Data Quality: Some Comments on the NASA Software Defect Data Sets

Martin Shepperd Qinbao Song Zhongbin Sun Carolyn Mair

Abstract

BACKGROUND – self evidently empirical analyses rely upon the quality of their data. Likewise replications rely upon accurate reporting and using the *same* rather than *similar* versions of data sets. In recent years there has been much interest in using machine learners to classify software modules into defect-prone and not defect-prone categories. The publicly available NASA datasets have been extensively used as part of this research.

OBJECTIVE – this short note investigates the extent to which published analyses based on the NASA defect data sets are meaningful and comparable.

METHOD – we analyse the five studies published in *IEEE Transactions on Software Engineering* since 2007 that have utilised these data sets and compare the two versions of the data sets currently in use.

RESULTS – we find important differences between the two versions of the data sets, implausible values in one data set and generally insufficient detail documented on data set pre-processing.

CONCLUSIONS – it is recommended that researchers (i) indicate the provenance of the data sets they use (ii) report any pre-processing in sufficient detail to enable meaningful replication and (iii) invest effort in understanding the data prior to applying machine learners.

I. Introduction

Presently there is a good deal of interest in using machine learning methods to induce prediction systems to classify software modules as faulty or not faulty. Accurate prediction is useful since it enables, amongst other things, testing resources to be targeted more accurately. A 2009 Mapping Study [4] identified 74 relevant studies and this has grown to 208 by the end of 2010 (as reported by Hall et al. [7]). These

Martin Shepperd is with Brunel University, UK.

Qinbao Song and Zhongbin Sun are with Xi'an Jiaotong University, China.

Carolyn Mair is with the University of the Arts, UK

studies have employed a range of methods (e.g. Bayesian, Support Vector Machines and instance-based learners) and applied them to different software defect data sets.

With so much research being undertaken, there is a clear need to combine individual results into a coherent body of knowledge. To accomplish this it is necessary to make valid comparisons between studies. This is facilitated where studies have used the same data sets. It also assumes that results are derived from meaningful data. However, a recent paper by Gray et al. [6] has raised important questions about the quality of the 13[1] software defect data sets that have been made publicly available and extensively used by researchers (e.g. [7] found more than a quarter of relevant defect prediction studies, that is 58 out of 208, made use of the NASA data sets). Therefore these concerns, about data integrity and inconsistencies between different versions of the NASA data sets in circulation, require urgent attention. To do otherwise undermines the scientific basis of empirical validation and replication of studies of software defect prediction.

This note builds upon initial work of Gray et al. who pointed out the quality problems with the NASA data sets currently in use by the research community. We consider the extent of the problem, describe in detail a preprocessing algorithm and the impact of various cleaning ordering issues, make different cleaned datasets publicly available and and conclude with suggestions as to the research community might avoid such problems in the future.

II. Investigation and Results

Machine learning is a data-driven form of research and so it comes as no surprise that data sets are archived and shared between researchers. In this regard, the Promise Data Repository[2] has served an important role in making software engineering data sets publicly available. For example, presently [June 14, 2012] there are 96 software defect data sets available. Amongst these are 13 out of the 14 data sets that have been provided by NASA and which were also available for download from the NASA Metrics Data Program (MDP) website[3].

Table I compares the two versions and in terms of cases (instances) and features (attributes). All data are for the raw, i.e. un-preprocessed versions of the files. We see that no two versions are identical although the scale of the differences varies considerably. We also note that the ordering of cases within

[1]There are presently 14 data sets but note that KC2 was not present on the MDP website and KC4 is not present in the Promise Data Repository.

