Software Defect Prediction: Review, Commentary, and Future Work

Matthew Neal
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA
meneal@ncsu.edu

Joseph Sankar
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA
jesankar@ncsu.edu

Alexander Sobran
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA
aisobran@ncsu.edu

ABSTRACT

This paper provides a comprehensive review and commentary on software defect prediction research. Review and criticism of previous work is presented. Future steps in the domain are given.

Keywords

Fault prediction model, Software mining, Ant Colony Optimization, Classification, Defect Prediction

1. INTRODUCTION

It is a well-known fact that software bugs are much cheaper and easier to fix before being released. But finding these bugs is often difficult and may not be cost effective to fix. Researchers have been working on fault detection models to predict which software modules are most likely to contain bugs post-release. Management can use these predictions to focus testing and bug-fixing efforts on those modules, resulting in fewer bugs in release which are less costly to fix.

In this paper, we focus on the advances made in software fault prediction models in the literature. Section 2 contains an assortment of existing work in the area.

2. RELATED WORK

Cagatay Catal and Banu Diri [3] gave an overview of software fault prediction studies and advancements up to 2008. They found that an increasing number of studies used datasets available to the public. They also found that since 2005, machine learning algorithms have become increasingly popular choices to implement the models. Finally, they observed that the most dominant metrics in fault prediction were at the method level. They recommended that machine learning algorithms and public datasets continue to be used, but caution against using method-level metrics and instead suggested class-level metrics as they can predict faults earlier in the software development cycle.

Vandecruys et al [15] mined software repositories to create predictive models. The authors used AntMiner+, an Ant Colony Optimization (ACO)-based classification technique. On public datasets, AntMiner+ was found to be competitive to alternative classification techniques, such as C4.5, logistic regression, and support vector machines. The authors suggested that software managers would find the output of rules produced by AntMiner+ easy to understand and accept.

While there have been plenty of studies about predicting software faults based on the code itself, few studies have looked at organizational structure as a factor. Nagappan et al [10] used organizational metrics, such as number of engineers, edit frequency, and organizational intersection factor to predict fault-proneness. The authors compared the effectiveness of the resulting model with alternative models which use traditional software metrics, like code churn and code complexity. They found that the model derived from organizational metrics had better precision and recall than others derived from software metrics, meaning they can also be effective indicators of failure-proneness.

Bird et al [2] examined whether development that is largely distributed produces more failures than development that is mainly collocated. The belief at the time was that global development was prone to more failures than collocated development. They found a negligible difference in both code metrics and failures between the two methods of development. The authors examined how the developers working on Windows Vista managed to work well together among teams located in different countries based on the relationships between the development sites, cultural barriers, communication, consistent use of tools, end to end ownership, common schedules, and organizational integration. They recommended that companies wishing to distribute development across sites located far apart to employ similar strategies to overcome some of the difficulties associated with such an endeavor.

Arisholm et al [1] examined different ways to build and evaluate fault prediction models. First, they tested a variety of modeling techniques, such as neural networks, C4.5 with some variants, support vector machines, and logistic regression. They then looked at different metrics: Process measures, object oriented code measures, and delta measures. Finally, they looked at ROC area and cost-effectiveness as evaluation criteria. They found that the choice of model-

WOODSTOCK '97 El Paso, Texas USA

DOI: 10.475/123_4

ing technique did not have much of an impact when evaluated with both criteria tested. Among the metric sets, they found that process measures provide a significant improvement over the others, even though they are typically more costly to collect. They also suggested cost-effectiveness as a good evaluation technique instead of traditional techniques like precision and recall as smarter decisions can be made to prioritize which parts of the project are tested and bug-fixed.

Jun Zheng [18] argued that the mistakenly predicting a module as non-defective is more dangerous and costly than mistakenly predicting a module as defective. He presented three algorithms which boost neural networks to predict software defects with cost in mind. When evaluated on four NASA datasets, he found that threshold moving algorithm gave the best results in the sole Normalized Expected Cost of Misclassification (NECM) measure. He especially recommended threshold-moving algorithms on projects written in object oriented languages.

Most software products are structured in a hierarchical format, for example into methods, classes, files, packages, etc. What level of analysis should fault prediction models operate on? And can analysis on one level of study apply to other levels? Posnett et al [12] examines these issues. They observed that sometimes relevant phenomena only occur at an aggregated level or data may only be observed at an aggregated level, meaning that studies are often conducted at aggregated levels. They also noted that in other fields of study there are ecological fallacies which make findings at aggregated levels not apply at disaggregated levels. They found that much of these fallacies also exist in the domain of computer software and that care needs to be taken when employing ecological inference. As to how exactly the risks of ecological fallacies can be dealt with, the authors left open for future research.

