative binomial distribution, and then these were used as starting values in a Levenberg-Marquardt optimization algorithm [100] to find the distribution parameters that best fit the data, optimizing on a chi-square goodness-of-fit test. The actual data sets were then compared to these optimal distributions using a chi-square test. In both cases the results indicate that the negative binomial distribution is a better fit to the fault counts than the Poisson distribution. Therefore, at least based on this

initial evaluation, one may conclude that NBR models would be more appropriate than Poisson regression models for fault counts.

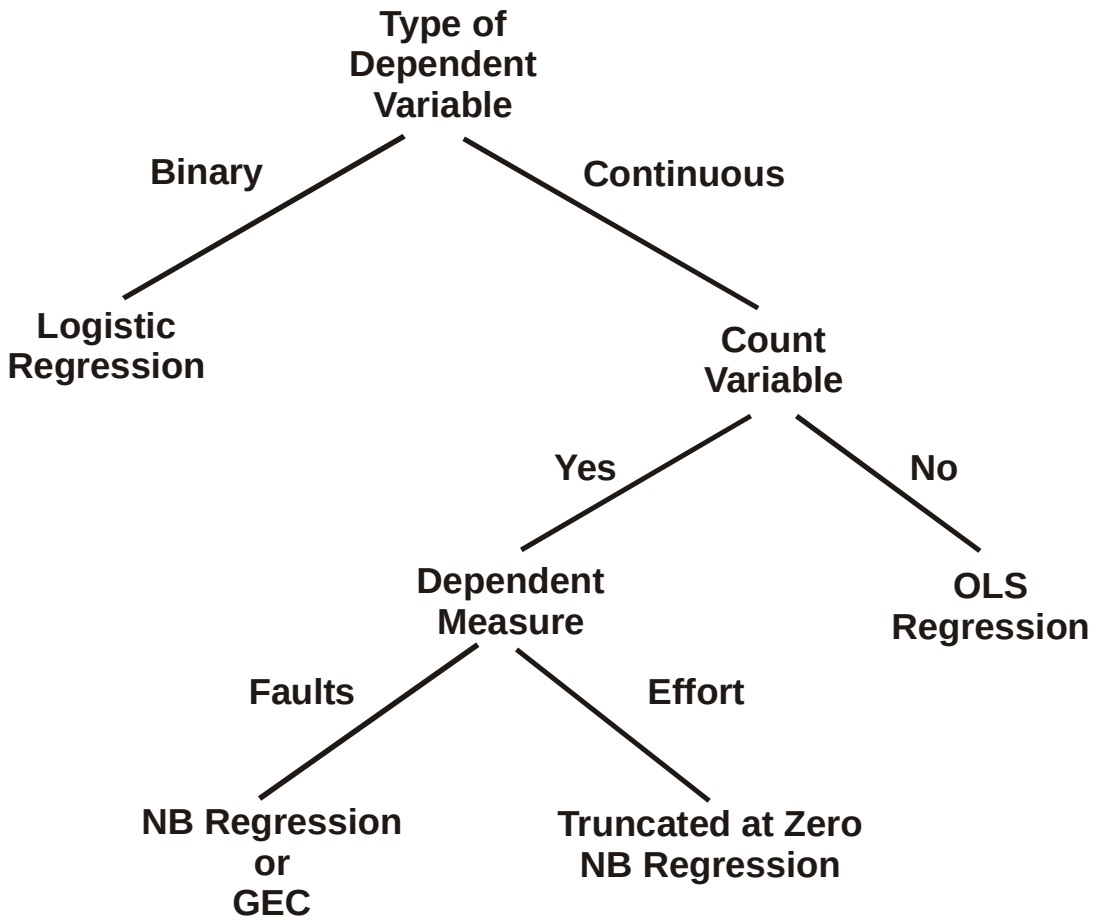

Figure 7: Decision tree for selecting an analysis method. The non-terminal nodes are conditions. Each arc is labeled with the particular value for the condition. Terminal nodes are the recommended analysis technique(s).

NBR would still have deficiencies for modeling faults since it does not allow for the possibility of negative contagion. A general event count (GEC) model that accommodates both types of contagion has been presented in [78]. Until there is a clearer understanding of how the incidence of a fault affects the incidence of another fault, one should use both modeling approaches and compare results: the NBR and the GEC.

In the case of development effort data, it is meaningless to have an effort value of zero. However the NBR model assumes that zero counts exist. To alleviate this, truncated-at-zero NBR is the most appropriate modeling technique [123]. This assumes that counts start from one.

Two simulation studies provide further guidance about the appropriate modeling techniques to use. One simulation using a logarithmic transformation of a count y and OLS regression found that this produced biased coefficient estimates, even for large samples [77]. It was also noted that the estimates were not consistent, and that the Poisson regression model generally performed better. Another Monte Carlo simulation [114] found that the Type I error rates[7] of the PR model were disturbingly large; the OLS

[7] This is the probability of incorrectly rejecting the null hypothesis.

regression model with and without a square root transformation yielded Type I error rates almost equal to what would be expected by chance; and that the negative binomial model also had error rates equal to the nominal level. Even though the second simulation seems encouraging for OLS regression models, they would still not be advisable because of their inability to provide unbiased system level predictions and due to the possibility of giving negative predictions. The PR model is not recommended based on the arguments cited above as well as the inflated Type I error rate. It would therefore be reasonable to use at least the NBR model for validating product metrics when the y variable is a fault count, and ideally the community should investigate the utility of the additional modeling of negative contagion through the GEC.

The above exposition has attempted to provide clear guidance and justifications for the appropriate models to use, in principle. We have summarized these guidelines in the decision tree of Figure 7. Admittedly, if one were to look at the software product metrics validation literature today, one would see a predominant use of OLS regression and LR.

4.5 Model Specification

When using a statistical technique it is necessary to specify the model a priori (this is not the case for many machine learning techniques). Model specification involves making five decisions on:

- Whether to use principal components.
- The functional form of the relationship between the product metrics and the dependent variable.
- Modeling interactions.
- Which confounding variables to include.
- Examination of thresholds.

4.5.1 The Use of Principal Components

Principal components analysis (PCA) [76] is a data reduction technique that can be used to reduce the number of product metrics. Therefore, if one has k product metrics, PCA will group them into z orthogonal dimensions such that $z < k$ where all the metrics within each dimension are highly correlated amongst themselves but have relatively small correlations with metrics in a different dimension. The logic is that each dimension represents a single underlying construct that is responsible for the observed correlations [93].

Once the dimensions are identified, there are typically two ways to produce a single metric for each dimension. The first is to sum the metrics within each dimension. The second is to use a weighted sum. The latter are often referred to as “domain metrics” [73]. Instead of using the actual metrics during validation, one can then use the domain metrics, as was done, for example, in [73][82]. The advantage of this approach is that it practically eliminates the problem of collinearity (discussed below).

Domain metrics have some nontrivial disadvantages as well. First, some studies report that there is no difference in the accuracy of a model using domain metrics versus the original metrics [82]. Another study [69] concluded that principal components are unstable across different products in different organizations. This suggests that the definition of a domain metric will be different across studies, making the accumulation of knowledge about product metrics rather tenuous. Furthermore, as noted in [40], such domain metrics are difficult to interpret and act upon by practitioners.

In general, the use of principal components or domain metrics as variables in models is not advised. In addition to the above disadvantages, there are other ways of dealing with multicollinearity, therefore there is no compelling reason for using domain metrics.

4.5.2 Specifying the Functional Form

It is common practice in software engineering validation studies to assume a linear relationship between the product metrics and the dependent variable. However, this is not always the best approach. Most product metrics are counts, and they tend to have a skewed distribution, which can be characterized as lognormal (or more appropriately negative binomial since the counts are discrete). This presents a

problem in practice in that many analysis techniques assume a normal distribution, and therefore just using the product metric as it is would be considered a violation of the assumptions of the modeling technique. To remedy this it is advisable to perform a transformation on the product metrics, typically a logarithmic transformation will be adequate. Since with product metrics there will be cases with values of zero, it is common to take $\log(x_i + 0.5)$. Taking the logarithm will also in many instances linearize the relationship between the product metrics and the dependent variable. For example, if one is building a LR model with a metric M and size S, then the model can be specified as:

$$\pi = \frac{1}{1 + e^{-(\beta_0 + \beta_1 M' + \beta_2 S')}} \qquad \textbf{Eqn. 3}$$

where $M' = \log(M + 0.5)$ and $S' = \log(S + 0.5)$. In practice, it is advised that a pair of models with transformed and untransformed variables are built, and then the results can be compared. Alternatively, a more principled approach is to check for the linearity assumption and to identify appropriate transformations that can be followed, such as the alternating conditional expectation algorithm for OLS regression which attempts to find appropriate transformations [10] (also see [120] for a description of how to use this algorithm to decide on transformations), and partial residual plots for LR [81].

4.5.3 Specifying Interactions

It is not common in software engineering validation studies to consider interactions. An interaction specifies that the impact of one independent variable depends on the level of the other independent variable. To some extent this is justified since there is no strong theory predicting interaction effects. However, it is also advisable to at least consider models with interactions and see if the interaction variable is statistically significant. If it is not then it is safe to assume that there are no interactions. If it is, then interactions must be included in the model specification. When there are many product metrics to consider the number of possible interactions can be very large. In such cases, it is also possible to use a machine learning technique to identify possible interactions, and then only include the interactions identified by the machine learning technique. A typical way for specifying interactions is to add a multiplicative term:

$$\pi = \frac{1}{1 + e^{-(\beta_0 + \beta_1 M' + \beta_2 S' + \beta_3 M' S')}} \qquad \textbf{Eqn. 4}$$

If the interaction coefficient, β_3, is statistically significant, then this indicates that there is an interaction effect.

4.5.4 Including Confounding Variables in the Specification

It is also absolutely critical to include potential confounding variables as additional independent variables when validating a product metric. A confounding variable can distort the relationship between the product metric and the dependent variable. One can easily control for confounding variables by including them as additional variables in the regression model.

A recent study [34] has demonstrated a confounding effect of class size on the validity of object-oriented metrics. This means that if one does not control the effect of class size when validating metrics, then the results would be quite optimistic. The reason for this is illustrated in Figure 8. Size is correlated with most product metrics (path (c)), and it is also a good predictor of most dependent variables (e.g., bigger components are more likely to have a fault and more likely to take more effort to develop – path (b)).

For example, there is evidence that object-oriented product metrics are associated with size. In [18] the Spearman rho correlation coefficients go as high as 0.43 for associations between some coupling and cohesion metrics with size, and 0.397 for inheritance metrics, and both are statistically significant (at an alpha level of say 0.1). Similar patterns emerge in the study reported in [15], where relatively large correlations are shown. In another study [20] the authors display the correlation matrix showing the Spearman correlation between a set of object-oriented metrics that can be collected from Shlaer-Mellor designs and C++ LOC. The correlations range from 0.563 to 0.968, all statistically significant at an alpha

level 0.05. This also indicates very strong correlations with size. The relationship between size and faults is clearly visible in the study of [20], where the Spearman correlation was found to be 0.759 and statistically significant. Another study of image analysis programs written in C++ found a Spearman correlation of 0.53 between size in LOC and the number of faults found during testing [50], and was statistically significant at an alpha level of 0.05. Briand et al. [18] find statistically significant associations between 6 different size metrics and fault-proneness for C++ programs, with a change in odds ratio going as high as 4.952 for one of the size metrics.