[2]See `http:\\promisedata.org`

[3]Presently they may be found at `http://www.filesanywhere.com/fs/v.aspx?v=896a648c5e5e6f799b`

TABLE I

COMPARISON OF THE TWO VERSIONS OF THE NASA DEFECT DATA SETS

	Cases		Features	
Data Set	MDP	Promise	MDP	Promise
CM1	505	498	43	22
JM1	10878	10885	24	22
KC1	2107	2109	27	22
KC2	n.a.	522	n.a.	22
KC3	458	458	43	40
KC4	125	n.a	43	n.a
MC1	9466	9466	42	39
MC2	161	161	43	40
MW1	403	403	43	38
PC1	1107	1109	43	22
PC2	5589	5589	43	37
PC3	1563	1563	43	38
PC4	1458	1458	43	38
PC5	17186	17186	42	39

the data sets differs. This may impact validation strategies such as n-fold cross-validation if random folds are not utilised.

Next we consider the different types of data quality problem that might arise. These are defined in Table II and employed in Tables III and IV which analyse the data quality issues of each data set from the NASA MDP repository and Promise repository in more detail. Column A refers to a situation where two or more features contain identical values for all observations, i.e. for all cases. Column B refers to features that contain the same value, i.e. add no information. Column C counts the number of features that contain one or more missing observations and Column I counts the number of instances with missing values. We provide both values since both list and case-wise deletion are possible remedial strategies. For the same reason we give both feature (Column D) and case (Column J) counts of conflicting values. These arise when some implied relational integrity constraint is violated, e.g. LOC_TOTAL cannot be less than Commented LOC, since the former must subsume the latter. Another instance is where the MacCabe's v(G) is 128 for a module yet the count of executable lines of code is zero (see [5] for a brief tutorial article on integrity constraints). Next, Columns E and K give the counts of the number of features

TABLE II

DATA QUALITY

Column Label	Data Quality Category	Explanation	Example
A	Identical features	Refers to a situation where two or more features contain identical values for all cases.	F1=F2=F3 $\wedge$ F4=F5 $\implies$ 3 features are identical so could be deleted.
B	Constant features	Refers to features that contain the same value for every instance, i.e. add no information.	
C	Features with missing values	Counts the number of features that contain one or more missing observations.	F1 has 10 missing values $\wedge$ F3 has 3 missing values $\implies$ 2 features contain missing values.
D	Features with conflicting values	Counts features that violate some referential integrity constraint	F1 should equal F2+F3 but does not. We cannot say which feature is in error therefore $\implies$ 3 problematic features.
E	Features with implausible values	Counts features that violate some integrity constraint	F1 should be non-negative but contains 1 or more instances < 0 $\implies$ 1 problematic feature.
F	Total problem features	Count of features impacted by 1 or more of A-E. Since features may contain more than one problem this need not be the sum of A to E .	
G	Identical cases	Refers to a situation where two or more cases contain identical values for all features including class label.	
H	Inconsistent cases	As per G but the class labels differ, all other data item values are identical.	There are two identical modules M1 and M2 where M1 is labelled as fault-free and M2 is labelled as faulty.
I	Cases with missing values	Counts the number of cases that contain one or more missing observations	
J	Cases with conflicting feature values	Counts cases that contain features (2 or more by definition) that violate some referential integrity constraint. Count each case irrespective of the number of features implicated.	As per Column D
K	Cases with implausible values	Counts cases that violate some integrity constraint. Count each case irrespective of the number of features implicated.	As per Column E.
L	Total of data quality problem cases	Count of cases impacted by one or more of I to K that we denote DS'. Since cases may contain more than one problem this need not be the sum of I to K.	
M	Total problem cases according to [6]	Count of cases impacted by one or more of G to K denoted DS''	

