

“Bad Smells” in Software Analytics Papers

Tim Menzies

*Dept. of Computer Science
North Carolina State University, USA*

Martin Shepperd

*Brunel Software Engineering Lab (BSEL)
Dept. of Computer Science
Brunel University London
UB8 3PH, UK*

Abstract

CONTEXT: There has been a rapid growth in the use of data analytics to underpin evidence-based software engineering. However the combination of complex techniques, diverse reporting standards and poorly understood underlying phenomena are causing some concern as to the reliability of studies.

OBJECTIVE: Our goal is to provide guidance for producers and consumers of software analytics studies (computational experiments and correlation studies).

METHOD: We propose using “bad smells”, i.e., surface indications of deeper problems and popular in the agile software community and consider how they may be manifest in software analytics studies.

RESULTS: We list 12 “bad smells” in software analytics papers (and show their impact by examples).

CONCLUSIONS: We believe the metaphor of bad smell is a useful device. Therefore we encourage more debate on what contributes to the validity of software analytics studies (so we expect our list will mature over time).

1. Introduction

The drive to establish software engineering as an evidence-based discipline has been gaining momentum since the seminal article of Kitchenham et al. [56]. In parallel there has been a massive growth in the amount of publicly available software data and sophisticated data analytics methods. This has resulted in a sharp increase in the number and reach of empirically based, data driven studies that are generally termed software analytics.

Typically software analytics studies seek to distill large amounts of low-value data into small chunks of very high-value information. Studies are more data-driven than theory-driven. For example, after examining many software projects, certain coding styles could be seen to be more bug prone. Follow up experimentation, by manipulating the coding styles, could lead us to believe there is causality. Hence we might

recommend some coding styles should be avoided (subject to various context-related caveats). However, absence of theory can lead to challenges. In particular, the lack of strong prior beliefs may make it difficult to choose between or evaluate potentially millions of possibilities [24, 28].

Thus far, so good. Unfortunately, concerns are being expressed about the reliability of many of these results both from within software engineering (SE) [93, 53] and more widely from other experimental and data driven disciplines such as the bio-medical sciences [50, 31]. This has reached the point that in experimental psychology researchers now refer to the “replication crisis”. This in turn raises questions of study validity.

Arguably the situation is not dissimilar in software engineering. In a major systematic review of SE experiments (1993–2002) Kampenes et al. [54] found similar effect sizes being reported to psychology. In 2012 we published an editorial for a Special Issue in the journal *Empirical Software Engineering* on repeatable results in software engineering prediction [77] where the principal focus was “conclusion instability”. In that editorial, we raised concerns about the numerous occasions where Study 1 concluded X yet Study 2 concluded $\neg X$. The inability to resolve these seeming contradictions leaves us at an impasse. Clearly, we need better ways to conduct and report our work, as well as to examine the work of others.

In this article we consider what may be learned from other disciplines and what specific lessons should be applied to improving the reporting of software analytics type studies. “Bad smells” is a term that comes from the agile community. According to Fowler [5], bad smells (a.k.a. code smells) are “a surface indication that usually corresponds to a deeper problem”. See Table 1 for a summary of the bad smells identified in this article.

As per Fowler, we say that the “smells” listed in this article do not necessarily indicate that the conclusions from the underlying study must be rejected. However, we believe they raise three areas of potential concern.

1. For the author(s) they reduce the likelihood of publication.
2. For the reader they raise red flags concerning the reliability of the results (i.e., their consistency) and the validity (i.e., the correctness of the analysis or how well it captures those constructs we believe it represents) .
3. For science, they hinder the opportunities of pooling results via meta-analysis and building bodies of knowledge.

So the goal of this article is to identify “smells” in order to (i) encourage a wide-ranging community debate and (ii) assist in the identification of potential problems and associated remedies.

The remainder of this article is organized as follows. The rest of this section addresses scope and then tackles two frequently asked questions about this work. Section 2 explores historical work on writing ‘good’ articles and methodological guidance, coupled with how other data-driven disciplines have addressed experimental reliability concerns. Next, in Section 3 we describe our approach to identifying “bad smells”. This is followed by Section 4 that lists some key symptoms or ‘smells’ in approximate

Table 1: Software Analytics Bad Smells Summary

	“Bad smell”	Core problem	Impact	Remedy
1	Not interesting	The software analytics equivalent of how “many angels can dance on a pinhead”. This may result from an absence of theory.	Research that has negligible software engineering impact	Dialogue with practitioners [6, 70, 51]
2	Not using related work	Unawareness of related work on (i) the research question and/or (ii) state of the art in relevant techniques.	(i) Unwitting duplication (ii) Not using state of the art methods (iii) Using unchallenging / outdated benchmarks	Read! Utilize systematic reviews whenever possible. Search for relevant theory or theory fragments. Utilize technical guidance on data analytics methods and statistical analysis. Speak to experts.
3	Using deprecated or suspect data e.g., D has been widely used in the past ...	Using data for convenience rather than for relevance	Data driven analysis is undetermined [2]	Justify choice of data sets wrt the research question.
4	Inadequate reporting	Partial reporting of results e.g., only giving means without measures of dispersion and/or incomplete description of algorithms	(i) Study is not reproducible [72] (ii) meta-analysis is difficult or impossible.	Provide scripts / code as well as data. Provide raw results [41, 79].
5	Under-powered studies	Small effect sizes and little or no underlying theory	Under-powered studies lead to over-estimation of effect sizes and replication problems	Analyse power in advance. Be wary of searching for small effect sizes, avoid Harking and p-hacking [15, 53, 79].
6	$p < 0.05$ and all that!	Over-reliance on, or abuse of, null hypothesis significance testing	A focus on statistical significance rather than effect sizes leading to vulnerability to questionable research practices and selective reporting [53]	Report effect sizes and confidence limits [33, 65].
7	Assumptions of normality and equal variances in statistical analysis	Impact of heteroscedasticity and outliers e.g., heavy tails ignored. Assumptions of analysis ignored.	Under-powered studies lead to over-estimation of effect sizes and replication problems.	Use robust statistics / involve statistical experts [58, 110].
8	No data visualisation	Simple summary statistics e.g., means and medians may mask underlying problems or unusual patterns.	Data are misinterpreted and anomalies missed.	Employ simple visualisations for the benefit of the analyst and the reader [80, 44].
9	Not exploring stability. Only reporting best or averaged results.	No sensitivity analysis. p-hacking	The impact of small measurement errors and small changes to the input data are unknown but potentially substantial. This is particularly acute for complex models and analyses.	Undertake sensitivity analysis or bootstraps/randomisation to understand variability. Minimally report all results and variances [73, 87].
10	Not tuning	Biased comparisons e.g., some models / learners tuned and others not. Non-reporting of the parameter settings, the tuning effort / expertise required.	Under-estimation of the performance of un-tuned predictors. Inability to reproduce results.	Read and use technical guidance on data analytics methods and statistical analysis [99, 37]. Unless relevant to the research question avoid off-the shelf defaults for sophisticated learners.
11	Not exploring simplicity	Excessively complex models, particularly with respect to sample size	Dangers of over-fitting	Independent validation sets or uncontaminated cross-validation [18].
12	Not justifying choice of learner	Permuting / recombining learners to make ‘new’ learners in a quest for spurious novelty	Under / non reporting of “uninteresting results” leading to over-estimates of effect sizes.	Principled generation of research questions and analysis techniques. Full reporting.

order of importance. Finally, in Section 5 we consider the implications for the future and how we, the research community, might collectively take ownership of these “smells”, re-prioritise, re-order, add and subtract “smells”.

1.1. Scope

Software analytics research is relatively new and growing fast, so much of our advice relates to articles discussing induction from software project data. We discuss problems with empirical data analysis via experiments, quasi-experiments and correlation studies [91] rather than issues relating to e.g., qualitative studies. Other articles should be consulted for tips and traps relating to qualitative research and experimentation involving human participants, e.g., [102, 25, 60, 84].

Finally, in order to delineate the scope of this article we feel it helpful to add some remarks concerning the role of theory in software analytics type research. Naturally much analytics research will be data-driven or inductive in nature. In such circumstances the role of theory or deductive reasoning will be secondary. This of course begs the question of what is a theory. Unfortunately there is limited agreement as to what constitutes a theory [97, 52], other than it may comprise constructs, relationships between constructs (possibly causal ones), scope and possibly propositions [101]. Most commentators are of the view that theory use is low and might fruitfully be increased. As a step towards this Stol and Fitzgerald [101] explore the notion of theory fragments which might take the form of propositions. Those that are testable might form hypotheses i.e., a mechanism from theory to the empirical world and putative propositions as a means of navigating from empirical observations back to theory.

Having said this, we do not include a detailed treatment of software engineering theory. Data analytics studies are almost always theory light because they’re inductive in their approach. The use of theory, or not, will be highly context sensitive. There is not presently consensus as to what theories (or for that models) should look like. Finally, as indicated, there are a number of good treatments that already exist (see [97, 52, 101]).

1.2. But is this article necessary?

Simply put: is this article necessary? Is not everything here just material that “everybody knows”?

In response, we first note that, over some decades, the authors have been members of many conference and journal review panels. Based on that experience, we assert that demonstrably everybody does not know (or at least apply) this material. Each year, numerous research articles are rejected for violating the recommendations of this article, or worse still, some slip through the net. In our view many of those rejections are avoidable. We believe, many of the issues raised by this article can be repaired before submission – some with very little effort.