Therefore, if an association is found between a particular metric and the dependent variable, this may be due to the fact that higher values on that metric also mean higher size values. Inclusion of size in the regression model, as we did in Eqn. 3, is straight-forward. This allows for a statistical; adjustment of the effect of size. It is common that validation studies do not control for size. For object-oriented metrics, validation studies that did not control for size include [8][15][18][51][116]. It is not possible to draw strong conclusions from such studies.

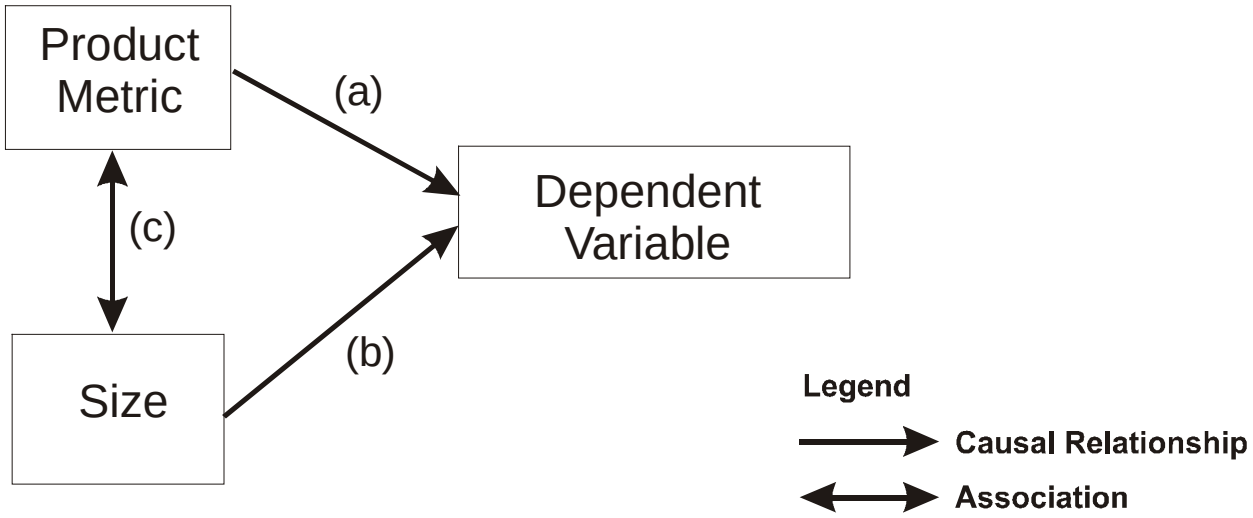

Figure 8: Illustration of counfounding effect of class size.

Therefore, in summary it is necessary to adjust for confounding variables. While there are many potential confounders, size at a minimum must be controlled since it is easy to collect in a metrics validation study. Other confounding variables are discussed in Section 5.

4.5.5 Specifying Thresholds

Given the practical and theoretical implications of thresholds, it is also important to evaluate thresholds for the product metrics being validated. In the case of a LR model, a threshold can be defined as [119]:

$$\pi = \frac{1}{1+e^{-(\beta_0+\beta_1 S+\beta_2 (M-\tau) I_+ (M-\tau))}} \qquad \textbf{Eqn. 5}$$

where:

$$I_+(z) = \begin{cases} 0 & if\ z \le 0 \\ 1 & if\ z > 0 \end{cases} \qquad \textbf{Eqn. 6}$$

and τ is the metric's threshold value. One can also specify the model with transformed variables.

A recent study [33] has shown that there are no size thresholds for object-oriented systems. Therefore size can be kept as a continuous variable in the model. However, this is not a foregone conclusion, and further studies need to verify this for object-oriented and procedural systems.

The difference between the no threshold and threshold model is illustrated in Figure 9. For the threshold model the probability of a being high risk only starts to increase once the metric is greater than the threshold value, τ .

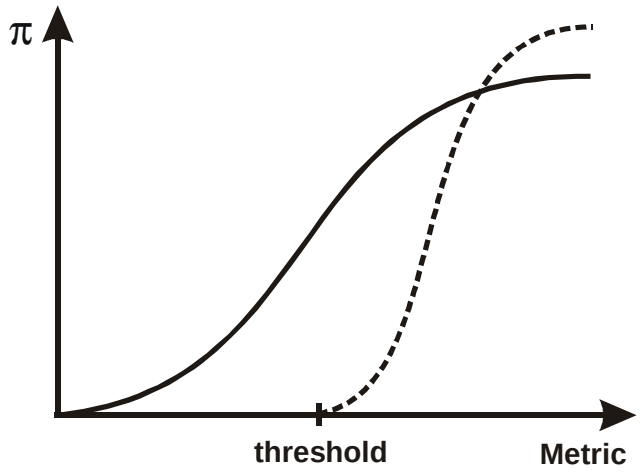

Figure 9: Relationship between the metric M and the probability of being high risk for the threshold and no threshold models. This is the bivariate relationship assuming size is kept constant.

To estimate the threshold value, τ , one can maximize the log-likelihood for the model in Eqn. 5. Ulm [119] presents an algorithm for performing this maximization.

Once a threshold is estimated, it should be evaluated. This is done by comparing the no-threshold model with the threshold model. Such a comparison is, as is typical, done using a likelihood ratio statistic (for example see [59]). The null hypothesis being tested is:

$$H_0 : \tau \leq M^{(1)} \qquad \textbf{Eqn. 7}$$

where $M^{(1)}$ is the smallest value for the measure M in the data set. If this null hypothesis is not rejected then it means that the threshold is equal to or below the minimal value. When the product metric is a count, this is exactly like saying that the threshold model is the same as the no-threshold model. Hence one can conclude that there is no threshold. The alternative hypothesis is:

$$H_1 : \tau > M^{(1)} \qquad \textbf{Eqn. 8}$$

which would indicate that a threshold effect exists. In principle, one can also specify models with thresholds for multiple product metrics and using other regression techniques.

4.6 Specifying Train and Test Data Sets

Although the issue of train and test data sets becomes important during the construction and evaluation of prediction models (see Section 4.10), it must be decided upon at the early stages of metrics validation. It is known that if an analyst builds a prediction model on one data set and evaluates its prediction accuracy on the same data set, then the accuracy will be artificially inflated. This means that the accuracy results obtained during the evaluation will not generalize when the model is used on subsequent data sets, for example, a new system. Therefore, the purpose of deciding on train and test sets is to come up with a strategy that will allow the evaluation of accuracy in a generalizeable way.

If the analyst has two data sets from subsequent releases of a particular system, then the earliest release can be designated as the training data set, and the later release as the test data set. If there are more than two releases, then the analyst could evaluate the prediction model on multiple test sets. Sometimes data sets from multiple systems are available. In such cases one of the data sets is designated as the training data set (usually the one with the largest number of observations) and the remaining system data sets as the test sets.

If the analyst has data on one system, then a common strategy that is used is to randomly split the data set into a train and a test set. For example, randomly select one third of the observations as the test set and the remaining two thirds is your training set. All model building is performed on the training data set. Evaluation of accuracy is then performed by comparing the predicted values from the model to the actual values in the test set.

Another approach is to use cross-validation. There are two forms of cross-validation: leave-one-out and k-fold cross-validation. In leave-one-out cross-validation the analyst removes one observation from the data set, builds a model with the remaining $n-1$ observations, and evaluates how well the model predicts the value of the observation that is removed. This process is repeated each time removing a different observation. Therefore, one builds and evaluates n models. With k-fold cross-validation the analyst randomly splits the data set into k approximately equally sized groups of observations. A model is built using k-1 groups and its predictions evaluated on the last group. This process is repeated each time by removing a different group. So for example if k is ten, then 10 models are constructed and validated.

Finally, a more recent approach that has been used is called bootstrapping [28]. A training set is constructed by *sampling with replacement*. Each sample consists of n observations from the data set, where the data set has n observations. Some of the observations may therefore be duplicates. The remaining observations that were not sampled at all are the test set. A model is built with the training set and evaluated on the test set. This whole process is repeated say 500 times, each time sampling with replacement n observations. Therefore one builds and evaluates 500 models.

In general, if one's data set is large (say more than 200 observations), then one can construct a train and test set through random splits. If the data set is smaller than that, one set of recommendations for selecting amongst these alternatives has been provided in [122]:

- For sample sizes greater than 100, use cross-validation, leave-one-out or 10-fold cross-validation.
- For sample sizes less than 100, use leave-one cross validation
- For sample sizes that are less than 50 one can use either the bootstrap approach or the leave-one-out approach.

There are alternative strategies that can be used, and these are discussed in [122]. However, at least in the software product metrics, the above are the most commonly seen approaches.

4.7 Model Building

During the construction of models to validate metrics, it is necessary to pay attention to issues of model stability and evaluating the goodness-of-fit of the model.

4.7.1 Model Building Strategy

A general strategy for model building consists of two stages. The first is to develop a model with each metric and the confounders. This model will allow the analyst to determine which metrics are related to the dependent variable (see Section 4.8). Recall that building a model without confounders does not provide useful information and therefore should be avoided. In the second stage a subset of the metrics that are related with the dependent variable are selected to build a prediction model. Variable selection is discussed in 4.9 and evaluating the prediction model is discussed in 4.10.

Below we present some of the important issues that arise when building a model, in either of the above stages.

4.7.2 Diagnosing Collinearity

One of the requirements for properly interpreting regression models is that no one of the independent variables are perfectly linearly correlated to one or more other independent variables. Perfect linear correlation is referred to as perfect *collinearity*. When there is perfect collinearity then the regression surface is not even defined. Perfect collinearity is rare, however, and therefore one talks about the *degree* of collinearity. Recall that confounders are associated with the product metric by definition, and therefore one should not be surprised if strong collinearity exists. The larger the collinearity, the greater the standard errors of the coefficient estimates. One implication of this is that the conclusions drawn about the relative impacts of the independent variables based on the regression coefficient estimates from the sample are less stable.

A commonly used diagnostic to evaluate the extent of collinearity is the condition number [6][7]. If the condition number is larger than a threshold, typically 30, then one can conclude that there exists severe collinearity and remedial actions ought to be taken. One possible remedy is to use ridge regression techniques. Ridge regression has been defined for OLS models [57][58], and extended for LR models [104][105].

4.7.3 Influential Observations

Influence analysis is performed to identify influential observations (i.e., ones that have a large influence on the regression model). This can be achieved through deletion diagnostics. For a data set with n observations, estimated coefficients are recomputed n times, each time deleting exactly one of the observations from the model fitting process. For OLS regression, a common diagnostic to detect influential observations is Cook's distance [24][25]. For logistic regression, one can use Pergibon's $\Delta\beta$ diagnostic [97].