TABLE III

DETAILED DATA QUALITY ANALYSIS OF THE NASA DEFECT DATA SETS BY FEATURES

	A		B		C		D		E		F	
	Identical features		Constant features		Features with missing values		Features with conflicting values		Features with implausible values		Total problem features	
Data Set	MDP	Prom	MDP	Prom	MDP	Prom	MDP	Prom	MDP	Prom	MDP	Prom
CM1	2	0	3	0	1	0	2	14	0	6	6	15
JM1	0	0	0	0	0	5	9	15	0	6	9	16
KC1	0	0	0	0	0	0	4	15	0	6	4	16
KC2	n.a.	0	n.a.	0	n.a.	0	n.a.	14	n.a.	6	n.a.	15
KC3	0	0	1	0	1	0	0	0	1	1	3	1
KC4	27	n.a.	26	n.a.	0	n.a.	3	n.a.	0	n.a.	30	n.a.
MC1	0	0	1	0	0	0	3	3	1	1	5	4
MC2	0	0	1	0	1	0	0	0	0	0	2	0
MW1	2	0	3	0	1	0	0	0	0	0	4	0
PC1	2	0	3	0	1	0	4	14	1	6	8	15
PC2	3	0	4	0	1	0	2	2	1	1	8	3
PC3	2	0	3	0	1	0	2	2	1	1	7	3
PC4	2	0	3	0	0	0	7	7	1	1	11	8
PC5	0	0	1	0	0	0	3	3	1	1	5	4

and cases containing one or more implausible values such as LOC=1.1. The checks are are described in the Appendix. The data quality problems are then summarised by the total number of features impacted (Column F) and cases (Column L).

Finally, note that all data sets 'suffer' from problems of duplicate cases. Some researchers have considered this to be a problem since the identical case may be used both for training and validation (e.g. [6], [10]), however, we do not fully concur since our view is that it depends upon the goal of the investigation. If one is concerned with generalisation to other settings and data sets then elimination of duplicate cases has some basis since duplicate instances may not be commonplace and will tend to lead to an over-optimistic view of predictive capability. If the research goal is to form a view of how the classifier will perform in a particular setting then naturally occurring duplicate cases (i.e. different software modules indicated by different module id's with similar profiles) offer either a learning opportunity (since a previously encountered, identical case should facilitate learning). We would also argue likewise with

TABLE IV

DETAILED DATA QUALITY ANALYSIS OF THE NASA DEFECT DATA SETS BY CASE

	G		H		I		J		K		L		M	
	Identical cases		Inconsistent cases		Cases with missing values		Cases with conflicting feature values		Cases with implausible values		Total problem cases, DS'		Total problem cases, DS''	
Dataset	MDP	Prom	MDP	Prom	MDP	Prom	MDP	Prom	MDP	Prom	MDP	Prom	MDP	Prom
CM1	26	94	0	2	161	0	2	3	0	1	161	3	178	61
JM1	2628	2628	889	889	0	5	1287	1294	0	1	1287	1294	3158	3165
KC1	1070	1070	253	253	0	0	12	14	0	1	12	14	945	947
KC2	n.a.	182	n.a.	118	n.a.	0	n.a.	38	n.a.	1	n.a.	38	n.a.	197
KC3	12	170	0	2	258	0	0	0	29	29	258	29	264	142
KC4	10	n.a.	9	n.a.	0	n.a.	125	n.a.	0	n.a.	125	n.a.	125	n.a.
MC1	7972	7972	106	106	0	0	189	189	4841	4841	4841	4841	7619	7619
MC2	4	6	0	2	34	0	0	0	0	0	34	0	36	5
MW1	15	36	5	7	139	0	0	0	0	0	139	0	152	27
PC1	85	240	13	13	348	0	3	26	48	49	355	74	411	196
PC2	984	4621	0	100	4004	0	129	129	1084	1084	4055	1163	4855	4297
PC3	79	189	6	9	438	0	2	2	52	52	444	54	490	138
PC4	166	166	3	3	0	0	60	60	111	111	112	112	182	182
PC5	15730	15730	1725	1725	0	0	185	185	1772	1772	1782	1782	15507	15507

inconsistent or conflicting cases. This leads to challenges for any learner. Removing them may distort results to being over-optimistic. The researchers' choices again depend upon the investigation goal.