These difficulties are compounded by the fact that good practice evolves. For example, whilst the statistical education of many engaged in empirical software engineering research is grounded in classical statistics there have been more recent developments such as robust approaches [57], use of randomisation and bootstrap [16] and Bayesian techniques [39]. Likewise, there are significant and recent developments in machine

learning which, sometimes, authors overlook. For example, we have had on occasions to review and reject papers that are some very small delta to an article one of us wrote in 2007 [76]. Since that paper was written, much has changed¹ so a mere increment on that 2007 paper is unlikely to be a useful contribution or an acceptable publication.

We also observe that, on occasions, we find some reviewers who have the role of gatekeepers to scientific publication, also seem unaware of some, or much of this material. Thus there exists some research that should not be published, at least in its present form, that is published [58, 53]. This harms the progress of science and our ability to construct useful bodies of knowledge relevant to software engineering.

A particularly marginalized group are industrial practitioners trying to get their voice heard in the research literature. Unless that group knows the established norms of publication, their articles run the risk of rejection. We note that those norms of publication are rarely recorded – a problem that this article is attempting to solve. Hence, for all the above reasons, we argue that this article is necessary and useful.

1.3. *Should papers be rejected if they have bad smells?*

It is important to stress that this paper:

- is primarily intended as *guidance* for authors;
- is **not** intended to be used by reviewers as a mindless checklist of potential reasons-to-reject a paper. Our bad smells are *indicative* rather than *definitive* proof of a potential issue. Hence we stress that *their presence alone is not a reason for reviewers to reject a paper*. Context is extremely important.

One of the reviewers of this paper was kind enough to expand on this point. They commented (and we agree) that reviewers tend to become overly strict — to the point where they can reject papers with very interesting and important ideas — just because there was one point that did not satisfy certain guidelines. This is most acute for conferences where there is seldom a revision cycle. Reviewers should not just blindly enforce the adoption of all the points raised by this paper, but also need to be *sensitive* to the context of the study; certain bad smells may not be applicable (or may be less relevant). Additionally, we wish to avoid adopting a perspective, such that the requirement of a perfection prevents useful, pioneering or relevant — but slightly imperfect — research from ever being published.

2. Related Work

Software engineering (SE) researchers have posed the question from time to time: what makes a good article? Examples are the article by Shaw [92] back in 2003 and recently revisited by Theisen et al. [107]. Shaw stated that one difficulty SE researchers

¹For example, improved evaluation techniques and experimental design; new problem areas for SE and machine learning (e.g. [88]); novel and insightful new evaluation measures (see Section 5 of [49] or the multi-objective approaches discussed in [81]; and challenging new learners that are so slow that it becomes impractical to reproduce prior results (see the discussion on deep learning in [35]).

and authors face is that there are no well-developed or agreed guidelines as to how research should be conducted and presented. Although in the intervening years there has been increasing emphasis placed upon evidence, it is our view that the situation has not greatly changed. She also remarked on the weakness of many articles in terms of validation and Theisen et al. report that, if anything, reviewer expectations have increased. Potentially, software analytics articles have an advantage here given their empirical basis, however statistical analysis is not always undertaken well, as our article will demonstrate.

There have also been various audits of statistical quality of empirical research [58] and published guidelines and recommendations regarding analysis [58, 111]. The guidelines are welcome since, by and large, software engineers have limited formal training as statisticians. Note that when researchers do publish methodological guides, those articles tend to have very high citation counts², for instance:

- “Preliminary guidelines for empirical research in software engineering”, in *IEEE Transactions on Software Engineering* [58] (1430 citations since 2002).
- “Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings” in *IEEE TSE 2008* [67] (765 citations since 2008).
- “Guidelines for conducting and reporting case study research in software engineering”, in the *Empirical Software Engineering* journal [86] (2500 citations since 2009).
- “A practical guide for using statistical tests to assess randomized algorithms in software engineering” at ICSE’11 [4] (500 citations since 2011).

Of particular concern is our our ability to replicate results (or reliability), which many argue is the cornerstone of science. This was investigated systematically in experimental psychology by Nosek et al. [83] who concluded from a major replication project of 100 experiments in leading psychology journals that “replication effects were half the magnitude of original effects” and that while 97% of the original studies had statistically significant results, only 36% of replications did. Researchers began to talk about the “replication crisis”. Note, however, that there has been little discussion as to exactly what constitutes a confirmation of the original study [100] and that for underpowered studies (as seems to be typical) results may well be dominated by sampling error. Nevertheless, it seems that there is legitimate cause for concern and this has triggered recent how-to type articles such as Munafò et al. [79].

Despite this “replication crisis” having sparked considerable soul searching when it was reported, less progress is being made than we would hope e.g., in cognitive neuro-science and experimental psychology study power remains low and essentially unchanged over 50 years [104]. Smaldino and McElreath [98] make the case that given the incentive structures for researchers — specifically that publication is the principal factor for career advancement — it is unsurprising that low quality research studies

²The citation counts were extracted from Google Scholar, November 2018.

are still being conducted, submitted and published. As they put it, although this occurs without “deliberate cheating nor loafing” it helps explain why improvements in scientific practice remain slow.

We believe that whilst incentive structures clearly matter and influence scientific behavior, many of the problems we identify could be remedied with relatively little effort. We also hope that the intrinsic motivation of “doing good science” will be sufficient for the large majority of our research community.

3. Our Approach

Our goal is to assist researchers towards conducting better research and writing better articles. As steps towards that goal, we have a two-part proposal:

1. A part of this proposal is to run workshops at major software engineering conferences where best, and worst, practices in analysis and writing can be explored and documented. Accordingly, this article aims to recruit colleagues to this goal and thereby increase the number of researchers reflecting and commenting on these issues.
2. The other part of this plan is this article itself. Whilst we hope that these ideas will prove useful in their own right, we also hope that (a) this article inspires more extensive discussions about best and worst paper writing practices; and (b) subsequent articles extend and improve the guidance we offer below.

In order for this article to be effective we adopt a pragmatic approach. We present, a hopefully useful but not exhaustive, list of problems with software analytics papers. The level of detail is illustrative rather than in depth. In the spirit of brevity we restrict our focus to data driven research so, for example, case studies and experiments involving human participants are excluded.

Some “smells” are more self-evident and consequently need relatively little explanation. In contrast, others are more subtle and therefore we devote extra space to illustrating them. We use examples to help illustrate the smells and their relationship to actual problems. We do not intend examples as existence proofs, and in any case consider it invidious to single out individual authors as examples of widespread errors.

The list of “smells” has been constructed by discussion and debate between the two authors. We then cross-checked the list (see Table 2) with an empirical study quality instrument, [29] that has been used when assessing article quality for inclusion within software engineering systematic reviews [55, chapter 7]. Hence we see our list addresses all four areas of quality identified by the quality instrument and have more confidence in its coverage.

4. “Bad Smells” Unpacked

This section discusses the bad smells listed in Table 1. Before beginning, we note that while some of the following “smells” are specific to software analytics, others are general to many scientific endeavors. We include them altogether since they are all important issues about good science and effective scientific communication.

Table 2: Cross referencing research area to bad smell

Research area [55]	Quality instrument question [29, Table 3]	Bad smell
<i>Reporting</i> - quality	Is the article based on research? Is there a clear statement of the research aims? Is there an adequate description of the context? Is there a clear statement of findings?	Yes, defined by our scope 1 4 4, 11
<i>Rigour</i> - details of the research design	Was the research design appropriate? Was the recruitment strategy appropriate? Treatments compared to a control group? Data collected addressed research issue? Was the data analysis sufficiently rigorous?	5 No human participants, but more generally: 1, 3 2, 10 3 6, 7, 8, 9, 10
<i>Credibility</i> - are the findings of the study are valid and meaningful?	Has the relationship between researcher and participants been adequately considered?	No human participants
<i>Relevance</i> - of the study to practice	Is the study of value for research or practice?	1, 11

4.1. Not Interesting

All science studies should strive to be valid and reliable. But it is important that research is also interesting (to a wider audience than just the authors). This is not a clarion call for quirky or quixotic research, however, recall that our discipline is software *engineering*. Potentially somebody must be invested in the answers, or potential answers. One rule-of-thumb for increasing the probability that an article will be selected is validity plus *engagement* with some potential audience.

So what do we mean by “interesting”? The article needs a motivation or a message that beyond mundane. More specifically, *articles should have a point*. Research articles should contribute to a current debate. This could be via theory or with respect to the results of other studies. If no such debate exists, research articles should motivate one. For example, one could inspect the current literature and report a gap in the research that requires investigation. But to be clear, this does not preclude incremental research or even checking the reproducibility of results from other researchers, particularly if those results may have strong practical significance.

Note that problem of *not inspiring* is particularly acute in this era of massive availability of data and new machine learning algorithms; not all possible research directions are actually interesting. The risk can be particularly high when the work is data-driven which can quickly resemble data-dredging. For more on this point, see our last two bad smells (*Not enough theory* and *Not much theory*).

An antidote and powerful way to consider practical significance is with effect size [33] especially if this can be couched in terms that are meaningful to the target audi-

ence. For instance, reducing unit test costs by $x\%$ or $y\%$ is inherently more engaging than the associated p value of some statistical test is below an arbitrary threshold. The notion of smallest effect size of interest has also been proposed as a mechanism to maintain focus on interesting empirical results [64] a point we shall return to.

4.2. Not Using Related Work

This issue should not exist. Nevertheless, we know of many articles that fail to make adequate use of related work. Typically we see the following specific sub-classes of “bad smell”.

The literature review contains no or almost no articles less than ten years old. Naturally older papers can be very important— but the absence of any recent contribution would generally be surprising. This strongly suggests the research is not state of the art, or is not being compared with state of the art.

The review overlooks relevant, recent systematic literature review(s). This strongly suggests that there will be gaps or misconceptions in the way the state of the art is represented. It may also allow bias to creep in since the authors’ previous work may become unduly influential.