It should be noted that an outlier is not necessarily an influential observation [103]. Therefore, approaches such as those described in [18] for identifying outliers in multivariate models will not necessarily identify influential observations. Specifically, they [18] use the Mahalanobis distance from the centroid, removing each observation in turn before computing the centroid. This approach has some deficiencies. For example, an observation may be further away from the other observations but may be right on the regression surface. Furthermore, influential observations may be clustered in groups. For example, if there are say two independent variables in our model and there are five out of the n observations that have exactly the same values on these two variables, then these 5 observations are a group. Now, assume that this group is multivariately different from all of the other $n-5$ observations. Which one of these 5 would be considered an outlier ? A leave-one-out approach may not even identify any one of these observations as being different from the rest since in each run the remaining 4 observations will be included in computing the centroid. Such masking effects are demonstrated in [6].

The identification of outliers is useful, for example, when interpreting the results of logistic regression: the estimate of the change in odds ratio depends on the sample standard deviation of the variable. If the standard deviation is inflated due to outliers then you will also get an inflated change in odds ratio.

When an influential observation or an outlier is detected, it would be prudent to inspect that observation to find out why it is problematic rather than just discarding it. It may be due to a data entry error, and fixing that error results in retaining the observation.

4.7.4 Goodness of fit

Once the model is constructed it is useful to evaluate its goodness-of-fit. Commonly used measures of goodness-of-fit are R^2 for OLS and pseudo-R^2 measures of logistic regression. While R^2 has a straightforward interpretation in the context of OLS, for logistic regression this is far from the case. Pseudo-R^2 measures of logistic regression tend to be small in comparison to their OLS cousins. Menard gives a useful overview of the different pseudo-R^2 coefficients that have been proposed [88], and recommends the one described by Hosmer and Lemeshow [59].

4.8 Evaluating the Validity of a Product Metric

The first step in validating a metric is to examine the relationship between the product metric and the dependent variable after controlling for confounding effects. The only reason for building models that do not control for confounding effects is to see how big the confounding effect is. In such a case one would build a model without the confounding variable, and a model with the confounding variable, and see how much the estimated parameter for the product metric changes. However, models without confounder control do not really tell us very much about the validity of the metric.

There are two aspects to the relationship between a product metric and the dependent variable: statistical significance and the magnitude of the relationship. Both are important, and both concern the estimated parameters in the regression models. Statistical significance tells us the probability of getting an estimated parameter as large as the one actually obtained if the true parameter was zero.[8] If this probability is larger than say 0.05 or 0.1, then we can have confidence that it is larger than zero. The magnitude of the parameter tells us how much influence the product metric has on the dependent variable. The above two are not necessarily congruent. It is possible to have a large parameter and no statistical significance. This is more likely to occur if the sample size is small. It is also possible to have a statistically significant parameter that is rather small. This is more likely to occur with large samples.

Most parameters are interpretable by themselves. However, sometimes it is also useful to compare the parameters obtained by different metrics. Direct comparison of regression parameters is incorrect because the product metrics are typically measured on different units. Therefore, they need to be normalized. In ordinary least squares regression one would use $\beta_i' = \sigma_i \beta_i$, which measures the change in the y variable when there is a one standard deviation change in the x_i variable (the σ_i value is the standard deviation of the x_i variable). In logistic regression one would use the change in odds ratio given by $\Delta\Psi = e^{\beta_i \sigma_i}$, which is interpreted as the change in odds by increasing the value of x_i by one standard deviation. Such normalized measures are appropriate for comparing different parameters, and finding out which product metric has the biggest influence on the dependent variable.

Since most software engineering studies are small sample studies, if a metric's parameter is found to be statistically significant, then it is considered to be "validated". Below we discuss the interpretation of insignificant results (see Section 4.12).

4.9 Variable Selection

Now that we have number of validated metrics, we wish to determine whether we can build a useful prediction model. A useful prediction model should contain the minimal number of "best" validated metrics; the best being interpreted as the metrics that have the biggest impact on the dependent variable. For example, in the case of logistic regression, this will be the variable with the largest change in odds ratio. Finding the "best" metrics is typically not a simple determination since many product metrics are strongly correlated with each other. For example, it has been postulated that for procedural software, product metrics capture only five dimensions of program complexity [93]: control, volume (size), action, effort, and modularity. If most of the "best" metrics are correlated with each other or are measuring the same thing, a model incorporating the strongly correlated variables will not be stable due to high collinearity. Therefore, the metrics selected should also be orthogonal.

The first caution is that one ought not use automatic variable selection techniques to identify the "best" variables. There have been a number of studies that present compelling evidence that automatic selection techniques tend to have a large tendency to select "noise" variables, especially when the set of variables is highly correlated. This means that the final model with the automatically selected variables will not tell us which are the best metrics, and will likely be unstable in future data sets. For example, a Monte Carlo simulation of forward selection indicated that in the presence of collinearity amongst the independent variables, the proportion of 'noise' variables that are selected can reach as high as 74% [26].

[8] In principle, other values than zero can be used. However, in software engineering studies testing for the zero value is almost universal.

It is clear that, for instance, in recent object-oriented metrics validation studies many of the metrics were correlated [15][18]. Harrell and Lee [48] note that when statistical significance is the sole criterion for including a variable the number of variables selected is a function of the sample size, and therefore tends to be unstable across studies. Furthermore, some general guidelines on the number of variables to consider in an automatic selection procedure for logistic regression are provided in [49]. Typically, software engineering studies include many metrics whose number is larger than those guidelines [15][18]. Therefore, the variables selected through such a procedure should not be construed as the best product metrics. In fact, automatic selection should only be considered as an exploratory technique whose results require further confirmation. Their use is not necessary though as easy alternatives can be used from which one can draw stronger conclusions.

If there are only a few metrics that have been found to be valid, then a simple correlation matrix would identify which metrics are strongly associated with each other. It is then easy to select a subset of metrics that both have the strongest relationship with the dependent variable and that have a weak correlation to each other.

If there are many validated metrics, one commonly used approach is to perform a principal components analysis [76]. The use of PCA for variable selection is different from its use for defining domain metrics discussed earlier (see Section 4.5.1). One can take the subset of metrics that are validated and perform a principal components analysis with them. The dimensions that are identified can then be used as the basis for selecting metrics. The analyst then selects one metric from each factor that is most strongly associated with the dependent variable.

4.10 Building and Evaluating a Prediction Model

Now that variables have been selected, one can develop a prediction model, following the guidelines above for model building. The prediction model must also include all the confounding variables considered earlier. Evaluation of the accuracy of the prediction model must follow. The approach for evaluation will depend on whether the analyst has only one data set or a train and test data set. These options were discussed above.

An important consideration is what coefficient to use to characterize prediction accuracy. A plethora of coefficients have been used in the software engineering literature. The coefficients differ depending on whether the dependent variable is binary or continuous and whether the predictions are binary or discrete.

4.10.1 Binary Dependent Variable and Binary Prediction

This situation occurs if the dependent variable is binary and the modeling technique that one is using makes binary predictions. For example, if one is using a classification tree algorithm, then the predictions are binary.

A plethora of coefficients have been used in the literature for evaluating binary predictions, for example, a chi-square test [1][5][82][108], sensitivity and specificity [1], proportion correct [1][82][107] (also called correctness in [98][99]), type I and type II misclassifications [70][71], true positive rate [11][15][18], and Kappa [15][18]. A recent comprehensive review of these measures [35] recommended that they should not be used as evaluative measures because either: (i) the results they produce depend on the proportion of high-risk components in the data set and therefore are not generalizeable except to other systems with the same proportion of high-risk components, or (ii) can only provide consistently unambiguous results if used in combination with other measures.

A commonly used notation for presenting the results of a binary accuracy evaluation is shown below (this is known as a *confusion matrix*).

		Predicted Risk: Low	Predicted Risk: High	
Real Risk	**Low**	n_{11}	n_{12}	N_{1+}
	High	n_{21}	n_{22}	N_{2+}
		N_{+1}	N_{+2}	N

A summary of all of the above measures is provided in Table 1 with reference to the notation in the above confusion matrix.

Measures of Binary Prediction Accuracy

Sensitivity and Specificity

The *sensitivity* of a classifier is defined as:

$$s = \frac{n_{22}}{n_{21} + n_{22}}$$

This is the proportion of high risk components that have been correctly classified as high risk components. The *specificity* of a classifier is defined as:

$$f = \frac{n_{11}}{n_{11} + n_{12}}$$

This is the proportion of low risk components that have been correctly classified as low risk components.

Proportion Correct

Proportion correct is defined as:

$$A = \frac{n_{11} + n_{22}}{N}$$

Type I and Type II Misclassifications

The Type I misclassification rate is $1 - f$, and the Type II misclassification rate is $1 - s$.

True Positive Rate

This is defined as:

$$TPA = \frac{n_{22}}{N_{+2}}$$

Kappa

Kappa is commonly used as a measure of inter-rater agreement. It is defined as:

$$\kappa = \frac{\frac{n_{11} + n_{22}}{N} - \sum_{i=1}^{2} \frac{N_{+i} N_{i+}}{N^2}}{1 - \sum_{i=1}^{2} \frac{N_{+i} N_{i+}}{N^2}}$$

The J Coefficient

This is defined as:

$$J = s + f - 1$$

Table 1: Measures of binary prediction accuracy.

El Emam et al. [35] recommend using Youdon's J coefficient [124] since it is independent of the proportion of high risk components in a particular data set and therefore is useful for producing generalizeable conclusions. Furthermore, it is a single number which can provide unambiguous results on the relative prediction accuracy of a model.

The J coefficient can vary from minus one to plus one, with plus one being perfect accuracy and –1 being the worst accuracy. A guessing classifier (i.e., one that guesses High/Low risk with a probability of

0.5) would have a J value of 0. Therefore, J values greater than zero indicate that the classifier is performing better than would be expected from a guessing classifier (i.e., by chance). Youdon provides estimates of the standard error of J, which can be used to construct confidence intervals.

4.10.2 Binary Dependent Variable and Continuous Prediction

This situation occurs most commonly when using a logistic regression model, which predicts the probability of a high-risk component (i.e., a value between 0 and 1), but the actual dependent variable is binary: high-risk/low-risk. In order to compare the results of the logistic regression prediction to the actual binary values, one needs to convert the continuous predicted probability to a binary value. This can be done by selecting a cutoff value on the predicted probability.

Previous studies have used a plethora of cutoff values to decide what is high risk or low risk, for example, 0.5 [4][15][17][92], 0.6 [15], 0.65 [15][18], 0.66 [14], 0.7 [18], and 0.75 [18]. In fact, and as noted by some authors [92], the choice of cutoff value is arbitrary, and one can obtain different results by selecting different cutoff values.

A general solution to the arbitrary thresholds problem mentioned above is Receiver Operating Characteristic (ROC) curves [89]. One selects many cutoff points, from 0 to 1 in our case, and calculates the sensitivity and specificity for each cutoff value, and plots sensitivity against 1-specificity as shown in Figure 10. Such a curve describes the compromises that can be made between sensitivity and specificity as the cutoff value is changed. The main advantages of expressing the accuracy of a prediction model (or for that matter any diagnostic test) as an ROC curve are that it is independent of prevalence (proportion of high risk classes), and therefore the conclusions drawn are general, and it is independent of the cutoff value, and therefore no arbitrary decisions need be made as to where to cut off the predicted probability to decide that a class is high risk [125]. Furthermore, using an ROC curve, one can easily determine the optimal operating point, and hence obtain an optimal cutoff value for an LR model.