Table V lists five empirical studies that have been published in TSE since 2007 of which three out of five report using the MDP versions of the data sets. This was established by emailing the authors of each study. Of course many other studies have been published elsewhere, but for brevity we focus on the community's flagship journal. What is clear is that there are differences in the base level data set version and in reporting detail. Unfortunately this hinders making sense of the combined results and building an overall body of knowledge. Unless these problems are resolved any attempt at meta-analysis will be compromised.

TABLE V

RECENT SOFTWARE DEFECT STUDIES PUBLISHED IN TSE BASED ON THE NASA DATASETS

Study	Year	Version	Pre-processing		
			Missing items	Inconsistent items	Duplicate cases
Menzies, Greenwald and Frank [16]	2007	Promise	×	×	×
Zhang and Zhang [21]	2007	n.a.	n.a.	n.a.	n.a.
Lessman, Baesens, Mues and Pietsch [12]	2008	MDP	×	×	×
Liu, Khoshgoftaar and Seliya [15]	2010	MDP	✓	✓	×
Song, Jia, Shepperd, Ying and Liu [19]	2011	MDP	✓	×	×

III. CLEANED VERSIONS OF THE NASA DATA SETS

In this section we address the problems identified previously, describe the pre-processing involved and make new versions of the data sets available for other researchers. This will enable a common basis for research and meaningful comparison between studies.

The pre-processing strategy is that first the problem data (e.g., cases with either conflicting feature values or implausible values) are discarded, and then the data, which are not problematic but do not help improve the defect prediction (e.g., the features with constant values and either identical or inconsistent cases), are removed. This results in data set DS being transformed to DS' and DS'' respectively. Procedure *NASA MDP Data Preprocessing Approach* provides the details.

The pre-processing algorithm consists of two parts: the first part (lines 3-24) deals with cases while the second part (lines 25-31) handles features. Cases with either implausible values or conflicting feature values are logically erroneous: they either are implausible or contain features that violate some referential integrity constraint, so they are removed first (lines 3-10). The identical cases may constitute problems as a consequence of the cross-validation strategy. The inconsistent cases are problematic since it is not obvious how the learner should be trained. Thus they are also both deleted (lines 11-20). Note that, the pre-processing order of these two situations cannot be swapped, otherwise some inconsistent cases may not be removed. For example, suppose cases *i* and *j* are a pair of inconsistent cases, and case *k* is identical to case *i* (i<j<k). Thus cases *k* and *j* are also a pair of inconsistent cases, and all these three cases should be removed. However, if cases *i* and *j* are removed first, then case *k* might not be removed as there is no longer any case that is inconsistent with case *k*. Lines 21-24 delete the cases with missing values. Finally, features with constant and identical values are removed (lines 25-31).

Procedure NASA MDP Data Preprocessing Approach

inputs: Data – the original NASA MDP data sets.

Flag – the indictor of whether or not the identical/inconsistent cases are removed, can be TRUE or FALSE.