The review is superficial, even perfunctory. Such an approach might be caricatured along the lines of:

“Here is some related work [1, ..., 30]. Now to get on with the important task of my describing own work.”

This style of writing — where reviewing the work of others is merely a burden — makes it less likely connections will be seen, that the article contributes to a body of knowledge but conversely, increases vulnerability to confirmation biases [82]. It is also unhelpful to the reader since there is little or no interpretation of other work. What is useful? What has been superseded?

The review is thorough, relevant and fair, however, it is not used to motivate the new study. In other words there is a disconnect between the review and new research. New results are not related to the current state of the art. The consequence is the reader cannot determine what new contributions have been made. It may also mean appropriate benchmarks are missed.

Methodological aspects of the research are motivated by extremely simplistic or outdated sources e.g., using traditional statistics texts. One of the many dangers from using such sources is to buttress obsolete or inappropriate data analysis techniques or learning algorithms that are no longer competitive.

One constructive suggestion, especially if no relevant systematic literature review exists, is to employ *lightweight literature inspections*. These need not be as complex as a full systematic literature review (that can take months to complete). However, with the growing number of published systematic literature reviews it is increasingly probable that other researchers will have already addressed at least some part of the literature relevant to the research topic you are exploring.

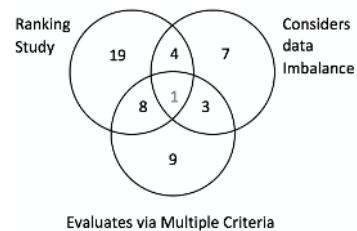


Figure 1: Summary of Table 3.

Table 3: 21 highly-cited software defect prediction studies, selected as follows: (a) from Google Scholar, search for “software” and “defect prediction” and “OO” and “CK” in the last decade; (b) delete anything with less than ten cites/year since publication; (c) search the resulting 21 articles for the three research items explored by Agrawal et al., i.e., do they rank multiple classifiers; are the classifiers ranked by multiple criteria; does the study consider class imbalance.

Reference	Year	Cites	Ranked Classifiers?	Evaluated using multiple criteria?	Considered Data Imbalance?
(a)	2007	855	✓	2	✗
(b)	2008	607	✓	1	✗
(c)	2008	298	✓	2	✗
(d)	2010	178	✓	3	✗
(e)	2008	159	✓	1	✗
(f)	2011	153	✓	2	✗
(g)	2013	150	✓	1	✗
(h)	2008	133	✓	1	✗
(i)	2013	115	✓	1	✓
(j)	2009	92	✓	1	✗
(k)	2012	79	✓	2	✗
(l)	2007	73	✗	2	✓
(m)	2007	66	✗	1	✓
(n)	2009	62	✓	3	✗
(o)	2010	60	✓	1	✓
(p)	2015	53	✓	1	✗
(q)	2008	41	✓	1	✗
(r)	2016	31	✓	1	✗
(s)	2015	27	✗	2	✓
(t)	2012	23	✗	1	✓
(u)	2016	15	✓	1	✗

For examples of lightweight literature inspections, see Figure 1 and Table 3 from Agrawal and Menzies [2]. Here, the researchers found a research gap in two dozen articles of the top-most cited software defect prediction studies that have at least 10 citations per year over the last decade. Specifically, as seen in Figure 1, most articles avoid issues of data imbalance and scoring via multiple criteria. Of course, research gaps need to be interesting (see Bad Smell #1). This finding led Agrawal and Menzies to perform a unique set of experiments that resulted in an ICSE’18 article.

4.3. Using Deprecated or Suspect Data

Researchers adopting a heavily data-driven approach, as is the case for software analytics, need to be vigilant regarding data quality. De Veaux and Hand point out “[e]ven when the statistical procedure and motives are correct, bad data can produce results that have no validity at all” [26]. A series of systematic reviews and mapping studies [68, 13, 85, 69] all point to problems of data quality not being taken very seriously and problems with how we definitively establish the validity of data sets, particularly when the gap in time and perhaps geography, between the collection and analysis is wide.

In terms of “smells” or warning signs we suggest researchers be concerned by the following:

- Very old data; e.g., the old PROMISE data from last-century NASA project (JM1, CM1, KC1, KC2, ...) or the original COCOMO data sets containing size and effort data from the 1970s.

- Problematic data; e.g., data sets with known quality issues (which, again, includes the old PROMISE data from last-century NASA project [94]).
- Data from trivial problems. For example, in the history of software test research, the Siemens test suite was an important early landmark in creating reproducible research problems. Inspired by that early work, other researchers have taken to document other, more complex problems³. Hence, a new paper basing all its conclusions on the triangle inequality problem from the Siemens suite is deprecated. Nevertheless tiny problems occasionally offer certain benefits, e.g., for the purposes of explaining and debugging algorithms, prior to attempting scale up.

That said, our focus is empirical studies in the field of software engineering where scale is generally the issue. In our experience, readers are increasingly interested in results from more realistic sources, rather than results from synthetic or toy problems. Given there now exist many on-line repositories where researchers can access a wide range of data from many projects (e.g. GitHub, GitTorrent⁴; and others⁵) this should not be problematic. Consequently, it is now normal to publish using data extracted from dozens to hundreds of projects e.g., [61, 3].

Finally, we acknowledge there are occasions when one must compromise, otherwise important and interesting software engineering questions may never be tackled. Not all areas have large, well-curated, high quality data sets. In such circumstances research might be seen as more exploratory or to serve some motivational role in the collection of better data. As long as these issues are transparent we see no harm.

4.4. Inadequate Reporting

Here there are two dimensions: completeness and clarity. Problems may be indicated by the following “smells”.

1. The work cannot be reproduced because the descriptions are too incomplete or high level. In addition, not all materials (data and code) are available [41, 79].
2. The work cannot be *exactly* reproduced because some of the algorithms are stochastic and seeds are not provided. Typical examples are the generation of folds for cross-validation in machine learning and heuristic search algorithms.
3. Non-significant and post hoc “uninteresting” results are not reported. This seemingly innocuous behavior, known as reporting bias, can lead to serious over-estimation of effect sizes and harm our ability to build bodies of knowledge via meta-analysis or similar means [41, 104].
4. The experimental details are full of “magic” numbers or arbitrary values for parameters and the experimental design. It is important to justify choices otherwise

³e.g. See the Software-artifact Infrastructure Repository at sir.unl.edu/portal.

⁴github.com/cjb/GitTorrent

⁵tiny.cc/seacraft

there are real dangers of researcher bias and over-fitting, i.e., choosing values that favor their pet algorithms. Standard justifications include “ k was selected via the standard Gap statistic procedure” or perform some sensitivity analysis to explore the impact of choosing different values [87].

5. It’s impossible for the reader to get a sense of what the article is about without reading it in its entirety. Structured abstracts (as per this article) can greatly assist both human readers [14, 59, 11] and text mining algorithms which are beginning to be deployed to assist meta-analysis, e.g., [104].
6. Using highly idiosyncratic section headings or article organization. Typically the following should suffice: introduction, motivation, related work, methods (which divides into data, algorithms, experimental method, and statistical procedures) followed by threats to discussion, threats to validity, conclusion, future work and references. Simple tables summarising algorithm details, hyper-parameters etc can be invaluable.
7. The article uses too many acronyms. Define terms when they are first used. If there are many, then consider using a glossary.

We conclude this sub-section with some simple and hopefully constructive suggestions.

Strive towards reproducible results: Table 4 defines a continuum of research artifacts and results. Reporting reproducible results (on the right-hand-side of that continuum) is the ideal case – but that can take a great deal of effort. For an excellent pragmatic discussion on how to structure your research artifacts in order to enhance reproducibility, see “Good Enough Practices for Scientific Computing” goo.gl/TD7Cax.






(Aside: Demands for increased reproducibility must be assessed on pragmatic grounds. If a prior study uses tools that are not open source, or tools that are very complex to use, then it is not appropriate to demand that subsequent work exactly reproduces the old work.)

Consider adding summary text to each table or figure caption, e.g., “Figure X: Statistical results showing that only a few learners succeed at this task”: Quite often, reviewers will first skim a article, glancing only at the figures. For such a reader, the locality of these annotations may prove extremely helpful.

Considering using research questions: Frequently software analytics research articles naturally sub-divide themselves into exploring a small set of research questions. Determining these research questions can require some creativity but is a powerful structuring device. Preferably there should be some progression so for example, here are the research questions from [37] (which discussed the merits of tuning the hyper-parameters of data mining algorithms):

- RQ1: Does tuning improve the performance scores of a predictor?
- RQ2: Does tuning change conclusions on what learners are better than others?
- RQ3: Does tuning change conclusions about what factors are most important in software engineering?
- RQ4: Is tuning impractically slow?

Table 4: The artifact continuum flows from left to right. More articles will be classified on the right once authors apply more of the methods on the left. This table presents the badges and definitions defined by the ACM (see goo.gl/wVEZGx). These definitions can be applied by the broader community as (1) aspirational goals, or (2) ways to audit past research, or (3) as guidance for organizing an “artifacts track” at a conference or journal.

No badge	Artifacts			Results	
	Functional	Reusable	Available	Replicated	Reproduced
					
	Artifacts documented, consistent, complete, executable, and include appropriate evidence of verification and validation.	Functional + very carefully documented and well-structured to the extent that reuse and re-purposing is facilitated. In particular, norms and standards of the research community for artifacts of this type are strictly adhered to.	Functional + placed on a publicly accessible archival repository. A DOI (Digital Object Identifier) or link to this repository along with a unique identifier for the object is provided. It is easy to obtain such DOIs: just host your artifact at repositories like SEACRAFT tiny.cc/seacraft .	Available + main results of the article have been obtained in a subsequent study by a person or team other than the authors, using, in part, artifacts provided by the author.	Available + the main results of the article have been independently obtained in a subsequent study by a person or team other than the authors, without the use of author-supplied artifacts.