Traditional software fault prediction models are trained on historical project data. Since new projects do not have such a volume of historical data, these fault prediction models are not as effective at within-project defect prediction. A possible solution to this issue is cross-project prediction which uses data from one project to predict defects in another. Still, models using this technique exhibit poor performance. Rahman et al [14] argued that one reason for such behavior is that standard evaluation measures such as precision, recall, and F-measure are taken at specific threshold settings, while they really should be taken in a range of time/cost vs. quality trade offs. They took the standard measures at a variety of tradeoffs and found that cross-project defect prediction is at least as good as within-project defect prediction, and sometimes substantially better.

Also within the area of cross-project defect prediction, Nam et al [11] found that when the source and target projects have a different feature distributions, the resulting model gives poor performance. The authors use Transfer Component Analysis (TCA) to find a common feature set for both projects and then map the data of both projects to it. They found that TCA is sensitive to normalization, so they developed an improvement, TCA+ to select appropriate normalization options. After using TCA+ to create cross-project defect prediction models for eight open-source projects, they found a significant improvement in prediction performance compared to traditional algorithms and techniques. The authors proposed applying knowledge in one domain to another to further improve performance.

As another approach to improving fault prediction, Tian Jiang [8] introduced *personalized defect prediction.* He argued that developers have different coding styles and techniques and that if each developer had their own defect prediction model, performance would improve. He was careful to note that the developer was not a feature of the model, but that there was a model for each developer. He ran experiments on six large open-source projects using PCC+, a model which chooses the highest confidence prediction among CC (traditional change classification), PCC (personalized change classification), and weighted PCC (PCC with changes from other developers added to a developer's model). Compared to CC and MARS (another predictor which also creates different models for different groups of data), PCC+ outperformed CC, MARS, and PCC as long as there is enough training data for each developer. Jiang recommended that PCC+ be applied to other recommendation systems and types of predictions.

3. MACHINE LEARNING ALGORITHMS

Every fault prediction study used some sort of machine learning algorithm as the heart of the model. There are a variety of ML algorithms including regression, neural networks, support vector machines, and decision trees.

3.1 Regression

Regression is a ML algorithm for data that can be fitted to a curve. A total of X papers use some form of regression ([12], [14], ...). Regression algorithms are considered to be the simplest type, but can perform well if the data can be well fitted to some sort of line or curve.

3.2 Neural Networks

Neural networks are designed to model the learning pattern of the human brain. Essentially, neural networks consist of nodes connected by links. The more a link is activated during training, the stronger it becomes and the more likely it will be selected during testing.

Jun Zheng [18] used a back propagation neural network (BPNN) with AdaBoost, a way to achieve better classification results by producing diverse base classifiers. Zheng states that he used neural networks because they are popular in pattern recognition and BPNNs because they were the most frequent neural network used in the literature. AdaBoost was chosen because it is a popular way to improve the performance of the model.

3.3 Other ML Algorithms

Vandecruys et al [15] used a novel algorithm known as AntMiner+, which is based on Ant Colony Optimization which models the foraging behavior of ant colonies.

Tian Jiang's algorithm [8] was based on a classification scheme known as Personalized Change Classification+ (PCC+).

Nam et al [11] implemented TCA+, which is a novel feature extraction technique for transfer learning with improved normalization.

3.4 Discussion

Catal et al [3] suggested that more models be based on machine learning techniques rather than statistical methods or expert based systems. We see that research has followed the advice of the authors. Arisholm et al [1] did not find any major differences among the different machine learning techniques, although they found that C4.5 with Adaboost gave the best results.

Not only have authors used well-known ML algorithms, but they have even implemented new ones. This shows that ML algorithms were the primary choice of recent studies and still are today.

4. EVALUATION METRICS

There are three major approaches to evaluating the performance of models in the area of fault prediction: Confusion matrix approaches, ROC curve based approaches, and finally cost effectiveness approaches.

4.1 Confusion Matrix Approaches

The use of confusion matrices, and the metrics (precision, recall, and F-measure) that revolve around them are very common in the fault prediction area. As will likely be known by the reader, a confusion matrix is a summary of predicted values compared with actual values. A confusion matrix in most cases will have been created based on test data where the actual values are known. Precision, recall, and F-Measure are all statistics that can be used to assess how close a model has come in predicting the actual on the test data. This methodology is used in nearly all of the papers we read [12] [11] [2] [15] [10] [1] [14] [8].