output: DS′ or DS′′ – the preprocessed NASA MDP data sets. The former is with identical and inconsistent cases while the latter not.

```
//DS - a specific data set in Data.
//M - the number of cases in DS; N - the number of features in DS.
//DS.Value[i][j] - the value of feature j for case i in DS.
```

```
Data′ = NULL;
Remove features MODULE_ID and ERROR_DENSITY, convert ERROR_COUNT into defective or non-defective flags (1/0);
for each DS ∈ Data do
    for i = 1 to M do                                          //step 1: remove cases with implausible values
        for j = 1 to N do
            if DS.Value[i][j] is an implausible value then
                DS = DS - DS.Value[i][1...N];
    for i = 1 to M do                                          //step 2: remove cases with conflict feature values
        if DS.Value[i][1...N] contains conflict feature values then
            DS = DS - DS.Value[i][1...N];
    if Flag then
        for i = 1 to M - 1 do                                  //step 3: remove identical cases
            for k = i + 1 to M do
                if DS.Value[i][1...N] ≡ DS.value[k][1...N] then
                    DS = DS - DS.Value[k][1...N];
        for i = 1 to M - 1 do                                  //step 4: remove inconsistent cases
            for k = i + 1 to M do
                if DS.Value[i][1...N-1] ≡ DS.Value[k][1...N-1] and DS.Value[i][N] ≠ DS.Value[k][N] then
                    DS = DS - DS.Value[i][1...N];
                    DS = DS - DS.Value[k][1...N];
    for i = 1 to M do                                          //step 5: remove cases with missing values
        for j = 1 to N do
            if DS.Value[i][j] is a missing value then
                DS = DS - DS.Value[i][1...N];
    for j = 1 to N do                                          //step 6: remove constant features
        if DS.Value[1...M][j] is constant then
            DS = DS - DS.Value[1...M][j];
    for j = 1 to N - 1 do                                      //step 7: remove identical features
        for k = j + 1 to N do
            if DS.Value[1...M][j] ≡ DS.Value[1...M][k] then
                DS = DS - DS.Value[1...M][k];
    Data′ = Data′ ∪ {DS};
If (!Flag) then DS′ = Data′ else DS′′ = Data′;
```

By applying the cleaning algorithm *NASA MDP Data Preprocessing Approach* to the original NASA data sets, we obtain the corresponding new version of the data sets, which were until recently available from their web site. Note that Gray et al. [6] suggest an alternative cleaning algorithm, however, as we have observed, there are ordering effects and we think it better to remove demonstrably incorrect data items first as this may resolve conflicting feature values and so less data are lost in the cleaning process. Such considerations illustrate the subtleties of data cleaning. The cleaned data sets are available from `http://j.mp/scvvIU`.

IV. Discussion

Our analysis of data sets used for defect prediction research poses two questions. Do the data quality and differences between data sets matter in any practical sense? And if so, what should researchers do about it?

This short paper raises some difficulties concerning the extensively used, collection of publicly available NASA software defect data sets. However, we consider it raises some significant and more widespread difficulties about how we conduct research into software defect prediction. This is for three reasons.

1) There is a growing movement towards ensuring that computational science (including machine learning) should generate reproducible results [20], [8], which is an undeniably good thing. A mainstay for this is sharing data and code. Such initiatives are undermined when differences, even subtle ones, between versions of the 'same' data are used, either due to different pre-processing or version control issues. When these differences are undetected by the research community the problems deepen.
2) The differences between versions of some of the NASA datasets are not large. Nevertheless it adds to the variance of the results so minimally it will make it more difficult to observe patterns across experimental results and will confound meta-analyses since a reduced proportion of the variability of the response variable (accuracy however measured, of the predictors under investigation) due to the treatments (different choices of learning algorithm, data set and so forth).
3) Generally we are dealing with small effects [17] but large samples (typically in the thousands or tens of thousands), consequently even small differences in training and validation data can lead to statistically significant differences in results. This in itself may be a reason to pay more attention to effect sizes and less to null hypothesis testing and p values [1].

Given we believe data quality problems can matter considerably, we now move to the question of what might be done about it.

First, whilst sharing and making data available to members of the research community is clearly a good thing, given the possibility of differences being injected through copying and sharing, researchers should indicate the source of their data. As confirmation that this is not an isolated problem consider the proliferation of *differing* versions of Fisher's famous Iris data set which has been used to explore linearly inseparable classification since the 1920s [2].

Second, as the methods of computational science become increasingly involved and demanding, great care is needed to ensure sufficient attention is paid to the data *as well as* the algorithms. Many machine and statistical learning methods are intricate and require a good deal of skill. This can divert attention away from data issues. In addition there is a danger that secondary data analysis distances the researcher from the real world phenomenon represented by the data. The meaning of the data can be lost and researchers may not know or ask what is meaningful. Consider the situation of a data item of zero LOC. Is this plausible? In some programming languages and depending upon how LOC is defined in the first place it is possible though unlikely. Being distanced from the data collection makes it hard to answer these kind of questions. We are not arguing against secondary analysis but pointing out it does bring some attendant dangers. Consequently, detailed documentation of the data is essential.