We can obtain a summary accuracy measure of a prediction logistic regression model from an ROC curve by calculating the area under the curve using a trapezoidal rule [47]. The area under the ROC curve has an intuitive interpretation [47][111]: it is the estimated probability that a randomly selected component that is high-risk will be assigned a higher predicted probability by the logistic regression model than another randomly selected component that is low-risk. Therefore, an area under the curve of say 0.8 means that a randomly selected high-risk component has an estimated probability larger than a randomly selected low-risk component 80% of the time.

When a model cannot distinguish between high and low risk components, the area will be equal to 0.5 (the ROC curve will coincide with the diagonal). When there is a perfect separation of the values of the two groups, the area under the ROC curve equals 1 (the ROC curve will reach the upper left corner of the plot).

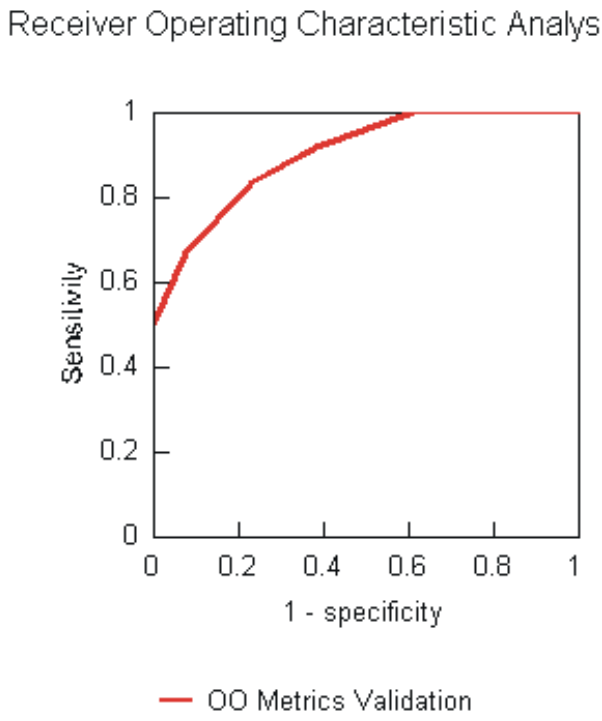

Figure 10: Hypothetical example of an ROC curve.

It is also possible to identify the optimal cutoff point on the ROC curve, which coincides with the leftmost top value on the ROC curve, and then use the J coefficient. The optimal cutoff point will allow the predicted probabilities to be dichotomized into a high and low risk prediction. This makes its appropriate then to use the J coefficient.

4.10.3 Continuous Dependent Variable and Continuous Prediction

This situation occurs if OLS or a count regression technique are used to build the models (or a machine learning algorithm such as regression trees) with, for example, development or maintenance effort as the dependent variable. The most common way of evaluating such predictions is to use a measure of relative error. Relative error for a single observation is defined as:

$$RE = \frac{\hat{y} - y}{y} \qquad \textbf{Eqn. 9}$$

where $\hat{y}$ is the predicted value and y is the actual value. The RE will be negative if the model underestimates, zero if the prediction is perfectly accurate, and positive if the model overestimates. If you multiply the RE by 100 then you would get a percentage deviation from the actual value. For example, an RE of 20% means that the model overestimates the actual value by 20%. A variation of this measure is the absolute relative error:

$$ARE = \left| \frac{\hat{y} - y}{y} \right| \qquad \textbf{Eqn. 10}$$

The ARE does not make a distinction between over and underestimation, and is perhaps appropriate when one the costs of over and underestimation are equivalent.

To aggregate the RE or ARE across multiple observations, one can take the mean or the median. Therefore, the MdARE is the median of the ARE values across many predictions and the MARE is the mean. Caution should be exercised though since aggregating RE and ARE may give different results, with the aggregation of RE being smaller in magnitude. The reason is that over and underestimation tend to cancel each other out.

4.11 Making System Level Predictions

Using product metrics, it is also desirable to make predictions at the system level. For example, if the dependent variable is whether a component is fault or not faulty, then a system level prediction would be the proportion of components that are faulty for the whole system. Similarly, if the dependent variable is the number of faults or development effort, then a system level prediction is the number of faults in the whole system or the total system development effort.

For the faulty/not-faulty case, a relatively straight forward equation provides us with an estimate of the proportion of components that have a fault. A naïve estimate of the proportion of faulty classes is:

$$\hat{t} = \frac{N_{+2}}{N} \qquad \textbf{Eqn. 11}$$

However, as shown in [101], this will only be unbiased if both sensitivity and specificity are equal to 1. The corrected estimate of the proportion of faulty components is given by:

$$\hat{p} = \frac{\hat{t} + \hat{f} - 1}{\hat{s} + \hat{f} - 1} \qquad \textbf{Eqn. 12}$$

If both $\hat{s}$ and $\hat{f}$ are equal to 1, then $\hat{p} = \hat{t}$. Since in practice this is unlikely to be the case, one should use Eqn. 12 to make the estimate.

In the case where the dependent variable is continuous, then summing up the predicted values for each component would give the system-level predicted value. As noted in Section 4.4, this will be biased unless a count regression model is used.

4.12 Interpreting Results

It is not uncommon for analysts, or their sponsors, to overinterpret results of validation studies. If a metric or metrics set are found to be valid in one study, this does not necessarily mean that they will be demonstrated to be valid in all future studies. It is only when evidence has accumulated that a particular metric is valid across systems and across organizations can we draw general conclusions.

Not being able to validate a metric in a single validation study may be due to a number of reasons. Four common ones are discussed below.

4.12.1 Methodological Flaws

Two main factors, which we have termed here methodological flaws, may have contributed towards a lack of significance in the studied relationship(s). The analyst should rule these out before concluding that a measure is not valid.

The first factor is the design of the empirical study that generated the data for the validation. There may have been problems in the study itself which would explain the lack of relationship. For example, if an experimental design was employed, then the analyst should investigate the possibility that some confounding variables that have an impact on the results were not appropriately controlled. This would be a problem with the internal validity of the study. Also if, for instance, the study was conducted with university students and small programming tasks, then the analyst should consider whether the expected relationship(s) are most likely to exist only in an industrial environment with more experienced programmers and large scale programming tasks. If the study was observational rather than experimental, then the analyst should consider that some additional confounding variables should be included for statistical adjustment.

The second factor concerns the characteristics of the data that were collected. For example, if maintainability data on easily maintainable programs were collected, then it is likely that there would be little variation in the maintainability variable. Such a restriction in range would lead to smaller empirical relationships. Also, as another example, if there are extreme outliers in the data, depending on the method of analysis, these may have a substantial impact on the strength of the empirical relationship.

4.12.2 Small Sample Size

Given that our criterion for the empirical validation of a metric is statistical significance, then the sample size can have a substantial impact on an analyst's findings and conclusions. This is because of statistical power. The power of a statistical test is defined as the probability of correctly rejecting the null hypothesis. The larger the sample size, the greater the statistical power of a given test. This means that if the analyst increases his/her sample size, and assuming the magnitude of the relationship remains unchanged, then s/he has a greater probability of finding the relationship statistically significant. This also means that if no relationship is identified, one possible reason is that the statistical test was not powerful (or sensitive) enough to detect it.

If an analyst does not find a statistically significant relationship, s/he should at a minimum determine whether the statistical test used was powerful enough for the given sample size. If s/he finds that the test was not powerful enough, then s/he should consider collecting more data and hence increasing the sample size. Often, however, that is not feasible. Alternatively, the analyst should consider using a more powerful test.

4.12.3 Invalid Theory

The approach that we have presented in this paper for empirically validating metrics of internal product attributes makes three assumptions:

1. that the internal attribute A_1 is related to the external attribute A_2 (i.e., that we can take the existence of this relationship in the real world for granted)
2. that measure X_1 measures the internal attribute A_1
3. that measure X_2 measures the external attribute A_2

Therefore, if the above assumptions are satisfied and if the analyst finds a relationship between X_1 and X_2, then s/he has validated X_1. If assumption 1 is satisfied, then the analyst can be confident that the relationship is not spurious. The analyst can ensure that assumption 2 is met by theoretically validating X_1 as noted earlier. A similar procedure may be followed for ensuring that assumption 3 is met.

If the analyst does not find a relationship between X_1 and X_2, then s/he should consider questioning assumption 1. One possible reason for not finding a relationship between the measured variables is that the hypothesized relationship between the attributes A_1 and A_2 is incorrect.

4.12.4 Unreliable Measures

In some cases, measures of internal attributes are not fully automated and hence they involve a level of subjectivity. A good example of this is the Function Point measure of the functionality attribute. An important consideration for such measures is their reliability. Reliability is concerned with the extent to which a measure is repeatable and consistent. For example, whether two independent raters will produce the same Function Point count for the same system is a question of reliability.

Less than perfect reliability of measured variables reduces the magnitude of their relationship, and hence reduces the possibility of finding the relationship statistically significant. If an estimate of the reliability of a measure is available, then one can correct the magnitude of the relationship for attenuation due to unreliability.

If the expected relationship involves measures that are not perfectly reliable, then the analyst should consider the possibility that attenuation due to unreliability is contributing towards the lack of significance. A correction for attenuation is most useful in terms of validation when the reliability of the measure(s) is quite low.

4.13 Reporting the Results of a Validation Study

Below we provide some guidelines on reporting the results of validating software product metrics. These may constitute sections that would typically be seen in a validation report, although we do not wish to imply that these must be section headings; only that the information under each of the items below should be in a report. Furthermore, some of the information might be too detailed for some publication outlets, and thus they may be included in appendices or publicly available technical reports accompanying published validation studies.

The motivation for such guidelines is to ensure some consistency in reporting results. This will help readers of such reports see the most pertinent information quickly, it will give the readers confidence that major validation considerations have been addressed (and this will add credibility to the studies), and should encourage analysts performing validation studies to pay attention to all the validation considerations covered in this chapter.