- RQ5: Is tuning easy?
- RQ6: Should data miners be used “off-the-shelf” with their default tunings?

A useful typographical device sometimes associated with research questions is the use of results boxes to stress the answers to the questions. Here is an early example from Zimmermann et al. [114], 2004. Although potentially powerful such summaries need to be carefully constructed to avoid trivial over-simplification.

*One can either have **precise** suggestions or **many** suggestions, but not both.*

4.5. Under-powered Studies

The power of an analysis is the probability that a test will reject a false null hypothesis i.e., correctly report a non-zero effect. More generally, it is the ability to find an effect were one to exist. Power is the complement of β or the probability of making a Type II (failing to detect a true effect in favour of the null hypothesis, i.e., a false negative) error, hence $\text{power} = 1 - \beta$. However, there is a tradeoff with the probability of making a Type I error (i.e., a false positive). At an extreme we could guard against false positives by setting an extremely high threshold of acceptance but the consequence would be an elevated false negative rate. And of course *vice versa*.

So what should β be set to? Traditionally, it has been argued that power should be 0.8 [21] indicating that researchers might consider Type I errors (wrongly rejecting

the null hypothesis when in reality there is no effect) as being four times more problematic than Type II errors and therefore needing proportionately more checks. Thus, if significance levels are set at $\alpha = 0.05$ then it follows that $\beta = 4\alpha = 0.2$ hence, as required, our power is $1 - \beta = 0.8$. To repeat, the important point is that we must trade off between Type I and Type II errors and our preferences are somewhat arbitrary.

However, what we want the power of a study to be, and what it *actually* is are two different things. Four factors influence power, namely: (i) sample size, (ii) the unknown, true size of the effect being investigated, (iii) the chosen α and (iv) the chosen β [22, 33]. In addition, experimental design can improve power, for example by measuring within experimental unit (e.g., via a repeated measures design) rather than between unit variation [75].

Many commentators recommend, the power of a proposed experiment or investigation should be determined *a priori* [30]. The purpose is to check if an experiment has a reasonable chance of detecting an effect were one to actually exist. One obvious and immediate difficulty is estimating the effect size that is to be investigated [74]. This is compounded by the problem that there is compelling evidence that we are systematically over-estimating effect sizes due to reporting and publication bias [15, 53]. However, the problems of power go beyond merely wasting scientific resources in failing to detect effects that actually exist. There are two other serious negative consequences. First is the tendency of over-estimation of the measured effect sizes and second, is the elevated probability of false statistically significant findings [50, 23]. Indeed, it is no exaggeration to state that *low and under-powered studies are the greatest threat to meta-analysis and constructing meaningful bodies of software engineering knowledge* [15, 104].

It is ironic that the success of software analytics research in being able to uncover patterns and effects in large, complex and noisy data sets is also its undoing. This, coupled with the lack of guiding theory (which makes it difficult to differentiate between “winner’s curse” and some substantive effect⁶), agreed methods of analysis and excessive researcher degrees of freedom [95, 71] results in low-powered studies and a high probability of over-estimating effect sizes that are hard for other researchers to replicate [50, 93, 53]. As an antidote, we strongly recommend:

1. Determining in advance the smallest effect size of interest (SESOI) [64]. This may entail debate with the problem owners, typically practitioners. It also helps address our “Bad Smell #1” by reducing the likelihood of conducting research that nobody cares about.
2. Also, undertake power calculations in advance to determine the likelihood of the study finding an effect of the expected size. Where possible, when there are relevant previous studies, use a bias-corrected pooled estimate [90] to help address likely exaggerated estimates.
3. If the power is low either modify the study design [75], use a larger sample or

⁶Huang et al. [48] describe a parallel situation in genomics where millions of associations can be potentially discovered but the lack of theory means there is little guidance for researchers.

investigate a different question.

The “smell” associated with this kind of problem can be surprisingly subtle. However, one clue can be an argument along the lines of:

Despite a small sample size we were able to detect a highly significant effect with a very low p . Thus it is concluded that the effect must be even stronger than it first appears given the inherent difficulties in detecting it.

Unfortunately, nothing can be further from the truth. If the study is under-powered, possibly due to a small sample, then it is extremely probable that the effect is a false positive or a Type I error [71]. If the study has low power then that is a problem and one that cannot be fixed by hunting for small p values. The problem that arises from small samples is that the variability will be high and only the studies that by chance obtain a sufficiently low p -value — and therefore satisfy the acceptance criterion typically $\alpha = 0.05$ — and these will of necessity have exaggerated estimates of the effect. This is sometimes known as the “winner’s curse”. The probability that an observed effect that reaches statistical significance actually reflects the true effect can be computed as $([1 - \beta] \times R) / ([1 - \beta] \times R + \alpha)$ where $(1 - \beta)$ is the power, β is the Type II error, α is the Type I error and R is the pre-study odds that an investigated effect is truly non-null among all the effects being investigated) [71, 104]. Note that this issue leads to the next “bad smell”.

4.6. $p < 0.05$ and all that!

Though widely deprecated [17, 96, 23], null hypothesis significance testing (NHST) persists as the dominant approach to statistical analysis. Hence, the following fictitious⁷ examples “smell bad”:

Our analysis yields a statistically significant correlation ($r = 0.01$; $n = 18000$; $p = 0.049$; $\alpha = 0.05$).

and

Table X summarizes the 500 results of applying the ABC inferential test and reveals that 40 are statistically significant at the customary threshold of $\alpha = 0.05$.

In the first instance the low (and under the $\alpha = 0.05$ acceptance threshold for Type I errors) p -value should not be conflated with a practical, real world effect. A correlation of $r = 0.01$ is trivial and to all practical purposes indistinguishable from no correlation. It is simply the artifact of the very large sample size ($n = 18000$) and the logic of NHST which demands that empirical results be dichotomised into true (there is a non-zero effect) and false (the effect size is precisely zero).

Hence, methodologists encourage the notion of the *smallest effect size of interest* (SESOI). This should be specified *before* the research is conducted. It also has the merit

⁷Since problems with the use of p -values are widely agreed to be endemic [96, 53] we consider it invidious to single out individuals.

of re-engaging the researchers with the software engineering problems they are trying to solve. So for, say software effort prediction, after consultation with the problem owners we may conclude that the SESOI is to be able to reduce the mean inaccuracy by 25%. Generally, although not necessarily, effect sizes are expressed as standardized measures to enable comparisons between settings so the above SESOI could be normalized by the variability (standard deviation) of the observations. For accessible overviews see [33, 22]. It is useful for practitioners if the article report the effect size with confidence limits. After all, we experience effects not p values!

In the second example of a “bad smell”, the problem relates to the practice of performing excessive numbers of statistical tests and only attending to the ones that are “significant”. Given that 500 tests are performed and we accept the likelihood of wrongly rejecting the null hypothesis at 1 in 20 we would expect to observe 40 “positive” results just with random data! A range of corrections have been proposed [7]. Controlling for the false discovery rate using the Benjamini-Hochberg approach [8] can be appropriate as it corrects the acceptance threshold without being as severe as the traditional Bonferroni correction (α/t where t is the number of tests being undertaken).

One way to avoid excessive statistical tests is to cluster results before performing a statistical analysis. In this approach, instead of saying “what distributions are different?” some grouping operator is applied prior to the application of statistics. For example, consider the Scott-Knott procedure recommended by Mittas and Angelis [78]. This method sorts a list of l treatments with l_s measurements by their median score. It then splits l into sub-lists m, n in order to maximize the expected value of differences in the observed performances before and after divisions. The Scott-Knott procedure identifies one of these divisions to be ‘best’ as follows. For lists l, m, n of size l_s, m_s, n_s where $l = m \cup n$, the ‘best’ division maximizes $E(\Delta)$; i.e., the difference in the expected mean value before and after the split:

$$E(\Delta) = \frac{m_s}{l_s} \text{abs}(m.\mu - l.\mu)^2 + \frac{n_s}{l_s} \text{abs}(n.\mu - l.\mu)^2$$

Scott-Knott then checks if the ‘best’ division is actually useful. To determine this the procedure applies some statistical hypothesis test H to check if m, n are significantly different. If so, Scott-Knott then recurses on each half of the ‘best’ division.

For a specific example, consider the results from $l = 5$ treatments:

```
rx1 = [0.34, 0.49, 0.51, 0.6]
rx2 = [0.6, 0.7, 0.8, 0.9]
rx3 = [0.15, 0.25, 0.4, 0.35]
rx4= [0.6, 0.7, 0.8, 0.9]
rx5= [0.1, 0.2, 0.3, 0.4]
```

After sorting and division, Scott-Knott determines:

- Ranked #1 is rx5 with median= 0.25
- Ranked #1 is rx3 with median= 0.3
- Ranked #2 is rx1 with median= 0.5

treatment	Results from multiple runs									
no-SWReg	174	38	0	32	50	141	34	317	114	
no-SLReg	127	73	45	35	47	159	37	272	103	
no-CART-On	119	257	425	294	181	98	409	520	16	
no-CART-Off	119	257	425	294	181	98	409	520	16	
no-PLSR	63	213	338	358	163	44	14	520	44	
no-PCR	55	240	392	418	176	45	12	648	65	
no-1NN	119	66	81	82	70	110	118	122	92	
log-SWReg	109	23	9	7	52	100	12	164	132	
log-SLReg	108	61	49	53	55	156	28	183	101	
log-CART-On	119	257	425	294	181	98	409	520	16	
log-CART-Off	119	257	425	294	181	98	409	520	16	
log-PLSR	163	7	60	70	2	152	31	310	175	
log-PCR	212	180	145	155	70	68	1	553	293	
log-1NN	46	231	246	154	46	23	200	197	158	

NOTE: treatments have a pre-processor ('no' = no pre-processor; 'log' = use a log transform) and a learner (e.g., 'SWReg' = stepwise regression; CART = decision tree learner that post-prunes its leaves after building the tree).