Even though the methodology is fairly standard throughout the papers we read, what actual metric is reported between precision, recall, overall accuracy, and F-Measure has considerable variance between papers. The Vandecruys paper [15] even uses terminology that is separate from many other publications in using sensitivity for recall and specificity for the true negative rate, which is a seldom reported statistic. One suggestion in [7] is to present the confusion matrix itself rather than to present one statistic or the other when possible (where possible is determined by the sheer number of confusion matrices that would be produced). The idea is that it would better enable comparison across studies since researchers could produce whatever statistic they happen to need from the data itself.

4.2 ROC curves

The ROC curve is another fairly standard metric for performance used in the literature. ROC, or Receiver Operating Characteristic analysis is an elaboration of the confusion matrix approach. The idea is to use the rates of false positives and true positives on an x-y plane and to have x values represent the rates of false positives, and y values represent the rates of true positives. The rate for a particular model can be plotted as point on the ROC curve and the area under the curve (AUC) be used as a metric to determine the quality of the model with a point at (0,1) being optimal [12].

These methods are used in a few of the papers we reviewed [14] [1] [12]. Some negatives have been pointed out about the ROC approach and it's usefulness in certain circumstances. One such negative is the fact that ROC curve approaches have "an overly optimistic view of an algorithm's performance if there is a large skew in the class distribution". [6] Other papers mention the fact that both ROC/AUC approaches and confusion matrix based approaches in general do not address the whether a particular fault prediction methodology can pinpoint the source of the faults or not. Since knowing where to find errors is just as important or more important than knowing whether errors exist or not several papers find ROC/AUC insufficient as an evaluation metric [12] [1] [14].

4.3 Cost Effectiveness

Arisholm et al [1] contributed Cost Effectiveness as a metric to software defect prediction. The high level idea behind Cost Effectiveness is the idea that it is impossible in most cases to inspect the entire code base of a large project for bugs. Defect prediction models that can guide inspection of code to find the largest number of bugs by inspecting the smallest percentage of the code base are viewed as models with high cost effectiveness.

The basic idea of the metric is a set of two curves on an x-y plane that has a percentage of faults as the y-axis, and the percentage of the lines of code included in classes selected to focus verification as the x-axis. Classes are ranked according to their likelihood of having defects, first by the model, and next by the size of the class in case of ties. The graph has a baseline of $y = x$ which is essentially the assumption that the percentage of faults will be equivalent to the percentage of lines of code inspected, this is done using a random ranking of the classes. On the same graph is the cost effectiveness curve which is the actual percentage of faults given the percentage of LOC of class selected to focus verification according to the ranking determined by the model.

Arisholm et al use an area calculation that is normalized to be a proportion of the optimal area under the curve. To find an approximation of the optimal cost effectiveness they use a ranking that puts the most error filled classes towards the front and then create a cost effectiveness line according to that ranking. The actual calculation is
$CE_{\pi} = (CE_{pi}(model) - CE_{pi}(baseline))/$
$(CE_{pi}(optimal) - CE_{pi}(baseline))$ Other papers have used a simpler calculation of the area under the cost effectiveness curve, or AUCCE [12]. The same concept is also used by Rahman et al under the name AUCEC [14].

Those same papers that find ROC to be inefficient all suggest that Cost Effectiveness is an important and to some degree a superior metric to use in determining the performance of an algorithm [12] [1] [14]. The Arisholm paper was in fact one of the most influential papers that we worked with throughout the semester. Cross-project defect prediction has been validated to some degree by the existence of Cost Effectiveness as an evaluation metric [14]. The technique was used in Jiang et al as a metric as well [8].

4.4 Discussion

It seems that Arisholm's contribution to the community with Cost Effectiveness back in 2010 has provided a better metric to evaluate fault prediciton models going forward. Seeing as the point of fault prediction should be to build tools that can be used in the real world for real projects injecting the realism that CE brings to the table seems to be totally appropriate in going forward. Even though that is viewed as the case by some in the community confusion matrix based approaches and the related ROC curve concept are seemingly going nowhere. These approaches will continue to be important in evaluating models until something better comes along.

Our recommendation in this area would be to work with a hybrid approach of using both confusion matrix approaches and the ROC curve as well as CE based metrics. The suggestion of Hall as listed above to simply report confusion matrices seems like a great recommendation when possible going forward. At that point researchers could simply report cost effectiveness data along with confusion matrices and leave any further analysis to those that are interested. If reporting confusion matrices were not to become popular, settling on one confusion matrix based approach such as the F-Measure would be preferable to the random mixture of reporting approaches viewed throughout our papers.