The items to consider when reporting a validation study are as follows:

- Theoretical Justification for the Metric(s)
  A section in the report ought to describe the theoretical justification for the metric. Since most metrics are based on assumptions of cognitive complexity, it should be made clear how this metric is expected to contribute to cognitive complexity.
- Definition and Operationalization of Metrics
  A precise definition of the metrics that are being validated ought to be provided, and especially how it was operationalized in the particular study. Since the details of some metrics may be influenced by the programming language, how the subtleties of operationalizing the metrics to the language used ought to be specified. For example, if one is studying a Java system, whether inner classes were counted or not. If one is analyzing a system written in a language such as C, then it is important to specify whether macros were expanded before static analysis. It is also important to specify the unit of measurement for the metric, for example, for object-oriented systems whether it is at the class or method level.
- Theoretical Validations
  Previous theoretical validations of the metrics ought to be summarized so that the reader would know the properties that the metric satisfies. This may also help interpret the results that are obtained.
- Confounding Variables
  The counfounding variables that will be controlled for ought to be described. At least the size measure used should be mentioned here. If there are multiple size measures, then these should be described and the impact of using the different size measures should be included in the results and discussion sections of the report.
- Source of Data
  A description of the system(s) that are studied ought to provided. Details such as whether the system was developed by students or professionals ought to be made clear, the project team size, the application domain, the programming language, and whether the data set comes from aggregations across multiple systems. If the data is publicly available (e.g., from a previously published study) then a reference and a summary of the source would suffice.
- Unit of Observation for Product Metrics
  A metric may be defined at a particular unit of measurement, but then observed/measured on a different unit. An example of that is aggregating the cyclomatic complexity metric across all procedures in a file, where the file is the unit of observation. In such a case, the form of the aggregation must be made clear.

- Count of External Components
  In many systems external libraries are used. Some of these libraries are public domain, e.g., GUI libraries. It is important to describe how connections with components from external libraries were handled during the computation of the metrics.
- Measurement of Dependent Variable
  A precise definition of the dependent variable is necessary. For example, if it is faults, then whether these were pre-release faults, post-release faults, or both. If there is information on fault severity then these should also be reported. The manner in which the dependent variable was measured is also important. For example, for faults whether the organization maintained records of faults attributed to each component or whether this information was extracted by parsing descriptions of changes.
- Measurement of Confounding Variables
  The measurement of confounding variables should also be described. Even a simple size measure has to be specified clearly. For example, in object-oriented systems size can be measured by the number of methods, number of attributes, or simply lines of code. If the confounding variables are measured subjectively, for example, rating of programmer experience on a five-point scale, then the exact scale used must be reported and any evidence as to its reliability (e.g., if different people rate the same programmer on that scale, will they produce the same ratings).
- Descriptive Statistics for the Data Set
  Minimal descriptive statistics for a data set include: the total number of observations, mean, median, inter-quartile-range, standard deviation, minimum value, maximum value, and the number of obvservations that do not have a zero value. If the metrics are transformed, then the summaries should be on the original scale. Also, it is important to present descriptive statistics on the dependent variable. If it is binary than the proportion in each category would suffice.
- Model Specification
  The model specification must be made clear. For example, whether logarithmic transformations were performed, and whether interactions were considered. If any specific techniques were used to identify optimal transformations, then these should be described (or summarized with the appropriate references).
- Model Building Procedure
  The procedure for building the models must be specified. This should include all diagnostics that were performed to test for, for example, collinearity, outliers, influential observations, heteroskedasticity, normality. Also the technique used for hypothesis testing should be clear, for example, whether it was an asymptotic test such as a t-test or a bootstrap procedure. When reporting the results all measures of magnitude of association (e.g., change in odds ratio) and p-values should be reported in full, as well as measures of goodness-of-fit for the models constructed.
- Variable Selection Procedure
  The variable selection procedure that is used should be made clear to the reader.
- Evaluation of Prediction Model
  The techniques used for evaluating prediction accuracy should be specified. This includes the overall strategy, such as splitting a data set into a train and test set, train on one release and test on the subsequent one, cross-validation, or bootstrapping. The coefficients that are used to compute accuracy must also be described.
- System Level Prediction Procedure
  If system level predictions are performed, these should be described and in the results they should also be evaluated as to their accuracy.
- Interpretation of Results
  The results of the validation study must be interpreted, especially if the study failed to validate some metrics. Potential reasons have been described above and the analyst should attempt to identify the ones most relevant. This will help future analysts who attempt to validate the same metrics.

- Comparison to Previous Validation Results
  To promote an accumulation of knowledge, it is preferable if researchers also review previous validation studies with the same metrics and provide an up-to-date summary of the findings thus far. The findings would include which metrics were systematically validated, which ones showed equivocal results, and which could not be validated systematically.

5 Further Considerations

5.1 Other Confounding Variables

Earlier we focused on size as the major confounding variable. The motivation is that if one is collecting software product metrics, then it is also trivial to collect size metrics. However, there are other potential confounding effects that might influence the results.

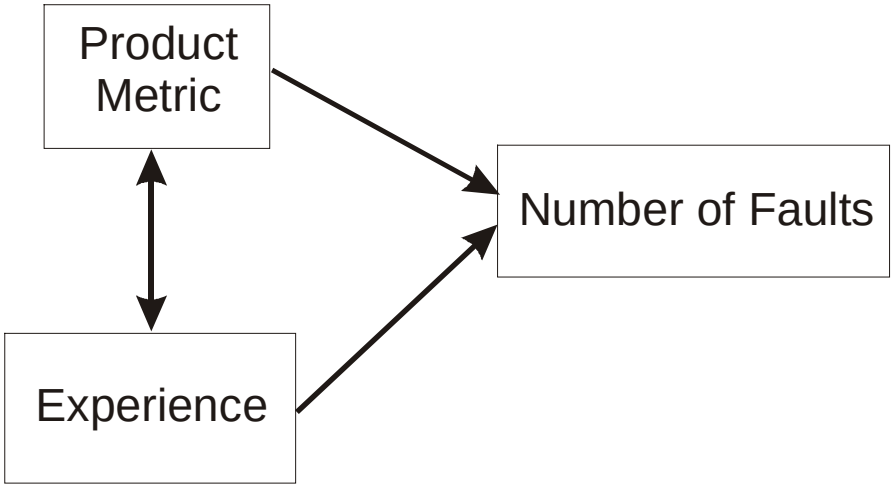

Figure 11: Example of the experience confounding variable on the number of faults.

Figure 11 shows an example of developer experience as another potential confounding variable. One would expect that the more experienced engineers will produce components that have less faults. Therefore the relationship between experience and the number of faults is expected to be negative. The association between experience and the product metrics can have either sign. In one instance an organization may assign the most complex components to its best engineers. This would mean that the association between experience and the product metric will be positive (we assume that higher values on the product metric mean greater complexity, e.g., coupling or lack of cohesion). In another instance one can argue that the most experienced engineers will produce components that are less complex because they understand, and have witnessed, the benefits of good design and programming practices. Then, one would expect e negative association. In either case, experience could also distort the relationship between a product metric and the dependent variable during validation.

Based on the above discussion, one should attempt to capture an experience or capability variable during a validation study and control for that. This is not always easy to do. Furthermore, a recent study showed no additional benefit to controlling programmer capability after controlling for size [34]. However, this is a single study, and the same result has not been demonstrated elsewhere yet. Therefore, we would recommend attempting to control for experience/capability.

5.2 Level of Measurement and Data Analysis Techniques

Several books and papers on the topic of measurement theory are conveying the idea that scale types should be used to proscribe the use of "inappropriate" statistical techniques. For example, a table similar to the one shown in Table 2 is given in [39]. This table, for instance, proscribes the use of the Pearson product moment correlation for scale types that are either nominal or ordinal. Such proscriptions, of course, are not unique to software engineering. For instance, they were originally presented by the psychologist Stevens [113], serve as the basis of the classic text of Siegel on nonparametric statistics [110], and serve as an integral part of the decision tree developed by Andrews et al. [2] to guide researchers in the selection of the most appropriate statistics. Accordingly, if a researcher's metrics do not reach the interval level, it is advised that s/he use non-parametric statistics (i.e., tests which make less stringent assumptions).

Scale Type	Examples of Appropriate Statistics	Type of Appropriate Statistics
Nominal	Mode Frequency Contingency Coefficient	Nonparametric Statistics
Ordinal	Median Kendall's tau Spearman's rho	
Interval	Mean Pearson's correlation	Nonparametric and Parametric Statistics
Ratio	Geometric Mean Coefficient of Variation	

Table 2: Stipulations on the appropriate statistics for various scale types.

However, in order to select the most "appropriate" statistics, a researcher has to know the type of scale(s) that s/he is using. The problem is that, in software engineering, like in other scientific disciplines, often it is very difficult to *determine* the scale type of a metric. For example, what is the scale type of cyclomatic complexity? Can we assume that the distances on the cyclomatic complexity scale are preserved across all of the scale? This is difficult to say and the answer can only be based on intuition. Despite a few available techniques to help the researchers in particular situations (see [13]), the answer to those questions is hardly ever straightforward.

Therefore, there are many cases where researchers cannot demonstrate that their scales are interval, but they are confident that they are more than only ordinal. By treating them as ordinal, researchers would be discarding a good deal of information. Therefore, as Tukey [117] notes *"The question must be 'If a scale is not an interval scale, must it be merely ordinal?'"*

Is it realistic to answer questions about scale type with absolute certainty, since their answers always rely on intuition and are therefore subjective? Can we know for sure the scale types of the metrics we use? Knowing the scale type of a metric with absolute certainty is difficult in the vast majority of cases. And in those cases, should we just discard our practical questions—whose answers may have a real impact on the software process—because we are not 100% positive about the scale types of the metrics we are using? To paraphrase Tukey [118], "Science is not mathematics" and we are not looking for perfection and absolute proofs but for *evidence* that our theories match reality as closely as possible. The other alternative, i.e., reject approximate theories, would have catastrophic consequences on most sciences, and in particular, on software engineering. What is not acceptable from a strictly mathematical perspective may be acceptable evidence and even a necessary one from an engineering or an experimental perspective.

It is informative to note that much of the recent progress in the social sciences would not have been possible if the use of "approximate" measurement scales had been strictly proscribed. For example, Tukey [117] states after summarizing Stevens' proscriptions *"This view thus summarized is a dangerous one. If generally adopted it would not only lead to inefficient analysis of data, but it would also lead to*

failure to give any answer at all to questions whose answers are perfectly good, though slightly approximate. All this loss for essentially no gain." Similarly, in the context of multiple regression, Cohen and Cohen [23] state: *"The issue of the level of scaling and measurement precision required of quantitative variables in [Multiple Regression/Correlation] is complex and controversial. We take the position that, in practice, almost anything goes. Formally, fixed model regression analysis demands that the quantitative independent variables be scaled at truly equal intervals ... Meeting this demand would rule out the use of all psychological tests, sociological indices, rating scales, and interview responses ... this eliminates virtually all kinds of quantitative variables on which the behavioral sciences depend."* Even Stevens himself, with respect to ordinal scales, concedes that [113]: *"In the strictest propriety the ordinary statistics involving means and standard deviations ought not to be used with these scales, for these statistics imply a knowledge of something more than relative rank-order of data. On the other hand, for this 'illegal' statisticizing there can be invoked a kind of pragmatic sanction: In numerous instances it leads to fruitful results."*

We do not wish to give the impression of a carte blanche whereby any statistics with any scale type will produce meaningful results. There are obvious cases where cognizance of the scale type is critical. For example, one would not treat a nominally scaled metric, such as defect classification, as an intervaly scaled variable in regression. In most cases, the questions to answer (i.e., our measurement goals) determine the scale under which data must be used, and not *vice versa*. One should use the appropriate technique assuming the level of measurement required by the question. If a pattern is detected, then the analyst should start thinking about the validity of the assumption s/he made about the scale types. In addition, it is sometimes possible to use different statistics assuming different scale types and compare the results.