Figure 2: How not to present data (in this case, percent error estimations where the treatment in column one was applied to multiple randomly selected stratifications of the data). Note that there is (a) no sort order of the treatments from most to least interesting; (b) no summary of the distributions; (c) no indication to indicate what values are best; and (d) no clustering together of results that are statistically indistinguishable. For a better way to present this data, see Figure 3.

- Ranked #3 is rx2 with median= 0.75
- Ranked #3 is rx4 with median= 0.75

Note that Scott-Knott finds little meaningful difference between rx5 and rx3. Hence, they have the same rank, even though their medians differ. For a larger example of this kind of analysis, see Figures 2 & 3.

The Scott-Knott procedure can ameliorate the excessive statistical tests problem. To see this, consider an all-pairs hypothesis test of 10 treatments. These can be compared statistically $(10^2 - 10)/2 = 45$ ways. To substantially reduce the number of pairwise comparisons, Scott-Knott only calls on hypothesis tests *after* it has found splits that maximize the performance differences. Hence, assuming binary splits of the whole data and all splits proving to be different, Scott-Knott would be called $\log_2(10) \approx 3$ times. A 95% confidence test run for these splits using Scott-Knott would have a confidence threshold of: $0.95^3 = 86\%$ assuming no corrections are made for multiple tests..

Implementations that use bootstrapping and a non-parametric effect size test (Cliff's Delta) to control its recursive bi-clustering are available as open source package, see goo.gl/uo3FL. That package converts data such as Figure 2 into reports like Figure 3.

4.7. Assumptions of normality and equal variances in statistical analysis

A potential warning sign or "bad smell" is when choosing statistical methods, to overlook the distribution of data and variances and focus on the "important" matters of description e.g., using measures of location such as the mean, and the application of inferential statistics such as t-tests.

It is now widely appreciated that applying traditional parametric statistics to data that violates the assumptions of normality, i.e., they deviate strongly from a Gaussian

RANK	TREATMENT	MEDIAN	(75-25)TH			
1	no-SWReg	50	107	- o	-----	
1	log-SWReg	52	97	o --		
1	log-SLReg	61	55	-o ----		
1	log-PLSR	70	132	- o	-----	
1	no-SLReg	73	82	- o	-----	
1	no-1NN	92	37		o	
2	log-PCR	155	142	---	o	-----
2	log-1NN	158	154	-	o	--
2	no-PLSR	163	294	--	o	-----
2	no-PCR	176	337	--	o	-----
3	log-CART-Off	257	290	----	o	-----
3	log-CART-On	257	290	----	o	-----
3	no-CART-On	257	290	----	o	-----
3	no-CART-Off	257	290	----	o	-----

Figure 3: Using Scott-Knott, there is a better way to representation the error results see in Figure 2 (and lower error values are better). Note that, here, it is clear that the “SWReg”, “CART” treatments are performing best and worst (respectively). Results from each treatment sorted by their median (and the first rows with less error are better than the last rows with most error). Results presented numerically and visually (right-hand column marks the 10th, 30th, 50th, 70th, 90th percentile range; and ‘o’ denotes the median value). Results are clustered in column one using a Scott-Knott analysis. Rows have different ranks if Cliff’s Delta reports more than a small effect difference and a bootstrap test reports significant differences. A background gray share is used to make different ranks stand out.

distribution *can* lead to highly misleading results [58]. However, four points may be less appreciated within the software engineering community.

1. Even minor deviations can have major impacts, particularly for small samples [110]. These problems are not always easily detected with traditional tests such as Kolmogorov-Smirnov so visualization via qq-plots should be considered.
2. Differences between variances of the samples or heteroscedasticity, may be a good deal more problematic than non-normality [34]. Thus the widespread belief that t-tests are robust to departures from normality when sample sizes are approximately equal and > 25 is unsafe [34, 113].
3. Although different transformations, e.g., log and square root can reduce problems of asymmetry and outliers they may well be insufficient to deal with other assumption violations.
4. Lastly, there have been major developments in the area of robust statistics over the past 30 years that are generally more powerful and preferable to non-parametric statistics. This includes trimming and winsorizing [46, 57].

Since SE data sets can be highly skewed, heavy tailed or both, we recommend performing robust tests for statistical inference. Note that we believe they should be preferred (wherever possible), to non-parametric statistics that are based on ranks. The latter, for example the Wilcoxon signed rank procedure can lack power and are problematic in face of many ties [10]. Kitchenham et al. [57] provide useful advice in the context of software engineering and Erceg-Hurn and Mirosevich an accessible general overview [34]. For an extensive and detailed treatment see [110].

Another approach is to use non-parametric bootstrapping (see Efron and Tibshirani [32, p220-223]). Note that Figure 3 was generated with such a bootstrap test. With

the advent of powerful computers, Monte Carlo and randomization techniques can be very practical alternatives to parametric methods [73].

4.8. No Data Visualisation

Whilst its customary to describe the data with a few summary statistics such as means, medians and quartiles this does not necessarily provide a check against unusual patterns and anomalies. Hence, we strongly advise *visualizing the data* first before *applying statistical tests*. For example:

- The horizontal box plots of Figure 3 can present results from a very large number of samples in a single row. A visual inspection of that data lets an analyst check if their conclusions are refutable due to, for instance extreme outliers or high variance.
- For another example, consider Figure 4 that shows results from a 5*5-cross validation experiment exploring the value of SMOTE:
 - SMOTE is a data rebalancing technique that adjusts training data such no class is a small minority [2].
 - A 5*5 cross-val experiment is an evaluation technique where $M =$ times, we randomize order of data. Each time divide the project data into N bins, train on $N - 1$ bins, test on the hold out (this is repeated for all bins). This procedure produces 25 results which in Figure 4 are reported as the medians and inter-quartile ranges (IQR).

From this visual inspection, before doing any statistics, we can observe that in no case is using SMOTE is worse that not using SMOTE. Also, as seen in the right-hand-side results of Figure 4, sometimes there are very large improvements from applying SMOTE.

4.9. Not Exploring Stability

For many reasons, software engineering data sets often exhibit large variances [77]. So it is important to check if the effect you want to report actually exists, or is just due to noise. Hence:

- When multiple data sets are available, try many data sets.
- Report measures of central tendency (e.g., median) *and* dispersion (e.g., standard deviation or interquartile range (IQR) (75th-25th percentile)).
- Do not just average results across N data sets. Instead, summarize the *spread* of results across your data sets which might include the creative use of graphics e.g. see Figure 4 above . Recall in that figure that the dashed lines at bottom report the variability in the 25 results shown for each data set in that figure. The key point here is that this variability is very small; i.e. differences between the median results is often much larger than the variability introduced by a stochastic selection of the train set (but, of course, that visual impressions requires a sanity

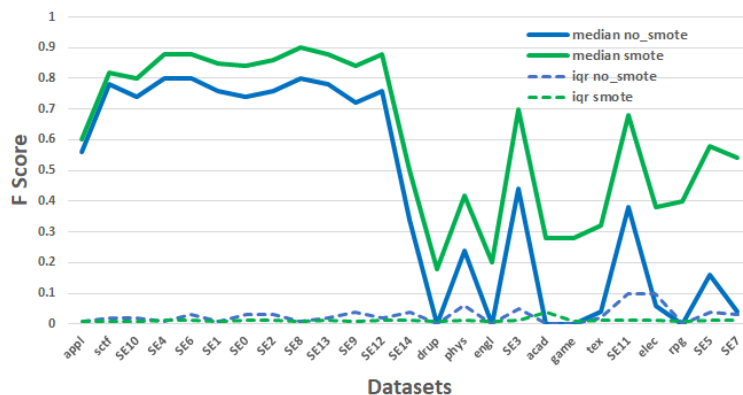


Figure 4: Effects of SMOTEing while text mining 25 SE data sets. In this figure, the data sets on the x-axis are sorted by the size of the different between using and not using SMOTE. Medians and IQRs shown as solid and dashed lines, respectively (recall that the medians are the 50th percentile result and the IQRs are the (75-25)th percentile). From [63].

check— which is the role of statistical tests). Meta-analysis is extremely difficult without dispersion statistics for the results needed to construct confidence intervals [12, 90].

- If exploring only one data set, then repeat the analysis perhaps 20 to 30 times (and then explore the distributions generated this way with a bootstrap procedure i.e., sampling with replacement).
- Temporal validation: given data divided into some time series (e.g., different versions of the same project) then train on past data and test on future data. Our recommendation here is to keep the size of the future test set constant; e.g., train on software versions $K_{-2,-1}$, then test on version K . Such validation enhances realism and is likely to be far more interesting to the intended users of the research.
- Cross-project validation: train on another project data, then test on the project. Note that such cross-project transfer learning can get somewhat complex when the projects use data with different attribute names or features.
- Within-project validation (also called “cross-validation”): divide the project data into N bins, train on $N - 1$ bins, test on the hold out. Repeat for all bins.

Note that, temporal validation (and leave-one-out cross-validation LOOCV) do not produce a range of results since the target is always fixed. However, cross- and within-validations produce distributions of results and so must be analyzed with appropriately so that both measures of central tendency (e.g., mean or median) and dispersion (e.g. variance or IQR) are reported. As a generally accepted rule, in order to obtain, say, 25 samples of cross-project learning, select 90% of the data, at random, 25 times. And to

obtain, say, 25 samples of within-project learning, $M = 5$ times shuffle the order of the data and for each shuffle, and run $N = 5$ cross-validation. Caveat: for small data sets under, say, 60 items, use $M, N = 8, 3$ (since $8 \times 3 \approx 25$).