5. FEATURES

This section could also be labeled metrics, as it is in many places throughout the literature on this topic, but to not confuse it and the previous section on evaluation metrics we have chosen to label it features. The main reasoning for this is the fact that metrics are actually used as features in this area and not actually as metrics for comparison of one method versus another. The Radjenovic paper [13] is a fairly exhaustive paper with relation to metrics/features in fault prediction. Radjenovic came up with three categories to discuss in their literature review, and we will stick with those same categories in relation to the papers we reviewed. The categories are: Traditional, Object-Oriented, and Process. Some of the papers we reviewed defy this categorization and use metrics that are either one offs or are hybridized to such a degree that they should be explained on their own.

5.1 Traditional

These are metrics that relate to complexity (ie McCabe or Halstead calculations) and lines of code. A good number of papers we reviewed used these metrics as feature [12] [15] [2]. There is considerable discussion in terms of whether these sorts of metrics provide good quality models or not. Hall et al indicate that in their review that models using only static code metrics that are complexity based have poor performance, but that Lines of Code based models tend to be useful [7]. Radjenovic et al indicate that LOC has no strong evidence behind it in terms of faults furthermore they state that smaller studies in general have given more weight to LOC and size metrics than larger studies, because of that they rated the effectiveness of the metric as "moderate"[13]. On complexity metrics Radjenovic states that these metrics are reasonable, but that "others are better"[13].

5.2 Object Oriented Metrics

Object oriented metrics seem to have been dominated by the metrics presented by Chidamber and Kemerer [4] [5]. These metrics include things like number of children(NOC), coupling between classes (CBO), or lack of cohesion of methods (LCOM). Object oriented metrics were used in only one of our papers, Arisholm et al, and used there really only as a comparison metric[1]. In terms of evaluation of their success or failure Radjenovic et al indicate that OO metrics are useful but that there is some debate about whether OO metrics and size are correlated and that further work is needed in that area [13]. Hall et al state that OO and LOC are actually on pretty equal footing and that OO provides better performance than source code metrics [7].

5.3 Process Metrics

Process metrics relate to things like number of developers, code churn, number of commits, features, etc. Process metrics are mostly mined from source code management systems like Git, and also bug tracking and issue tracking systems like Jira. These metrics have a strong following in the fault prediction community and are well represented in our papers [12] [2] [1] [14]. Rahman et al particularly only used a set of process metrics while the other papers mentioned used some sort of combination of process and other metrics. There is some dispute between Radjenovic and Hall with regards to process metrics, with Hall suggesting that their performance was the worst out of everything they had examined where Radjenovic considers process metrics to be promising and that they are deserving of further study and validation while additionally stating that they provide superior post-release fault prediction [7] [13].

5.4 Organizational Structure

This is an approach that was proposed by Nagappan et al at Microsoft Research [10]. The metrics included a set of 8 measures of organizational complexity: Number of engineers, Edit Frequency, Depth of Master Ownership, Percentage of Org contributing to development, Overall Organization Ownership, and Organization Intersection Factor. They used these metrics to create a fault prediction model and compared its performance in Precision and recall against a set of OO, Traditional, and Process metrics. For Windows Vista they show that Organizational metrics are superior to the other types of metrics available. As mentioned in the paper there are questions about whether this approach would work outside Microsoft or with a smaller project.

Naggapan and Bird produced another paper in the next year using organizational metrics again with a bit of a twist in looking at whether distributed development had an effect on software quality [2]. This paper added an additional metric of the actual location of the members of teams working on Windows Vista with levels starting at the same building and growing to the point of developers working at locations across the globe from one another. Organizaitonal metrics as well as a set of process and traditional metrics were used in conjunction with the distribution metric seemingly out of surprise that there was little difference between distributed projects and more local projects. The idea was to try to prove that there was not in fact some correlation between the other metrics they looked at and the performance of teams across long distances. They found that in fact there was no difference and that distributed development within one company was possible without sacrificing quality.

5.5 Text Classification

The paper on Personalized Defect Prediction by Jiang et al uses a methodology based on change classification [8]. To understand the metrics involved it was necessary to go back to a paper by Kim et al and get an idea of exactly what was done for feature extraction [9]. The basic methodology uses process metrics such as changed LOC between commits, and when commits take place, but also uses traditional techniques like LOC, and complexity analysis.