For example, if a statistically significant linear relationship is found between coupling and faults through regression then, theoretically, the researcher must start wondering if the computed level of significance is real (or close to reality) since there is some uncertainty with respect to the type of the coupling scale. External information may be examined in order to confirm or otherwise the scale assumption. For example, assuming we want to model fault counts, programmers may be surveyed by asking them to score the relative "difficulty" of programs with different coupling levels. If the scores confirm that the distance is, on average, preserved along the studied part of the scale (hopefully, the relevant one for the environment under study), then the equal interval properties may be assumed with greater confidence. In addition, thorough experience and a good intuitive understanding of the phenomenon under study can help a great deal. For example, in a given environment, very often programmers know the common causes of faults and their relative impact. Scales may thus be validated with the help of experts.

Such an approach is supported by numerous studies (for a more detailed discussion, see [13]) which show that, in general, parametric statistics are robust when scales are not too far from being interval. In other words, when a scale is not an exponential distortion of an interval scale, the likelihood of error of type I (i.e., the null hypothesis is incorrectly rejected) does not significantly increase. In addition, other studies have shown that, in most cases, non-parametric statisics were of lesser power (i.e., higher likelihood of error of type II, i.e., the null hypothesis is falsely not rejected) than parametric statistics when their underlying assumptions were not violated to an extreme extent. Moreover, dealing with variable interactions in a multivariate tends to be much easier when using parametric techniques, e.g., multivariate regression analysis with interaction terms.

5.3 Independence of Analyst and Replication

Ideally, whenever an analyst proposes a new metric s/he should validate it. It is almost always the case that such validations are "successful", at least as presented. Caution should be exercised in overinterpreting a metric that has been shown to be valid by its developer. There are three reasons. First, the developer of metric may not even attempt to publish the results unless the validation was successful. Therefore, it is plausible that we only see the successfully validated metrics. Second, the developers may have tried many possible variants of the proposed metric(s) until one variant was found that can be successfully validated on a particular system. Since the metric was customized for a particular data set, it may not work as well on another system. Finally, with sufficient time and effort an analyst can perform an analysis that validates a metric (e.g., by removing observations).

On the other hand, if the developer of a metric cannot validate his/her own metric, then one must seriously question whether that metric is valid at all in other contexts. Therefore, self-validation is an important first step. It does demonstrate that under certain circumstances a metric can be associated with the dependent variable. However, it is important that validation studies are replicated and confirmed independently. It is only through such independent studies that one can start to build a convincing case that a particular metric is actually valid.

6 Conclusions

In this chapter we have presented a complete methodology for validating software product metrics. Admittedly this methodology is biased towards the use of statistical techniques, however, it is with statistical techniques that analysts have the most difficulty, and many of the issues discussed are equally applicable to the use of machine learning modeling techniques.

While it is not claimed that the methodology presented here is the last word on the validation of software product metrics, it does represent best current practices in software engineering. It is possible, and indeed encouraged, that future analysts improve on this methodology. Software product metrics play a central role in software engineering, and their proper validation will ensure that there is a compelling case for their use in practice.

7 Acknowledgements

I would like to thank Shadia El Gazzar and Mazen Fahmi for their comments on an earlier version of this chapter.

8 Appendix A: Evaluating Code Churn

In this appendix we present the results of a small study to determine whether code churn is a reasonable measure of overall maintenance effort. We would expect that code churn should have at least a monotonically increasing relationship with maintenance effort if it is to be used as a surrogate measure.[9] We test this through a data set collected from a 40 KSLOC systems application written in C that had a peak staff load of 5 persons. Our focus is on corrective maintenance only, since this has been the focus of previous maintenance effort prediction studies as well [55]. The development process for this project included rigorous inspections. The data we collected was from maintenance of the system after its first release. Both code churn and effort data collected through a measurement program were measured.

When a fault is discovered it is reported to the maintainers who issue a Change Report Form (CRF). The CRF is assigned to an engineer who is responsible for bringing it to closure. This involves recreating the problem, isolating the cause, and performing the necessary changes to fix the fault. The CRF requires a description of the resolution and reporting of the total amount of effort spent on fixing the fault in minutes, including isolation effort. When checking in each module into the configuration management system, it is necessary to specify the CRF number that caused the module to be checked out. Therefore, it is possible to have a complete mapping between all changes made to all of the source code and the CRFs. The effort data for each CRF were obtained from the forms.

The development environment was constructed such that each individual function was in a separate file. Therefore, it was easy to localize changes to specific functions.

During the maintenance of this system there were also requirements changes. Therefore, some functions were modified due to new requirements. After each such change a suite of regression tests were

[9] A stricter requirement would be that there is a linear relationship. However, if there is no monotonic relationship then there is no linear relationship either.

performed. Sometimes faults were discovered during regression testing. These faults were also included in our data set.

The code churn was computed from the configuration management system. It was possible to extract the before and after version of each function that was changed due to a CRF. In total, we had data on 98 error CRFs. These came from maintenance over a period of 18 months.

Given that our hypothesis is that of a monotonic relationship, we first determine the magnitude of the relationship between change effort and code churn using the Spearman rank order correlation coefficient [109]. We also test whether the correlation is different from zero at an alpha level of 0.05. Only one-sided tests were considered since our hypothesis is directional.

For determining the p-value for the Spearman correlation, we computed the exact permutation distribution, and used that as the basis for determining statistical significance. The permutation distribution does not require asymptotic assumptions.

Summary statistics for the code churn and effort data are presented in Figure 12. There are a number of noticeable outliers, and the distributions are clearly not symmetric. Hence a non-parametric approach for evaluating the relationships is certainly justifiable.

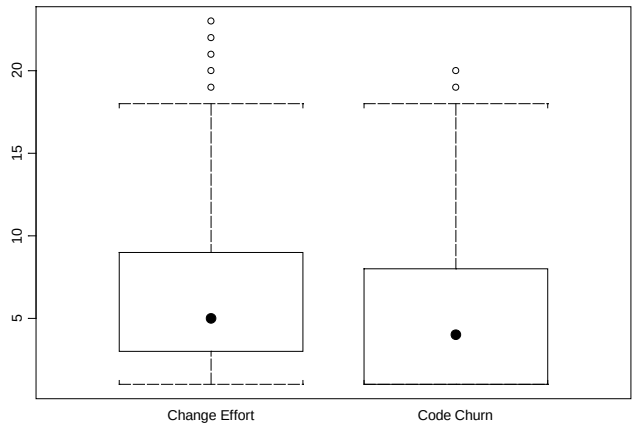

Figure 12: Distribution of the code churn and actual effort variables.

The Spearman correlation is 0.3638 ($p<0.0001$), which is statistically significant. A scatterplot illustrating this relationship is depicted in Figure 12. No easily identifiable pattern can be seen. The correlation is rather small, and does not bide well for using code churn as a surrogate measure of maintenance effort, and would recommend *against* using it in future studies.

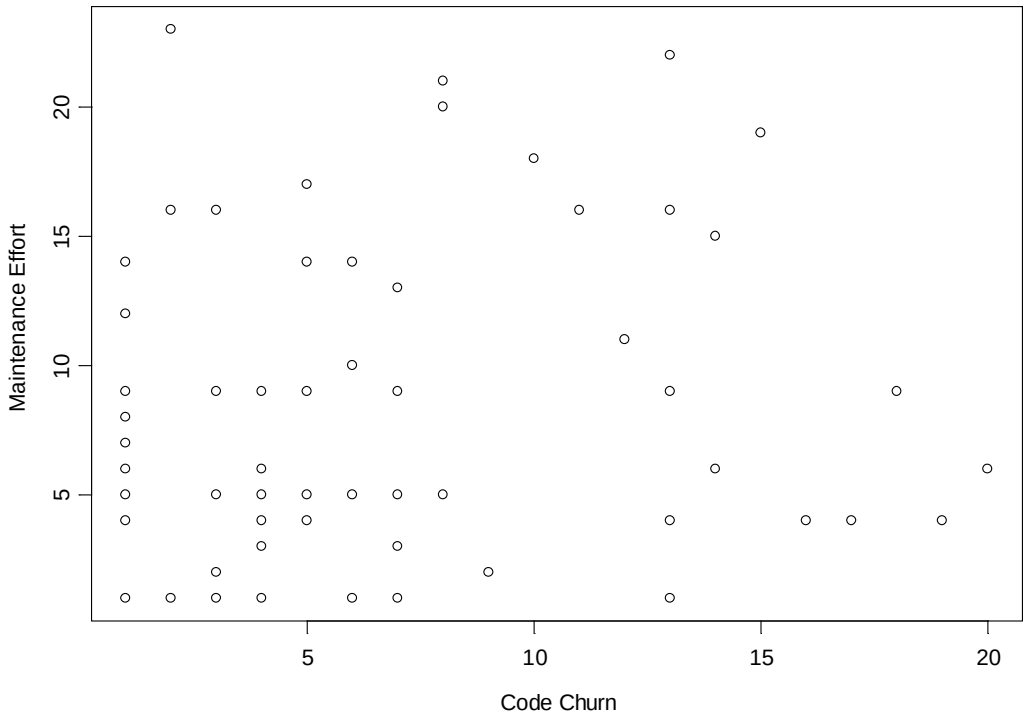

Figure 13: Scatterplot of code churn vs. maintenance effort.

9 References

[1] M. Almeida, H. Lounis, and W. Melo, "An Investigation on the Use of Machine Learned Models for Estimating Correction Costs," in *Proceedings of the 20th International Conference on Software Engineering*, pp. 473-476, 1998.

[2] F. Andrews, L. Klem, T. Davidson, P. O'Malley, and W. Rodgers, *A Guide for Selecting Statistical Techniques for Analyzing Social Science Data*, Institute for Social Research, University of Michigan, 1981.

[3] A. Baker, J. Bieman, N. Fenton, D. Gustafson, A. Mellon, and R. Whitty, "A Philosophy for Software Measurement," *Journal of Systems and Software*, vol. 12, pp. 277-281, 1990.

[4] V. Basili, L. Briand, and W. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators," *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 751-761, 1996.

[5] V. Basili, S. Condon, K. El-Emam, R. Hendrick, and W. Melo, "Characterizing and Modeling the Cost of Rework in a Library of Reusable Software Components," in *Proceedings of the 19th International Conference on Software Engineering*, pp. 282-291, 1997.

[6] D. Belsley, E. Kuh, and R. Welsch, *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity*, John Wiley and Sons, 1980.

[7] D. Belsley, *Conditioning Diagnostics: Collinearity and Weak Data in Regression*, John Wiley and Sons, 1991.

[8] A. Binkley and S. Schach, "Validation of the Coupling Dependency Metric as a Predictor of Run-Time Failures and Maintenance Measures," in *Proceedings of the 20th International Conference on Software Engineering*, pp. 452-455, 1998.

[9] S. Biyani and P. Santhanam, "Exploring Defect Data from Development and Customer Usage of Software Modules Over Multiple Releases," in *Proceedings of the International Syposium on Software Reliability Engineering*, pp. 316-320, 1998.