4.10. Not Tuning

Many machine learners, including data miners, have poorly chosen defaults [108, 45, 37, 105, 1, 2]. Numerous recent results show that the default control settings for data miners are far from optimal. For example, when performing defect prediction, various groups report that tuning can find new settings that dramatically improve the performance of the learned mode [37, 105, 106]. As a specific of that tuning, consider the standard method for computing the distance between rows x and y (where each row contains i variables):

$$d(x, y) = \left(\sum_i ((x_i - y_i)^2) \right)^{1/2}$$

When using SMOTE to re-balance data classes, tuning found it useful to alter the “2” value in the above equation to some other value (often, some number as high as “3”).

One particularly egregious “smell” is to compare a heavily tuned new or pet approach with other benchmarks that are untuned. Although, superficially the new approach may appear far superior what we really learn is using the new technique well can outperform using another technique badly. This kind of finding is not particularly useful.

As to how to tune, we *strongly* recommend against the “grid search” i.e., a set of nested for-loops that try a wide range of settings for all tuneable parameters. This is a slow approach; an exploration of grid search for defect prediction and found it took days to terminate [37]. Not only that, it has been reported that grid search can miss important optimizations [38]. Every grid has “gaps” between each grid division which means that a supposedly rigorous grid search can still miss important configurations [9]. Bergstra and Bengio [9] comment that for most data sets only a few of the tuning parameters really matter – which means that much of the run-time associated with grid search is actually wasted. Worse still, Bergstra and Bengio observed that the important tunings are different for different data sets, a phenomenon that makes grid search a poor choice for configuring software analytics algorithms for new data sets.

Rather than using “grid search”, we recommend very simple optimizers (such as differential evolution [103]) can suffice for finding better tunings. Such optimizers are very easy to code from scratch. They are also readily available in open-source toolkits such as jMETAL⁸ or DEAP⁹.

4.11. Not Exploring Simplicity

One interesting, and humbling, aspect of using Scott-Knott tests is that, often, dozens of treatments can be grouped together into just a few ranks. For example, in the Scott-Knott results of [40], 32 learners were divided into only four groups.

⁸jmetal.sourceforge.net

⁹github.com/DEAP/deap

While not all problems succumb to simple approaches, it is wise to compare a seemingly sophisticated method against some simpler alternative. Often when we code and test the seemingly naïve method, we find it has some redeeming feature that makes us abandon the more complex methods [19, 36, 62]. Moreover, the more parsimonious our models, the less prone to overfitting [43].

This plea, to explore simplicity, is particularly important in the current research climate that is often focused on Deep Learning that can be a very slow method indeed [35]. Deep Learning is inherently better for problems that have highly intricate internal structure [66]. Nevertheless, many SE data sets have very simpler regular structures, so very simple techniques can execute much faster and perform just as well, or better, than Deep Learning. Such faster executions are very useful since they enable easier replication of prior results.

This is not to say that Deep Learning is irrelevant for SE. Rather, it means that any Deep Learning (or for that matter any other complex technique) result should also be compared against some simpler baseline (e.g., [36]) to justify the additional complexity. As an example, Whigham et al. [109] propose a simple benchmark based on automated data transformation and OLS regression as a comparator, the idea being that if the sophisticated method cannot beat the simple method one has to question its practical value.

Three simple methods for exploring simpler options are *scouts*, *stochastic search* and *feature selection*. Holte [47] reports that a very simple learner called IR can be *scout* ahead in a data set and report back if more complex learning will be required. More generally, by first running a very simple algorithm, it is easy to quickly obtain baseline results to glean the overall structure of the problem.

As to stochastic search, for optimization, we have often found that *stochastic searches* (e.g., using differential evolution [103]) performs very well for a wide range of problems [37, 1, 2]. Note that Holte might say that an initial stochastic search of a problem is a *scout* of that problem.

Also, for classification, we have found that *feature selection* will often find most of the attributes are redundant and can be discarded. For example, the feature selection experiments of [76] show that up to 39 of 41 static code attributes can be discarded while preserving the predictive prowess. A similar result with effort estimation, where most of the features can be safely ignored is reported in [20]. This is important since models learned from fewer features are easier to read, understand, critique, and explain to business users. Also, by removing spurious features, there is less chance of researchers reporting a spurious conclusion. Further, after reducing the dimensionality of the data, then any subsequent processing is faster and easier.

Note that many articles perform a very simple linear-time feature selection such as (i) sort by correlation, variance, or entropy; (ii) then remove the worst remaining feature; (iii) build a model from the surviving features; (iv) stop if the new model is worse than the model learned before; (v) otherwise, go back to step (ii). Hall and Holmes report that such greedy searchers can stop prematurely and that better results are obtained by growing in parallel sets of useful features (see their correlation-based feature subset selection algorithm, described in [42]).

Before moving on, we note one paradoxical aspect of simplicity research—it can be very hard to do. Before certifying that some simpler method is better than the most

Table 5: The Ghortra et al. [40] article from ICSE'15 ranked 32 different learners often used in defect prediction using a Scott-Knott procedure. Those ranks fell into four groups. The following table shows a sample of those groups.

RANK	LEARNER	NOTES
1 'best'	RF= random forest	Random forest of entropy-based decision trees.
	LR= Logistic regression	A generalized linear regression model.
2	KNN kth-nearest	Classify a new instance by finding k examples of similar instances. Ghortra et al. suggest $K = 8$.
	NB= Naïve Bayes	Classify a new instance by (a) collecting mean and standard deviations of attributes in old instances of different classes; (b) return the class whose attributes are statistically most similar to the new instance.
3	DT= decision trees	Recursively divide data by selecting attribute splits that reduce the entropy of the class distribution.
4 'worst'	SVM= support vector machines	Map the raw data into a higher-dimensional space where it is easier to distinguish the examples.

complex state-of-the-art alternative, it may be required to test the simpler against the complex. This can lead to the peculiar situation where weeks of CPU are required to evaluate the value of (say) some very simple stochastic method that takes just a few minutes to terminate.

4.12. Not Justifying Choice of Learner

The risk, and “bad smell”, derives from the rapid growth in machine learning techniques and the ability to construct sophisticated ensembles of different learners means that the number of permutations are rapidly becoming vast. It is trivially easy to permute an old learner L into L' and each new learner results in a new article. The “bad smell” then is to do this with no motivation, with no justification a priori as to why we might expect L' to be interesting and worth exploring.

When assessing some new method, researchers need to compare their methods across a range of interesting algorithms. Comparing all new methods to all prior methods is a daunting task (since the space of all prior methods is so very large). Instead, we suggest that most researchers maintain a watching brief on the analytics literature looking for prominent papers that cluster together learners that appear to have similar performance, then draw their comparison set from at least one item of each cluster.

For learners that predict for discrete class learning, one such clustering can be seen in Table 9 of [40]. This is a clustering of 32 learners commonly used for defect prediction. The results clusters into four groups, best, next, next, worst (shown in Table 5). We would recommend:

- Selecting your range of comparison algorithms as one (selected at random) for each group;
- Or selecting your comparison algorithms all from the top group.

For learners that predict for continuous classes, it is not clear what is the canonical set, however, Random Forest regression trees and logistic regression are widely used.

For search-based SE problems, we recommend an experimentation technique called (you+two+next+dumb), defined as

- ‘You’ is your new method;
- ‘Two’ are well-established widely-used methods (often NSGA-II and SPEA2 [89]);
- A ‘next’ generation method e.g. MOEA/D [112], NSGA-III [27];
- and one ‘dumb’ baseline method (random search or SWAY [19]).

5. Discussion

Methodological debates are evidence of a community of researchers rechecking each other’s work, testing each other’s assumptions, and refining each other’s methods. In SE we think there are too few articles that step back from specific issues and explore the more general question of “what is a valid article?”.

To encourage that debate, this article has proposed the idea of a “bad smell” as a means to use surface symptoms to highlight potential underlying problems. To that end, we have identified over a dozen “smells”.

This article has argued that, each year, numerous research articles are rejected for violating the recommendations of this article. Many of those rejections are avoidable — some with very little effort — just by attending to the bad smells listed in this article. This point is particularly important for software practitioners. That group will struggle to get their voice heard in the research literature unless they follow the established norms of SE articles. To our shame, we note that those norms are rarely recorded — a problem that this article is attempting to solve.

We make no claim that our list of bad smells is exhaustive – and this is inevitable. For example, one issue we have not considered here is assessing the implication of using different evaluation criteria; or studying the generality of methods for software analytics. Recent studies [62] have shown that seemingly state-of-the-art techniques for tasks such as transfer learning that were designed for one domain (defect prediction) in fact translate very poorly when applied to other domains (such as code-smell detection, etc.). Nevertheless, the overall aim of this article is to encourage more debate on what constitutes a “valid” article, recognizing that validity is not a black and white concept (hence the quotation marks). We hope that further debate, across the broader research community, significantly matures this list.

We think that such a debate could be enabled as follows:

1. Read more, reflect more. Study the methods deployed in software analytics and beyond. When reading a article, consider not just the content of the argument, but also its form.
2. Promote and share “bad smells” and anti-patterns as a means of sharing poor practice but with the goal of promoting and good practice.
3. Pre-register studies and identify the smallest effect size of (practical) interest.

4. Analyze the likely power of a study in advance. If too low revise the study design (e.g., collect more data or perform a within-type analysis) or change the research.
5. Strive toward reproducible studies results.
6. Develop, agree and use community standards for software analytics studies.