The more interesting and complex aspect of both of these papers is text classification. They both use a tool called BOW (Bag Of Words) to pull features out of log messages (called metadata in Jiang et all) and source code. The metadata aspect of this approach is also used in the Arisholm paper under the name Deltas where it is stated as simply the change between releases and also calls it Code Churn [1]. Code churn in later papers is simply subsumed into process metrics, where later survey papers like Radjenovic simply categorize them together [13].

The source code inspection can be incredibly low level as in the Kim paper where they "This (text classification) means that every variable, method name, function name, keyword, comment word, and operator, that is everything in the source code separated by whitespace or a semicolon is used as a feature"[9]. The Jiang paper pulled out features like incorrect calls of individual language level functions like malloc and calloc on a per developer basis and then used those to predict faults for individual developers. It is an intriguing approach and one that was successful based on Jiang's results, It is an approach that has been cited, but not repeated. Partially that may be because the paper is fairly recent having been published in 2013, but it may also owe something to the fact that this is a seemingly intensive effort that few would use in industry.

5.6 Benchmark Data Sets

In one of our most recent papers (the Nam paper 2013 [11]) we found researchers using benchmark data sets that had sets of metrics that could be chosen from in order to build models. The two sets discussed in the paper are ReLink and AEEEM [16] [5].

D'Ambros' AEEEM seemingly took everything he could find that he could rationalize and some new ideas and then just threw all of it into a metric set. AEEEM includes: change metrics, CK metrics, Object Oriented metrics, number of previous defects, complexity of code change, churn of CK and OO metrics, entropy of CK and OO metrics. ReLink unfortunately was not as helpful in listing out what is available in terms of metrics. Based on the Nam paper it contains 26 complexity metrics [11].

5.7 Discussion

As can be seen by the length and complexity of this section, the metrics area is still a very open area in fault defect prediction. The simple fact that there are major differences in terms of what metric set is being used across different projects creates some difficulty when comparing them. Also since many of these projects also use a hybrid approach, it makes it even less likely that another project will use exactly the same set of metrics.

There is simply no consensus on what a good metric set really is, and that makes research going forward a difficult proposition. Our recommendation would be for the community to continue to codify what they believe to be the best metric sets and then to build benchmark data sets like AEEEM and ReLink that can be used across projects so that the approaches of researchers can be appropriately compared.

6. DATA SOURCES

The choice of dataset is extremely important, especially when doing software defect prediction research. Datasets should be representative of actual software projects and should ideally be accessible to the public so the results can be retested and verified by others. Within the referenced papers, we found a variety of data sources which we feel need to be discussed. Primarily, we found open-source datasets, NASA datasets, and Windows Vista datasets. We will discuss each along with some reasons why they may have been chosen.

6.1 Open-source

The majority of papers we reviewed used open-source data, with most of those being Apache projects. Open-source projects are accessible to everyone, so researchers can verify the results they read, making every author accountable for the results they publish. Many of the Apache projects are tracked using JIRA which contains a plethora of data, including defect information.

Apache libraries are also commonly used in a variety of products, so defects that are in released code can manifest themselves in projects that use them, creating a sort of ripple effect.

6.2 NASA

A few papers (Zheng [18]...) used datasets from NASA. Among them were KC1, used for storage management, KC2, used for scientific data processing, CM1, used for NASA spacecraft instruments, and PC1, used for flight software.

One major reason for using NASA datasets is that most of their software is critical. A malfunction or crash due to a software defect is not only costly, but can also be deadly. NASA's software needs to be thoroughly tested and bug-fixed before it is ready for use. Software defect models which work well on these datasets can be claimed to be more reliable and trustworthy. Additionally, NASA's datasets contain code metrics and defect information, which is easy to mine and create training and test data from.

NASA datasets are publicly available. Researchers can more easily verify the results obtained in a study. Also, since these datasets are static, they can be used as a baseline to compare the results from a newly created model against those from other studies.

6.3 Windows Vista

Two of the referenced papers (Bird et al [2] and Nagappan et al [10]) use data from Windows Vista. Neither paper tests a fault prediction model using Vista test data. Instead, the first paper examined how different development teams worked together within Microsoft and the second paper examined Microsoft's organizational structure. Two of the three authors of the second paper are associated with Microsoft Research.

But why was Windows Vista chosen? Windows in general is a widely used operating system which contains many lines of code. Windows is also known for being vulnerable without frequent patching and antivirus software. Windows Vista may have been the latest version of Windows at the time of both papers, but Vista is also known as a slow and buggy release. Maybe the authors intended to show that Vista did not contain an unusually high number of bugs compared to other releases, or maybe they just wanted to choose a Microsoft product that was well-known, whether for good or bad.