[10] L. Breiman and J. Friedman, "Estimating Optimal Transformations for Multiple Regression and Correlation," *Journal of the American Statistical Association*, vol. 80, pp. 580-597, 1985.

[11] L. Briand, V. Basili, and C. Hetmanski, "Developing Interpretable Models with Optimized Set Reduction for Identifying High-Risk Software Components," *IEEE Transactions on Software Engineering*, vol. 19, no. 11, pp. 1028-1044, 1993.

[12] L. Briand, W. Thomas, and C. Hetmanski, "Modeling and Managing Risk Early in Software Development," in *Proceedings of the International Conference on Software Engineering*, pp. 55-65, 1993.

[13] L. Briand, K. El-Emam, and S. Morasca, "On the Application of Measurement Theory to Software Engineering," *Empirical Software Engineering: An International Journal*, vol. 1, no. 1, pp. 61-88, 1996.

[14] L. Briand, J. Daly, V. Porter, and J. Wuest, "Predicting Fault-Prone Classes with Design Measures in Object Oriented Systems," in *Proceedings of the International Symposium on Software Reliability Engineering*, pp. 334-343, 1998.

[15] L. Briand, J. Wuest, S. Ikonomovski, and H. Lounis, "A Comprehensive Investigation of Quality Factors in Object-Oriented Designs: An Industrial Case Study," International Software Engineering Research Network, ISERN-98-29, 1998. (available at http://www.iese.fhg.de/network/ISERN/pub/isern_biblio_tech.html)

[16] L. Briand and J. Wuest, "The Impact of Design Properties on Development Cost in Object-Oriented Systems," International Software Engineering Research Network, ISERN-99-16, 1999.

[17] L. Briand, J. Wuest, S. Ikonomovski, and H. Lounis, "Investigating Quality Factors in Object-Oriented Designs: An Industrial Case Study," in *Proceedings of the International Conference on Software Engineering*, 1999.

[18] L. Briand, J. Wuest, J. Daly, and V. Porter, "Exploring the Relationships Between Design Measures and Software Quality in Object Oriented Systems," *Journal of Systems and Software*, vol. 51, pp. 245-273, 2000.

[19] A. Cameron and P. Trivedi, "Econometric Models Based On Count Data: Comparisons and Applications of Some Estimators and Tests," *Journal of Applied Economics*, vol. 1, pp. 29-53, 1986.

[20] M. Cartwright and M. Shepperd, "An Empirical Investigation of an Object-Oriented Software System," *IEEE Transactions on Software Engineering (to appear)*, 2000.

[21] S. Chidamber, D. Darcy, and C. Kemerer, "Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis," *IEEE Transactions on Software Engineering*, vol. 24, no. 8, pp. 629-639, 1998.

[22] F. Coallier, J. Mayrand, and B. Lague, "Risk Management in Software Product Procurement," in *Elements of Software Process Assessment and Improvement*, K. El-Emam and N. H. Madhavji, Eds., IEEE CS Press, 1999.

[23] J. Cohen and P. Cohen, *Applied Multiple Regression / Correlation Analysis for the Behavioral Sciences*, Lawrence Erlbaum Associates, 1983.

[24] R. Cook, "Detection of Influential Observations in Linear Regression," *Technometrics*, vol. 19, pp. 15-18, 1977.

[25] R. Cook, "Influential Observations in Linear Regression," *Journal of the American Statistical Association*, vol. 74, pp. 169-174, 1979.

[26] S. Derksen and H. Keselman, "Backward, Forward and Stepwise Automated Subset Selection Algorithms: Frequency of Obtaining Authentic and Noise Variables," *British Journal of Mathematical and Statistical Psychology*, vol. 45, pp. 265-282, 1992.

[27] C. Ebert and T. Liedtke, "An Integrated Approach for Criticality Prediction," in *Proceedings of the 6th International Symposium on Software Reliability Engineering*, pp. 14-23, 1995.

[28] B. Efron and R. Tibshirani, *An Introduction to the Bootstrap*, Chapman & Hall, 1993.

[29] K. El-Emam, "The Predictive Validity Criterion for Evaluating Binary Classifiers," in *Proceedings of the 5th International Symposium on Software Metrics*, pp. 235-244, 1998.

[30] K. El-Emam, S. Benlarbi, N. Goel, and S. Rai, "A Validation of Object-Oriented Metrics," National Research Council of Canada, NRC/ERB 1063, 1999.

[31] K. El-Emam and W. Melo, "The Prediction of Faulty Classes Using Object-Oriented Design Metrics," National Research Council of Canada, NRC/ERB 1064, 1999.
[32] K. El-Emam, "Object-Oriented Metrics: A Review of Theory and Practice," in *Advances in Software Engineering: Topics in Comprehension, Evolution, and Evaluation (to appear)*, O. Tanir and H. Erdogmus, Eds., Spinger-Verlag, 2000.
[33] K. El-Emam, S. Benlarbi, N. Goel, W. Melo, H. Lounis, and S. Rai, "The Optimal Class Size for Object-Oriented Software: A Replicated Study," National Research Council of Canada, NRC/ERB 1074, 2000.
[34] K. El-Emam, S. Benlarbi, N. Goel, and S. Rai, "The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics," *IEEE Transactions on Software Engineering (to appear)*, 2000.
[35] K. El-Emam, S. Benlarbi, N. Goel, and S. Rai, "Comparing Case-Based Reasoning Classifiers for Predicting High Risk Software Components," *Journal of Systems and Software (to appear)*, 2001.
[36] W. Evanco, "Poisson Analyses of Defects for Small Software Components," *Journal of Systems and Software*, vol. 38, pp. 27-35, 1997.
[37] N. Fenton, "Software Metrics: Theory, Tools and Validation," *Software Engineering Journal*, pp. 65-78, January, 1990.
[38] N. Fenton and B. Kitchenham, "Validating Software Measures," *Journal of Software Testing, Verification and Reliability*, vol. 1, no. 2, pp. 27-42, 1990.
[39] N. Fenton, *Software Metrics: A Rigorous Approach*, Chapman & Hall, 1991.
[40] N. Fenton and M. Neil, "A Critique of Software Defect Prediction Models," *IEEE Transactions on Software Engineering*, vol. 25, no. 5, pp. 676-689, 1999.
[41] N. Fenton and N. Ohlsson, "Quantitative Analysis of Faults and Failures in a Complex Software System," *IEEE Transactions on Software Engineering (to appear)*, 2000.
[42] V. French, "Establishing Software Metrics Thresholds," in *Proceedings of the 9th International Workshop on Software Measurement*, 1999.
[43] T. Furuyama, Y. Arai, and K. Iio, "Fault Generation Model and Mental Stress Effect Analysis," *Journal of Systems and Software*, vol. 26, pp. 31-42, 1994.
[44] T. Furuyama, Y. Arai, and K. Iio, "Analysis of Fault Generation Caused by Stress During Software Development," *Journal of Systems and Software*, vol. 38, pp. 13-25, 1997.
[45] W. Gardner, E. Mulvey, and E. Shaw, "Regression analyses of counts and rates: Poisson, overdispersed Poisson, and negative binomial models," *Psychological Bulletin*, vol. 118, pp. 392-404, 1995.
[46] D. Glasberg, K. El-Emam, W. Melo, J. Machado, and N. Madhavji, "Evaluating Thresholds for Object-Oriented Design Measures," *(submitted for publication)*, 2000.
[47] J. Hanley and B. McNeil, "The Meaning and Use of the Area Under a Receiver Operating Characteristic Curve," *Diagnostic Radiology*, vol. 143, no. 1, pp. 29-36, 1982.
[48] F. Harrell and K. Lee, "Regression Modelling Strategies for Improved Prognostic Prediction," *Statistics in Medicine*, vol. 3, pp. 143-152, 1984.
[49] F. Harrell, K. Lee, and D. Mark, "Multivariate Prognostic Models: Issues in Developing Models, Evaluating Assumptions and Adequacy, and Measuring and Reducing Errors," *Statistics in Medicine*, vol. 15, pp. 361-387, 1996.
[50] R. Harrison, L. Samaraweera, M. Dobie, and P. Lewis, "An Evaluation of Code Metrics for Object-Oriented Programs," *Information and Software Technology*, vol. 38, pp. 443-450, 1996.
[51] R. Harrison, S. Counsell, and R. Nithi, "Coupling Metrics for Object Oriented Design," in *Proceedings of the 5th International Symposium on Software Metrics*, pp. 150-157, 1998.
[52] W. Harrison, "Using Software Metrics to Allocate Testing Resources," *Journal of Management Information Systems*, vol. 4, no. 4, pp. 93-105, 1988.
[53] L. Hatton, "Does OO Sync with How We Think ?," *IEEE Software*, pp. 46-54, May/June, 1998.
[54] B. Henderson-Sellers, *Object-Oriented Metrics: Measures of Complexity*, Prentice-Hall, 1996.
[55] S. Henry and S. Wake, "Predicting Maintainability with Software Quality Metrics," *Software Maintenance: Research and Practice*, vol. 3, pp. 129-143, 1991.
[56] E. Hilgard, R. Atkinson, and R. Atkinson, *Introduction to Psychology*, Harcourt Brace Jovanovich, 1971.
[57] A. Hoerl and R. Kennard, "Ridge Regression: Biased Estimation for Nonorthogonal Problems," *Technometrics*, vol. 12, no. 1, pp. 55-67, 1970.