Further to the last point, as we mentioned above, this paper is part of a strategic plan to develop community standards in software analytics. To that end, we hope it inspires:

- More extensive discussions about best and worst paper writing practices;
- A workshop series where researchers meet to debate and publicise our community's views on what constitutes a "good" paper;
- Subsequent articles that extend and improve the guidance offered above.

Acknowledgments

We extremely grateful to the editor and referees for their detailed and constructive comments.

References

References

- [1] A. Agrawal, W. Fu, T. Menzies, What is Wrong with Topic Modeling? (and How to Fix it Using Search-based SE), CoRR abs/1608.08176, URL <http://arxiv.org/abs/1608.08176>.
- [2] A. Agrawal, T. Menzies, Is "better data" better than "better data miners"? On the benefits of tuning SMOTE for defect prediction, in: Proceedings of the 40th International Conference on Software Engineering, ACM, 1050–1061, 2018.
- [3] A. Agrawal, A. Rahman, R. Krishna, A. Sobran, T. Menzies, We Don't Need Another Hero? The Impact of "Heroes" on Software Development, CoRR abs/1710.09055, URL <http://arxiv.org/abs/1710.09055>.
- [4] A. Arcuri, L. Briand, A Practical Guide for Using Statistical Tests to Assess Randomized Algorithms in Software Engineering, in: Proceedings of the 33rd International Conference on Software Engineering, ICSE '11, ACM, New York, NY, USA, ISBN 978-1-4503-0445-0, 1–10, 2011.
- [5] K. Beck, M. Fowler, G. Beck, Bad smells in code, Refactoring: Improving the design of existing code (1999) 75–88.
- [6] A. Begel, T. Zimmermann, Analyze this! 145 questions for data scientists in software engineering, in: Proceedings of the 36th ACM International Conference on Software Engineering, ACM, 12–23, 2014.

- [7] R. Bender, S. Lange, Adjusting for multiple testing—when and how?, *Journal of Clinical Epidemiology* 54 (4) (2001) 343–349.
- [8] Y. Benjamini, D. Yekutieli, The Control of the False Discovery Rate in Multiple Testing under Dependency, *The Annals of Statistics* 29 (4) (2001) 1165–1188.
- [9] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, *Journal of Machine Learning Research* 13 (Feb) (2012) 281–305.
- [10] C. Blair, J. Higgins, Comparison of the power of the paired samples t test to that of Wilcoxon’s signed-ranks test under various population shapes., *Psychological Bulletin* 97 (1) (1985) 119–128.
- [11] A. Booth, A. O’Rourke, The value of structured abstracts in information retrieval from MEDLINE, *Health libraries review* 14 (3) (1997) 157–166.
- [12] M. Borenstein, L. Hedges, J. Higgins, H. Rothstein, *Introduction to Meta-Analysis*, Wiley, Chichester, West Sussex, UK, 2009.
- [13] M. Bosu, S. MacDonell, Data quality in empirical software engineering: a targeted review, in: *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, ACM, 171–176, 2013.
- [14] D. Budgen, A. Burn, B. Kitchenham, Reporting computing projects through structured abstracts: a quasi-experiment, *Empirical Software Engineering* 16 (2) (2011) 244–277.
- [15] K. Button, J. Ioannidis, C. Mokrysz, B. Nosek, J. Flint, E. Robinson, M. Munafò, Power failure: why small sample size undermines the reliability of neuroscience, *Nature Reviews Neuroscience* 14 (5) (2013) 365.
- [16] J. Carpenter, J. Bithell, Bootstrap confidence intervals: when, which, what? A practical guide for medical statisticians, *Statistics in Medicine* 19 (9) (2000) 1141–1164.
- [17] R. P. Carver, The case against statistical significance testing, revisited, *The Journal of Experimental Education* 61 (4) (1993) 287–292.
- [18] G. Cawley, N. Talbot, On over-fitting in model selection and subsequent selection bias in performance evaluation, *Journal of Machine Learning Research* 11 (7) (2010) 2079–2107.
- [19] J. Chen, V. Nair, R. Krishna, T. Menzies, " Sampling" as a Baseline Optimizer for Search-based Software Engineering, *IEEE Transactions on Software Engineering* .
- [20] Z. Chen, T. Menzies, D. Port, D. Boehm, Finding the right data for software cost modeling, *IEEE software* 22 (6) (2005) 38–46.
- [21] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*, Lawrence Erlbaum Associates, Hillsdale, NJ, 2nd edn., 1988.

- [22] J. Cohen, A power primer, *Psychological Bulletin* 112 (1) (1992) 155–159.
- [23] D. Colquhoun, An investigation of the false discovery rate and the misinterpretation of p-values, *Royal Society Open Science* 1 (140216).
- [24] R. Courtney, D. Gustafson, Shotgun correlations in software measures, *Software Engineering Journal* 8 (1) (1993) 5–13.
- [25] D. S. Cruzes, T. Dybå, Research synthesis in software engineering: A tertiary study, *Information and Software Technology* 53 (5) (2011) 440–455.
- [26] R. De Veaux, D. Hand, How to Lie with Bad Data, *Statistical Science* 20 (3) (2005) 231—238.
- [27] K. Deb, H. Jain, An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints., *IEEE Trans. Evolutionary Computation* 18 (4) (2014) 577–601.
- [28] P. Devanbu, T. Zimmermann, C. Bird, Belief & evidence in empirical software engineering, in: *IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, IEEE, 108–119, 2016.
- [29] T. Dybå, T. Dingsøy, Strength of evidence in systematic reviews in software engineering, in: *Proceedings of the 2nd ACM-IEEE international Symposium on Empirical Software Engineering and Measurement*, ACM, 178–187, 2008.
- [30] T. Dybå, V. Kampenes, D. Sjøberg, A systematic review of statistical power in software engineering experiments, *Information and Software Technology* 48 (8) (2006) 745–755.
- [31] B. Earp, D. Trafimow, Replication, falsification, and the crisis of confidence in social psychology, *Frontiers in Psychology* 6 (2015) 621–6322.
- [32] B. Efron, R. J. Tibshirani, *An Introduction to the Bootstrap*, *Mono. Stat. Appl. Probab.*, Chapman and Hall, London, 1993.
- [33] P. Ellis, *The Essential Guide to Effect Sizes: Statistical Power, Meta-Analysis, and the Interpretation of Research Results*, Cambridge University Press, 2010.
- [34] D. Erceg-Hurn, V. Mirosevich, Modern robust statistical methods: an easy way to maximize the accuracy and power of your research., *American Psychologist* 63 (7) (2008) 591–601.
- [35] W. Fu, T. Menzies, Easy over hard: A case study on deep learning, in: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ACM, 49–60, 2017.
- [36] W. Fu, T. Menzies, Revisiting unsupervised learning for defect prediction, in: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ACM, 72–83, 2017.

- [37] W. Fu, T. Menzies, X. Shen, Tuning for Software Analytics, *Inf. Softw. Technol.* 76 (C) (2016) 135–146, ISSN 0950-5849.
- [38] W. Fu, V. Nair, T. Menzies, Why is Differential Evolution Better than Grid Search for Tuning Defect Predictors?, *CoRR* abs/1609.02613, URL <http://arxiv.org/abs/1609.02613>.
- [39] A. Gelman, J. Carlin, H. Stern, D. Dunson, A. Vehtari, D. Rubin, *Bayesian data analysis*, CRC, 3rd edn., 2013.
- [40] B. Ghotra, S. McIntosh, A. E. Hassan, Revisiting the impact of classification techniques on the performance of defect prediction models, in: *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, IEEE Press, 789–800, 2015.
- [41] S. Goodman, D. Fanelli, J. Ioannidis, What does research reproducibility mean?, *Science Translational Medicine* 8 (341) (2016) 341ps12–341ps12.
- [42] M. A. Hall, G. Holmes, Benchmarking attribute selection techniques for discrete class data mining, *IEEE Transactions on Knowledge and Data Engineering* 15 (6) (2003) 1437–1447.
- [43] D. Hawkins, The problem of overfitting, *Journal of Chemical Information and Computer Sciences* 44 (1) (2004) 1–12.
- [44] K. Healy, *Data Visualization: A Practical Introduction*, Princeton University Press, 2018.
- [45] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, S. Babu, Starfish: A Self-tuning System for Big Data Analytics, in: *Cidr*, vol. 11, 261–272, 2011.
- [46] D. Hoaglin, F. Mosteller, J. Tukey, *Understanding Robust and Exploratory Data Analysis*, John Wiley and Sons, 1983.
- [47] R. Holte, Very Simple Classification Rules Perform Well on Most Commonly Used Datasets, *Machine Learning* 11 (1) (1993) 63–90, ISSN 1573-0565.
- [48] Q. Huang, S. Ritchie, M. Brozynska, M. Inouye, Power, false discovery rate and Winner’s Curse in eQTL studies, *bioRxiv* (2017) 209171.
- [49] Q. Huang, X. Xia, D. Lo, Supervised vs unsupervised models: A holistic look at effort-aware just-in-time defect prediction, in: *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, 159–170, 2017.
- [50] J. Ioannidis, Why most published research findings are false, *PLoS Medicine* 2 (8) (2005) e124–6.
- [51] M. Ivarsson, T. Gorschek, A method for evaluating rigor and industrial relevance of technology evaluations, *Empirical Software Engineering* 16 (3) (2011) 365–395.