For example, we have observed, in common with Boetticher [3], some of the data sets contain implausible values such as LOC=1.1. Given that some of these have occurred in the first case and feature (e.g. data sets CM1 and PC1) it is striking that this has elicited so little comment from those using these files. We collectively must be more zealous to police the quality of the data that drives our research. As Jian et al. state "as we present our research results, we rely on the integrity of metric collection process and the description of software metrics reported in MDP repository" [9]. The problem is compounded with duplicate versions of data sets that turn out to be inexact duplicates.

Therefore the role of groups such as the Promise Data Repository who manage public archives of data sets needs to be extended to embrace data quality issues. Systematic reviews on how data quality is handled within empirical software engineering by Liebchen and co-workers [14], [13], [18] indicate that presently there is diversity in approach and scope for improvement. Even a simple traffic light system indicating the level of confidence in a data set could be useful. The inputs for determining colour would be the extent of (i) documentation and (ii) reproducibility of results based upon the data set.

Third, as is evident from Table V, authors (including ourselves) have not been in the habit of providing complete information regarding pre-processing of data. Given that many reported differences between machine learners are quite modest, the means by which missing values are handled and whether duplicates are removed or inconsistent values checked, matter a good deal. Keung et al. [11] also comment that

"ranked estimator lists are highly unstable in the sense that different methods combining with different preprocessors may yield very different rankings, and that a small change of the data set usually affects the obtained estimator list considerably." Thus the trivial detail may have a far reaching impact upon the final results and conclusions. These problems can be addressed by agreed reporting protocols which need to be developed and owned by the research community.

To conclude, some of the differences and data quality may seem trivial and sometimes impact only a small proportion of the observations. However, if our research is to have the respect of our fellow scientists then addressing such problems is not optional.

Lastly, we should stress the foregoing discussion is not in anyway a criticism of NASA, rather it raises some questions concerning how we, as a research community, have made use of such data in order to learn more about predicting defect-prone software modules.

ACKNOWLEDGMENT

We would like to thank the NASA MDP organisation for making their defect data sets publicly available. We also thank David Bowes and David Gray for useful discussions, the authors of the other TSE studies for providing additional information concerning their analysis techniques, and the referees for their careful reading and constructive comments. This work is supported in part by the National Natural Science Foundation of China under grant 61070006.

REFERENCES

[1] S. Armstrong. Significance tests harm progress in forecasting. *International Journal of Forecasting*, 23(2):321–327, 2007.

[2] J. Bezdek, J. Keller, R. Krishnapuram, L. Kuncheva, and N. Pal. Will the real Iris data please stand up? *IEEE Transactions on Fuzzy Systems*, 7(3):368–369, 1999.

[3] G. Boetticher. *Improving credibility of machine learner models in software engineering*, pages 52–72. Idea Group Inc, London, 2007.

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

[5] W. Fan, F. Geerts, and X. Jia. A revival of integrity constraints for data cleaning. *Proc. VLDB Endow.*, 1(2):1522–1523, Aug. 2008.

[6] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson. The misuse of the NASA metrics data program data sets for automated software defect prediction. In *EASE 2011*, Durham, UK, 2011. IET.

[7] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, Accepted for publication - available online, 2011.

[8] D. Ince, L. Hatton, and J. Graham-Cumming. The case for open computer programs. *Nature*, 482(7386):485–488, 2012.

[9] Y. Jiang, B. Cukic, and T. Menzies. Fault prediction using early lifecycle data. In *The 18th IEEE International Symposium on Software reliability Engineering (ISSRE)*, pages 237–246. IEEE, 2007.