[58] A. Hoerl and R. Kennard, "Ridge Regression: Applications to Nonorthogonal Problems," *Technometrics*, vol. 12, no. 1, pp. 69-82, 1970.
[59] D. Hosmer and S. Lemeshow, *Applied Logistic Regression*, John Wiley & Sons, 1989.
[60] J. Hudepohl, S. Aud, T. Khoshgoftaar, E. Allen, and J. Mayrand, "EMERALD: Software Metrics and Models on the Desktop," *IEEE Software*, vol. 13, no. 5, pp. 56-60, 1996.
[61] J. Hudepohl, S. Aud, T. Khoshgoftaar, E. Allen, and J. Mayrand, "Integrating Metrics and Models for Software Risk Assessment," in *Proceedings of the 7th International Symposium on Software Reliability Engineering*, pp. 93-98, 1996.
[62] IEEE, "IEEE Standard Glossary of Software Engineering Terminology," IEEE Computer Society, IEEE Std 610.12-1990 (Revision and redesignation of IEEE Std 729-1983), 1990.
[63] ISO/IEC, "ISO/IEC 9126: Information Technology - Software Product Evaluation - Quality Characteristics and Guidelines for their Use," International Organization for Standardization and the International Electrotechnical Commission, 1991.
[64] ISO/IEC, "Information Technology - Software Product Evaluation; Part 1: Overview," International Organization for Standardization and the International Electrotechnical Commission, ISO/IEC DIS 14598-1, 1996.
[65] R. Johnson and D. Wichern, *Applied Multivariate Statistical Analysis*, Prentice Hall, 1998.
[66] M. Jorgensen, "Experience with the Accuracy of Software Maintenance Task Effort Prediction Models," *IEEE Transactions on Software Engineering*, vol. 21, no. 8, pp. 674-681, 1995.
[67] M. Kaaniche and K. Kanoun, "Reliability of a Commercial Telecommunications System," in *Proceedings of the International Symposium on Software Reliability Engineering*, pp. 207-212, 1996.
[68] J. Kearney, R. Sedlmeyer, W. Thompson, M. Gray, and M. Adler, "Software Complexity Measurement," *Communications of the ACM*, vol. 29, no. 11, pp. 1044-1050, 1986.
[69] T. Khoshgoftaar and D. Lanning, "Are the Principal Components of Software Complexity Data Stable Across Software Products ?," in *Proceedings of the International Symposium on Software Metrics*, pp. 61-72, 1994.
[70] T. Khoshgoftaar, E. Allen, K. Kalaichelvan, N. Goel, J. Hudepohl, and J. Mayrand, "Detection of Fault-Prone Program Modules in a Very Large Telecommunications System," in *Proceedings of the 6th International Symposium on Software Reliability Engineering*, pp. 24-33, 1995.
[71] T. Khoshgoftaar, E. Allen, L. Bullard, R. Halstead, and G. Trio, "A Tree Based Classification Model for Analysis of a Military Software System," in *Proceedings of the IEEE High-Assurance Systems Engineering Workshop*, pp. 244-251, 1996.
[72] T. Khoshgoftaar, E. Allen, K. Kalaichelvan, and N. Goel, "The Impact of Software Evolution and Reuse on Software Quality," *Empirical Software Engineering: An International Journal*, vol. 1, pp. 31-44, 1996.
[73] T. Khoshgoftaar, E. Allen, K. Kalaichelvan, and N. Goel, "Early Quality Prediction: A Case Study in Telecommunications," *IEEE Software*, pp. 65-71, January, 1996.
[74] T. Khoshgoftaar, K. Ganesan, E. Allen, F. Ross, R. Munikoti, N. Goel, and A. Nandi, "Predicting Fault-Prone Modules with Case-Based Reasoning," in *Proceedings of the Eighth International Symposium on Software Reliability Engineering*, pp. 27-35, 1997.
[75] T. Khoshgoftaar, E. Allen, W. Jones, and J. Hudepohl, "Classification Tree Models of Software Quality Over Multiple Releases," in *Proceedings of the International Symposium on Software Reliability Engineering*, pp. 116-125, 1999.
[76] J. Kim and C. Mueller, *Factor Analysis: Statistical Methods and Practical Issues*, Sage Publications, 1978.
[77] G. King, "Statistical Models for Political Science Event Counts: Bias in Conventional Procedures and Evidence for the Exponential Poisson Regression Model," *American Journal of Political Science*, vol. 32, pp. 838-862, 1988.
[78] G. King, "Variance Specification in Event Count Models: From Restrictive Assumptions to a Generalized Estimator," *American Journal of Political Science*, vol. 33, no. 3, pp. 762-784, 1989.
[79] B. Kitchenham and S. Linkman, "Design Metrics in Practice," *Information and Software Technology*, vol. 32, no. 4, pp. 304-310, 1990.
[80] B. Kitchenham, S-L Pfleeger, and N. Fenton, "Towards a Framework for Software Measurement Validation," *IEEE Transactions on Software Engineering*, vol. 21, no. 12, pp. 929-944, 1995.

[81] J. Landwehr, D. Pergibon, and A. Shoemaker, "Graphical Methods for Assessing Logistic Regression Models," *Journal of the American Statistical Association*, vol. 79, no. 385, pp. 61-71, 1984.
[82] F. Lanubile and G. Visaggio, "Evaluating Predictive Quality Models Derived from Software Measures: Lessons Learned," *Journal of Systems and Software*, vol. 38, pp. 225-234, 1997.
[83] J. Lewis and S. Henry, "A Methodology for Integrating Maintainability Using Software Metrics," in *Proceedings of the International Conference on Software Maintenance*, pp. 32-39, 1989.
[84] W. Li and S. Henry, "Object-Oriented Metrics that Predict Maintainability," *Journal of Systems and Software*, vol. 23, pp. 111-122, 1993.
[85] W. Li, S. Henry, D. Kafura, and R. Schulman, "Measuring Object-Oriented Design," *Journal of Object-Oriented Programming*, pp. 48-55, July/August, 1995.
[86] M. Lorenz and J. Kidd, *Object-Oriented Software Metrics*, Prentice-Hall, 1994.
[87] M. Lyu, J. Yu, E. Keramides, and S. Dalal, "ARMOR: Analyzer for Reducing Module Operational Risk," in *Proceedings of the 25th International Symposium on Fault-Tolerant Computing*, pp. 137-142, 1995.
[88] S. Menard, "Coefficients of Determination for Multiple Logistic Regression Analysis," *The American Statistician*, vol. 54, no. 1, pp. 17-24, 2000.
[89] C. Metz, "Basic Principles of ROC Analysis," *Seminars in Nuclear Medicine*, vol. VIII, no. 4, pp. 283-298, 1978.
[90] G. Miller, "The Magical Number 7 Plus or Minus Two: Some Limits on Our Capacity for Processing Information," *Psychological Review*, vol. 63, pp. 81-97, 1957.
[91] K-H Moller and D. Paulish, "An Empirical Investigation of Software Fault Distribution," in *Proceedings of the First International Software Metrics Symposium*, pp. 82-90, 1993.
[92] S. Morasca and G. Ruhe, "Knowledge Discovery from Software Engineering Measurement Data: A Comparative Study of Two Analysis Techniques," in *Proceedings of the International Conference on Software Engineering and Knowledge Engineering*, 1997.
[93] J. Munson and T. Khoshgoftaar, "The Dimensionality of Program Complexity," in *Proceedings of the 11th International Conference on Software Engineering*, pp. 245-253, 1989.
[94] J. Munson and T. Khoshgoftaar, "The Detection of Fault-Prone Programs," *IEEE Transactions on Software Engineering*, vol. 18, no. 5, pp. 423-433, 1992.
[95] N. Ohlsson and H. Alberg, "Predicting Fault-Prone Software Modules in Telephone Switches," *IEEE Transactions on Software Engineering*, vol. 22, no. 12, pp. 886-894, 1996.
[96] D. Parnas, "On the Criteria to be Used in Decomposing Systems into Modules," *Communications of the ACM*, vol. 14, no. 1, pp. 221-227, 1972.
[97] D. Pergibon, "Logistic Regression Diagnostics," *The Annals of Statistics*, vol. 9, no. 4, pp. 705-724, 1981.
[98] A. Porter and R. Selby, "Evaluating Techniques for Generating Metric-Based Classification Trees," *Journal of Systems and Software*, vol. 12, pp. 209-218, 1990.
[99] A. Porter, "Using Measurement-Driven Modeling to Provide Empirical Feedback to Software Developers," *Journal of Systems and Software*, vol. 20, pp. 237-243, 1993.
[100] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, 1990.
[101] W. Rogan and B. Gladen, "Estimating Prevalence from the Results of a Screening Test," *American Journal of Epidemiology*, vol. 107, no. 1, pp. 71-76, 1978.
[102] L. Rosenberg, R. Stapko, and A. Gallo, "Object-Oriented Metrics for Reliability," presented at *IEEE International Symposium on Software Metrics*, 1999.
[103] [P. Rousseeuw and A. Leroy, *Robust Regression and Outlier Detection*, John Wiley & Sons, 1987.
[104] R. Schaefer, L. Roi, and R. Wolfe, "A Ridge Logistic Estimator," *Communications in Statistics - Theory and Methods*, vol. 13, no. 1, pp. 99-113, 1984.
[105] R. Schaefer, "Alternative Estimators in Logistic Regression when the Data are Collinear," *Journal of Statistical Computation and Simulation*, vol. 25, pp. 75-91, 1986.
[106] N. Schneidewind, "Methodology for Validating Software Metrics," *IEEE Transactions on Software Engineering*, vol. 18, no. 5, pp. 410-422, 1992.
[107] N. Schneidewind, "Validating Metrics for Ensuring Space Shuttle Flight Software Quality," *IEEE Computer*, pp. 50-57, August, 1994.

[108] N. Schneidewind, "Software Metrics Model for Integrating Quality Control and Prediction," in *Proceedings of the 8th International Symposium on Software Reliability Engineering*, pp. 402-415, 1997.
[109] D. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, CRC Press, 1997.
[110] S. Siegel and J. Castellan, *Nonparametric Statistics for the Behavioral Sciences*, McGraw Hill, 1988.
[111] D. Spiegelhalter, "Probabilistic Prediction in Patient Management in Clinical Trials," *Statistics in Medicine*, vol. 5, pp. 421-433, 1986.
[112] Entropy and Jensen Inequality, <http://sepwww.stanford.edu/sep/prof/toc_html/pvi/jen/paper_html/>, Stanford Exploration Project, accessed: 20th April 2000.
[113] S. Stevens, "On the Theory of Scales of Measurement," *Science*, vol. 103, no. 2684, pp. 677-680, June, 1946.
[114] M. Sturman, "Multiple Approaches to Analyzing Count Data in Studies of Individual Differences: The Propensity for Type I Errors, Illustrated with the Case of Absenteeism Prediction," *Educational and Psychological Measurement*, vol. 59, no. 3, pp. 414-430, 1999.
[115] J. Szentes and J. Gras, "Some Practical Views of Software Complexity Metrics and a Universal Measurement Tool," in *Proceedings of the First Australian Software Engineering Conference*, pp. 83-88, 1986.
[116] M-H. Tang, M-H. Kao, and M-H. Chen, "An Empirical Study on Object Oriented Metrics," in *Proceedings of the Sixth International Software Metrics Symposium*, pp. 242-249, 1999.
[117] J. Tukey, "Data Analysis and Behavioral Science or Learning to Bear the Quantitative Man's Burden by Shunning Badmandments," in *The Collected Works of John W. Tukey - Vol. III*, Wadsworth, 1986.
[118] J. Tukey, "The Future of Data Analysis," in *The Collected Works of John W. Tukey - Vol. III*, Wadsworth, 1986.
[119] K. Ulm, "A Statistical Method for Assessing A Threshold in Epidemiological Studies," *Statistics in Medicine*, vol. 10, pp. 341-349, 1991.
[120] R. De Veaux, "Finding Transformations for Regression Using the ACE Algorithm," *Sociological Methods and Research*, vol. 18, no. 2/3, pp. 327-359, 1989.
[121] H. Weisberg, *Central Tendency and Variability*, Sage Publications, 1992.
[122] S. Weiss and C. Kulikowski, *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*, Morgan Kaufmann Publishers, 1991.
[123] R. Winkelmann, *Econometric Analysis of Count Data*, Springer-Verlag, 1997.
[124] W. Youden, "Index for Rating Diagnostic Tests," *Cancer*, vol. 3, pp. 32-35, 1950.
[125] M. Zweig and G. Campbell, "Receiver-Operating Characteristic (ROC) Plots: A Fundamental Evaluation Tool in Clinical Medicine," *Clinical Chemistry*, vol. 39, no. 4, pp. 561-577, 1993.