6.4 Discussion

Most of the studies we examined used open-source datasets which is in line with previous recommendations ([3], [15]). The two papers which used Windows Vista were not proposing a new or improved fault prediction model but were instead analyzing Microsoft's organizational structure to see how it may have affected bug levels. Although the NASA software isn't open source, we think that its datasets are acceptable because of their domain and the fact that NASA has made the datasets public for research purposes. Research has heeded the recommendations of earlier researchers and now primarily use open-source datasets

7. DATA PREPROCESSING

8. STATISTICAL TESTS

8.1 Mann-Whitney U test

The Mann-Whitney U test, a non-parametric test that determines if two independent distributions have equally large value, was used by Zhang et al [17] to pair-wise compare the distribution of metrics of projects in the domain of cross-project defect prediction.

8.2 Cliff's δ

Cliffs δ measures the effect size of the importance of the difference between two distributions. It is a nonparametric statistical test which represents the overlap two distributions. If two distributions are identical then the measurement is 0. It was used by Zhang et al [17] to determine the difference in the distributions of two project's metrics. If the difference of the effect size was large then the projects were not clustered together. This was done as second step in clustering similar projects and only on project's whose Mann-Whitney U test determined a statistically significant difference.

9. CROSS-PROJECT DEFECT PREDICTION

9.1 Context Factors

Context factors for projects were assessed by Zhanget et al [17] Used as a measurement of project similarity. The provides a set of measurements to rank a project's relatedness when performing cross-project defect prediction.

Programming Languages Projects can be divided into groups depending on the programming language they are written in. Projects written in multiple languages could be problematic unless multiple language categories are included.

Issue Tracking Whether or not a project uses an issue tracking system.

Total Lines Of Code Project size based on the total number of lines of code split by quartiles.

Total Number of Commits Project size as a function of the number of commits. Split into quartiles.

Total Number of Developers Project size as a determined by the number of developer. Split into quartiles.

9.2 Context-Aware Rank Tranformation

Zhang et al [17] propose a context-aware rank transformation approach. First, split the project up into non-overlapped groups based on the previous context factors. Then cluster the groups with similar context factor distributions. Obtain a ranking function based using the 10th quantiles of predictors. Apply this ranking function to bin the predictor values into the 10 levels.

9.3 Clustering

9.4 Math Equations

You may want to display math equations in three distinct styles: inline, numbered or non-numbered display. Each of the three are discussed in the next sections.

9.4.1 Inline (In-text) Equations

A formula that appears in the running text is called an inline or in-text formula. It is produced by the **math** environment, which can be invoked with the usual `\begin. . .\end` construction or with the short form `$. . .$`. You can use any of the symbols and structures, from α to ω, available in LaTeX[?]; this section will simply show a few examples of in-text equations in context. Notice how this equation: $\lim_{n\rightarrow \infty}x=0$, set here in in-line math style, looks slightly different when set in display style. (See next section).

9.4.2 Display Equations

A numbered display equation – one set off by vertical space from the text and centered horizontally – is produced by the **equation** environment. An unnumbered display equation is produced by the **displaymath** environment.

Again, in either environment, you can use any of the symbols and structures available in LaTeX; this section will just give a couple of examples of display equations in context. First, consider the equation, shown as an inline equation above:

$$\lim_{n\rightarrow \infty}x=0 \tag{1}$$

Notice how it is formatted somewhat differently in the **displaymath** environment. Now, we'll enter an unnumbered equation:

$$\sum_{i=0}^{\infty} x + 1$$

and follow it with another numbered equation:

$$\sum_{i=0}^{\infty}x_i=\int_{0}^{\pi+2} f \tag{2}$$

just to demonstrate LaTeX's able handling of numbering.

9.5 Citations

Citations to articles [?, ?, ?, ?], conference proceedings [?] or books [?, ?] listed in the Bibliography section of your article will occur throughout the text of your article. You should use BibTeX to automatically produce this bibliography; you simply need to insert one of several citation commands with a key of the item cited in the proper location in the `.tex` file [?]. The key is a short reference you invent to uniquely identify each work; in this sample document, the key is the first author's surname and a word from the title. This identifying key is included with each item in the `.bib` file for your article.

The details of the construction of the `.bib` file are beyond the scope of this sample document, but more information can be found in the *Author's Guide*, and exhaustive details in the *LaTeX User's Guide*[?].

This article shows only the plainest form of the citation command, using `\cite`. This is what is stipulated in the SIGS style specifications. No other citation format is endorsed or supported.