[10] K. Kaminsky and G. Boetticher. Building a genetically engineerable evolvable program (geep) using breadth-based explicit knowledge for predicting software defects. In *IEEE Annual meeting of the Fuzzy Information Processing Society*, pages 10–15, Banff, Canada, 2004. IEEE Computer Society.

[11] J. Keung, E. Kocaguneli, and T. Menzies. A ranking stability indicator for selecting the best effort estimator in software cost estimation. *Automated Software Engineering*, Accepted for publication, 2011.

[12] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, 34(4):485–496, 2008.

[13] G. Liebchen. *Data Cleaning Techniques for Software Engineering Data Sets*. Doctoral thesis, Brunel University, 2011.

[14] G. Liebchen and M. Shepperd. Data sets and data quality in software engineering. In *PROMISE 2008*, Leipzig, 2008. ACM Press.

[15] Y. Liu, T. Khoshgoftaar, and N. Seliya. Evolutionary optimization of software quality modeling with multiple repositories. *IEEE Transactions on Software Engineering*, 36(6):852–864, 2010.

[16] T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1):2–13, 2007.

[17] T. Menzies and M. Shepperd. Editorial: Special issue on repeatable results in software engineering prediction. *Empirical Software Engineering*, 17(1-2):1–17, 2012.

[18] M. Shepperd. Data quality: Cinderella at the software metrics ball? In *Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics*, pages 1–4. ACM, 2011.

[19] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu. A general software defect-proneness prediction framework. *IEEE Transactions on Software Engineering*, 11(3):356–370, 2011.

[20] V. Stodden. The scientific method in practice: reproducibility in the computational sciences. 2010.

[21] H. Zhang and X. Zhang. Comments on "Data mining static code attributes to learn defect predictors". *IEEE Transactions on Software Engineering*, 33(9):635–636, 2007.

APPENDIX

The data cleaning tool may be downloaded from http://nasa-softwaredefectdatasets.wikispaces.com/.

Implausible values

LOC_TOTAL = 0

value of any attribute is < 0

any count is a non-integer

Referential integrity checks

(1) NUMBER_OF_LINES $\geq$ LOC_TOTAL

(2) NUMBER_OF_LINES $\geq$ LOC_BLANK

(3) NUMBER_OF_LINES $\geq$ LOC_CODE_AND_COMMENT

(4) NUMBER_OF_LINES $\geq$ LOC_COMMENTS

(5) NUMBER_OF_LINES $\geq$ LOC_EXECUTABLE

(6) LOC_TOTAL $\geq$ LOC_EXECUTABLE

(7) LOC_TOTAL $\geq$ LOC_CODE_AND_COMMENT

(8) NUM_OPERANDS $\geq$ NUM_UNIQUE_OPERANDS

(9) NUM_OPERATORS $\geq$ NUM_UNIQUE_OPERATORS

(10) HALSTEAD_LENGTH = NUM_OPERATORS + NUM_OPERANDS

(11) CYCLOMATIC_COMPLEXITY $\leq$ NUM_OPERATORS+1

(12) CALL_PAIRS $\leq$ NUM_OPERATORS

(13) HALSTEAD_VOLUME = (NUM_OPERATORS+NUM_OPERANDS)
*$\log_2$(NUM_UNIQUE_OPERATORS+NUM_UNIQUE_OPERANDS)

(14) HALSTEAD_LEVEL = (2/NUM_UNIQUE_OPERATORS)
*(NUM_UNIQUE_OPERANDS/NUM_OPERANDS)

(15) HALSTEAD_DIFFICULTY = (NUM_UNIQUE_OPERATORS/2)
*(NUM_OPERANDS/NUM_UNIQUE_OPERANDS)

(16) HALSTEAD_CONTENT = HALSTEAD_VOLUME/HALSTEAD_DIFFICULTY

(17) HALSTEAD_EFFORT = HALSTEAD_VOLUME*HALSTEAD_DIFFICULTY

(18) HALSTEAD_PROG_TIME = HALSTEAD_EFFORT/18