9.6 Tables

Table 1: **Frequency of Special Characters**

Non-English or Math	Frequency	Comments
Ø	1 in 1,000	For Swedish names
π	1 in 5	Common in math
\$	4 in 5	Used in business
Ψ_1^2	1 in 40,000	Unexplained usage

Figure 1: A sample black and white graphic.

Because tables cannot be split across pages, the best placement for them is typically the top of the page nearest their initial cite. To ensure this proper "floating" placement of tables, use the environment **table** to enclose the table's contents and the table caption. The contents of the table itself must go in the **tabular** environment, to be aligned properly in rows and columns, with the desired horizontal and vertical rules. Again, detailed instructions on **tabular** material is found in the *LaTeX User's Guide*.

Immediately following this sentence is the point at which Table 1 is included in the input file; compare the placement of the table here with the table in the printed dvi output of this document.

To set a wider table, which takes up the whole width of the page's live area, use the environment **table*** to enclose the table's contents and the table caption. As with a single-column table, this wide table will "float" to a location deemed more desirable. Immediately following this sentence is the point at which Table 2 is included in the input file; again, it is instructive to compare the placement of the table here with the table in the printed dvi output of this document.

9.7 Figures

Like tables, figures cannot be split across pages; the best placement for them is typically the top or the bottom of the page nearest their initial cite. To ensure this proper "floating" placement of figures, use the environment **figure** to enclose the figure and its caption.

This sample document contains examples of **.eps** files to be displayable with LaTeX. If you work with pdfLaTeX, use files in the **.pdf** format. Note that most modern TeX system will convert **.eps** to **.pdf** for you on the fly. More details on each of these is found in the *Author's Guide*.

As was the case with tables, you may want a figure that spans two columns. To do this, and still to ensure proper "floating" placement of tables, use the environment **figure*** to enclose the figure and its caption. and don't forget to end the environment with figure*, not figure!

9.8 Theorem-like Constructs

Other common constructs that may occur in your article are the forms for logical constructs like theorems, axioms, corollaries and proofs. There are two forms, one produced by the command `\newtheorem` and the other by the command `\newdef`; perhaps the clearest and easiest way to distinguish

Figure 2: A sample black and white graphic that has been resized with the `includegraphics` command.

them is to compare the two in the output of this sample document:

This uses the **theorem** environment, created by the `\newtheorem` command:

THEOREM 1. *Let f be continuous on $[a,b]$. If G is an antiderivative for f on $[a,b]$, then*

$$\int_a^b f(t)dt = G(b) - G(a).$$

The other uses the **definition** environment, created by the `\newdef` command:

Definition 1. If z is irrational, then by e^z we mean the unique number which has logarithm z:

$$\log e^z = z$$

Two lists of constructs that use one of these forms is given in the *Author's Guidelines*.

There is one other similar construct environment, which is already set up for you; i.e. you must *not* use a `\newdef` command to create it: the **proof** environment. Here is a example of its use:

PROOF. Suppose on the contrary there exists a real number L such that

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = L.$$

Then

$$l = \lim_{x\to c} f(x) = \lim_{x\to c} \left[gx \cdot \frac{f(x)}{g(x)}\right] = \lim_{x\to c} g(x) \cdot \lim_{x\to c} \frac{f(x)}{g(x)} = 0 \cdot L = 0,$$

which contradicts our assumption that $l \neq 0$. $\square$

Complete rules about using these environments and using the two different creation commands are in the *Author's Guide*; please consult it for more detailed instructions. If you need to use another construct, not listed therein, which you want to have the same formatting as the Theorem or the Definition shown above, use the `\newtheorem` or the `\newdef` command, respectively, to create it.

A *Caveat* for the TeX Expert

Because you have just been given permission to use the `\newdef` command to create a new form, you might think you can use TeX's `\def` to create a new command: *Please refrain from doing this!* Remember that your LaTeX source code is primarily intended to create camera-ready copy, but may be converted to other forms – e.g. HTML. If you inadvertently omit some or all of the `\def`s recompilation will be, to say the least, problematic.

10. CONCLUSIONS

This paragraph will end the body of this sample document. Remember that you might still have Acknowledgments or Appendices; brief samples of these follow. There is still the Bibliography to deal with; and we will make a disclaimer about that here: with the exception of the reference to the LaTeX book, the citations in this paper are to articles which have nothing to do with the present subject and are used as examples only.

Table 2: Some Typical Commands

Command	A Number	Comments
`\alignauthor`	100	Author alignment
`\numberofauthors`	200	Author enumeration
`\table`	300	For tables
`\table*`	400	For wider tables

Figure 3: A sample black and white graphic that needs to span two columns of text.

Figure 4: A sample black and white graphic that has been resized with the `includegraphics` command.

11. ACKNOWLEDGMENTS

This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you. In the present case, for example, the authors would like to thank Gerald Murray of ACM for his help in codifying this *Author's Guide* and the **.cls** and **.tex** files that it describes.

12. REFERENCES

[1] E. Arisholm, L. C. Briand, and E. B. Johannessen. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *J. Syst. Softw.*, 83(1):2–17, Jan. 2010.

[2] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy. Does distributed development affect software quality?: An empirical case study of windows vista. *Commun. ACM*, 52(8):85–93, Aug. 2009.

[3] C. Catal and B. Diri. A systematic review of software fault prediction studies. *Expert Systems with Applications*, 36(4):7346 – 7354, 2009.

[4] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, 20(6):476–493, June 1994.

[5] M. D'Ambros, M. Lanza, and R. Robbes. An extensive comparison of bug prediction approaches. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 31–41, May 2010.

[6] J. Davis and M. Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 233–240, New York, NY, USA, 2006. ACM.

[7] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. *Software Engineering, IEEE Transactions on*, 38(6):1276–1304, Nov 2012.

[8] T. Jiang, L. Tan, and S. Kim. Personalized defect prediction. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pages 279–289, Nov 2013.

[9] S. Kim, E. Whitehead, and Y. Zhang. Classifying software changes: Clean or buggy? *Software Engineering, IEEE Transactions on*, 34(2):181–196, March 2008.

[10] N. Nagappan, B. Murphy, and V. Basili. The influence of organizational structure on software quality: An empirical case study. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pages 521–530, New York, NY, USA, 2008. ACM.

[11] J. Nam, S. J. Pan, and S. Kim. Transfer defect learning. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 382–391, Piscataway, NJ, USA, 2013. IEEE Press.

[12] D. Posnett, V. Filkov, and P. Devanbu. Ecological inference in empirical software engineering. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, ASE '11, pages 362–371, Washington, DC, USA, 2011. IEEE Computer Society.

[13] D. Radjenovic, M. HericÂ■ko, R. Torkar, and A. ZivkovicÂ■. Software fault prediction metrics: A systematic literature review. *Information and Software Technology*, 55(8):1397 – 1418, 2013.

[14] F. Rahman, D. Posnett, and P. Devanbu. Recalling the "imprecision" of cross-project defect prediction. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 61:1–61:11, New York, NY, USA, 2012. ACM.

[15] O. Vandecruys, D. Martens, B. Baesens, C. Mues, M. De Backer, and R. Haesen. Mining software repositories for comprehensible software fault prediction models. *J. Syst. Softw.*, 81(5):823–839, May 2008.

[16] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung. Relink: Recovering links between bugs and changes. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, pages 15–25, New York, NY, USA, 2011. ACM.

[17] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou. Towards building a universal defect prediction model. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 182–191, New York, NY, USA, 2014. ACM.

[18] J. Zheng. Cost-sensitive boosting neural networks for software defect prediction. *Expert Systems with Applications*, 37(6):4537–4543, 2010.

APPENDIX

A. HEADINGS IN APPENDICES

The rules about hierarchical headings discussed above for the body of the article are different in the appendices. In the **appendix** environment, the command **section** is used to indicate the start of each Appendix, with alphabetic order designation (i.e. the first is A, the second B, etc.) and a title (if you include one). So, if you need hierarchical structure *within* an Appendix, start with **subsection** as the highest level. Here is an outline of the body of this document in Appendix-appropriate form:

A.1 Introduction

A.2 The Body of the Paper

A.2.1 Type Changes and Special Characters

A.2.2 Math Equations

Inline (In-text) Equations.

Display Equations.

A.2.3 Citations

A.2.4 Tables

A.2.5 Figures

A.2.6 Theorem-like Constructs

A Caveat for the TEX Expert

A.3 Conclusions

A.4 Acknowledgments

A.5 Additional Authors

This section is inserted by LATEX; you do not insert it. You just add the names and information in the `\additionalauthors` command at the start of the document.

A.6 References

Generated by bibtex from your .bib file. Run latex, then bibtex, then latex twice (to resolve references) to create the .bbl file. Insert that .bbl file into the .tex source file and comment out the command `\thebibliography`.

B. MORE HELP FOR THE HARDY

The sig-alternate.cls file itself is chock-full of succinct and helpful comments. If you consider yourself a moderately experienced to expert user of LATEX, you may find reading it useful but please remember not to change it.