- [52] P. Johnson, M. Ekstedt, I. Jacobson, Where's the theory for software engineering?, *IEEE software* 29 (5) (2012) 96–96.
- [53] M. Jørgensen, T. Dybå, K. Liestøl, D. Sjøberg, Incorrect Results in Software Engineering Experiments: How to Improve Research Practices, *J. of Syst. & Softw.* 116 (2016) 133–145.
- [54] V. Kampenes, T. Dybå, J. Hannay, D. Sjøberg, A systematic review of effect size in software engineering experiments, *Information and Software Technology* 49 (11-12) (2007) 1073–1086.
- [55] B. Kitchenham, D. Budgen, P. Brereton, *Evidence-Based Software engineering and systematic reviews*, CRC Press, Boca Raton, FL, US, 2015.
- [56] B. Kitchenham, T. Dybå, M. Jørgensen, Evidence-based Software Engineering, in: *26th IEEE International Conference on Software Engineering (ICSE 2004)*, IEEE Computer Society, 273–281, 2004.
- [57] B. Kitchenham, L. Madeyski, D. Budgen, J. Keung, P. Brereton, S. Charters, S. Gibbs, A. Pohthong, Robust statistical methods for empirical software engineering, *Empirical Software Engineering* 22 (2) (2017) 579–630.
- [58] B. Kitchenham, S. Pfleeger, L. Pickard, P. Jones, D. Hoaglin, K. El Emam, J. Rosenberg, Preliminary guidelines for empirical research in software engineering, *IEEE Transactions on Software Engineering* 28 (8) (2002) 721–734.
- [59] B. A. Kitchenham, O. P. Brereton, S. Owen, J. Butcher, C. Jefferies, Length and readability of structured software engineering abstracts, *IET Software* 2 (1) (2008) 37–45.
- [60] A. J. Ko, T. D. Latoza, M. M. Burnett, A practical guide to controlled experiments of software engineering tools with human participants, *Empirical Software Engineering* 20 (1) (2015) 110–141.
- [61] R. Krishna, A. Agrawal, A. Rahman, A. Sobran, T. Menzies, What is the Connection Between Issues, Bugs, and Enhancements? (Lessons Learned from 800+ Software Projects), *CoRR* abs/1710.08736, URL <http://arxiv.org/abs/1710.08736>.
- [62] R. Krishna, T. Menzies, Simpler Transfer Learning (Using "Bellwethers"), *arXiv* abs/1703.06218, URL <http://arxiv.org/abs/1703.06218>.
- [63] R. Krishna, Z. Yu, A. Agrawal, M. Dominguez, D. Wolf, The 'BigSE' Project: Lessons Learned from Validating Industrial Text Mining, *2016 IEEE/ACM 2nd International Workshop on Big Data Software Engineering (BIGDSE)* (2016) 65–71.
- [64] D. Lakens, Equivalence tests: a practical primer for t tests, correlations, and meta-analyses, *Social Psychological and Personality Science* 8 (4) (2017) 355–362.

- [65] D. Lakens, E. Evers, Sailing from the seas of chaos into the corridor of stability: Practical recommendations to increase the informational value of studies, *Perspectives on Psychological Science* 9 (3) (2014) 278–292.
- [66] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, *nature* 521 (7553) (2015) 436.
- [67] S. Lessmann, B. Baesens, C. Mues, S. Pietsch, Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings, *IEEE Transactions on Software Engineering* 34 (4) (2008) 485–496.
- [68] G. Liebchen, M. Shepperd, Data Sets and Data Quality in Software Engineering, in: *PROMISE 2008*, ACM Press, 2008.
- [69] G. Liebchen, M. Shepperd, Data sets and data quality in software engineering: Eight years on, in: *Proceedings of the 12th International Conference on Predictive Models and Data Analytics in Software Engineering*, ACM, 2016.
- [70] D. Lo, N. Nagappan, T. Zimmermann, How practitioners perceive the relevance of software engineering research, in: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ACM, 415–425, 2015.
- [71] E. Loken, A. Gelman, Measurement error and the replication crisis, *Science* 355 (6325) (2017) 584–585.
- [72] L. Madeyski, B. Kitchenham, Would Wider Adoption of Reproducible Research be Beneficial for Empirical Software Engineering Research?, *Journal of Intelligent & Fuzzy Systems* 32 (2017) 1509–1521.
- [73] B. Manly, *Randomization, Bootstrap and Monte Carlo Methods in Biology*, Chapman & Hall, London, 1997.
- [74] S. Maxwell, K. Kelley, J. Rausch, Sample size planning for statistical power and accuracy in parameter estimation, *Annual Review of Psychology* 59 (2008) 537–563.
- [75] G. McClelland, Increasing statistical power without increasing sample size., *American Psychologist* 55 (8) (2000) 963–964.
- [76] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, *IEEE Transactions on Software Engineering* 33 (1) (2007) 2–13.
- [77] T. Menzies, M. Shepperd, Editorial: Special issue on repeatable results in software engineering prediction, 2012.
- [78] N. Mittas, L. Angelis, Ranking and clustering software cost estimation models through a multiple comparisons algorithm, *IEEE Transactions on Software Engineering* 39 (4) (2013) 537–551.

- [79] M. Munafò, B. Nosek, D. Bishop, K. Button, C. Chambers, N. du Sert, U. Simonsohn, E. Wagenmakers, J. Ware, J. Ioannidis, A manifesto for reproducible science, *Nature Human Behaviour* 1 (2017) 0021.
- [80] G. Myatt, W. Johnson, *Making sense of data II: A practical guide to data visualization, advanced data mining methods, and applications*, John Wiley & Sons, 2009.
- [81] V. Nair, A. Agrawal, J. Chen, W. Fu, G. Mathew, T. Menzies, L. Minku, M. Wagner, Z. Yu, *Data-Driven Search-based Software Engineering*, in: *MSR'18*, 2018.
- [82] R. Nickerson, Confirmation bias: A ubiquitous phenomenon in many guises., *Review of General Psychology* 2 (2) (1998) 175–220.
- [83] . Open Science Collaboration, Estimating the reproducibility of psychological science, *Science* 349 (6251) (2015) aac4716–3.
- [84] K. Petersen, S. Vakkalanka, L. Kuzniarz, Guidelines for conducting systematic mapping studies in software engineering: An update, *Information and Software Technology* 64 (2015) 1–18.
- [85] M. Rosli, E. Tempero, A. Luxton-Reilly, Can we trust our results? a mapping study on data quality, in: *20th IEEE Asia-Pacific Software Engineering Conference (APSEC'13)*, 116–123, 2013.
- [86] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, *Empirical Software Engineering* 2 (14) (2009) 131–164.
- [87] A. Saltelli, S. Tarantola, F. Campolongo, Sensitivity Analysis as an Ingredient of Modeling, *Statistical Science* 15 (4) (2000) 377–395.
- [88] A. Sarkar, J. Guo, N. Siegmund, S. Apel, K. Czarnecki, Cost-efficient sampling for performance prediction of configurable systems (t), in: *Automated Software Engineering (ASE)*, 2015 30th IEEE/ACM International Conference on, IEEE, 342–352, 2015.
- [89] A. S. Sayyad, H. Ammar, Pareto-optimal search-based software engineering (POSBSE): A literature survey, in: *2nd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*, IEEE, 21–27, 2013.
- [90] F. Schmidt, J. Hunter, *Methods of Meta-Analysis: Correcting Error and Bias in Research Findings*, Sage Publications, 3rd edn., 2014.
- [91] W. Shadish, T. Cook, D. Campbell, *Experimental and quasi-experimental designs for generalized causal inference*, Houghton Mifflin, Boston, 2002.
- [92] M. Shaw, Writing good software engineering research papers, in: *25th IEEE International Conference on Software Engineering*, IEEE Computer Society, 726–736, 2003.

- [93] M. Shepperd, D. Bowes, T. Hall, Researcher bias: The use of machine learning in software defect prediction, *IEEE Transactions on Software Engineering* 40 (6) (2014) 603–616.
- [94] M. Shepperd, Q. Song, Z. Sun, C. Mair, Data quality: Some comments on the NASA software defect datasets, *IEEE Transactions on Software Engineering* 39 (9) (2013) 1208–1215.
- [95] R. Silberzahn, E. Uhlmann, D. Martin, P. Anselmi, F. Aust, E. Awtrey, Š. Bahnik, F. Bai, C. Bannard, E. Bonnier, et al., Many analysts, one dataset: Making transparent how variations in analytical choices affect results URL <https://osf.io/j5v8f>.
- [96] J. Simmons, L. Nelson, U. Simonsohn, False-positive psychology: Undisclosed flexibility in data collection and analysis allows presenting anything as significant, *Psychological science* 22 (11) (2011) 1359–1366.
- [97] D. Sjøberg, T. Dybå, B. Anda, J. Hannay, Building theories in software engineering, in: *Guide to advanced empirical software engineering*, Springer, 312–336, 2008.
- [98] P. Smaldino, R. McElreath, The natural selection of bad science, *Royal Society Open Science* 3 (9) (2016) 160384.
- [99] J. Snoek, H. Larochelle, R. Adams, Practical Bayesian Optimization of Machine Learning Algorithms, in: *Advances in Neural Information Processing Systems (NIPS 2012)*, 2951–2959, 2012.
- [100] J. Spence, D. Stanley, Prediction Interval: What to Expect When You’re Expecting . . . A Replication, *PLoS ONE* 11 (9) (2016) e0162874.
- [101] K.-J. Stol, B. Fitzgerald, Uncovering theories in software engineering, in: *2nd SEMAT Workshop on a General Theory of Software Engineering (GTSE)*, IEEE, 5–14, 2013.
- [102] K.-J. Stol, P. Ralph, B. Fitzgerald, Grounded theory in software engineering research: a critical review and guidelines, in: *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*, IEEE, 120–131, 2016.
- [103] R. Storn, K. Price, Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces, *Journal of Global Optimization* 11 (4) (1997) 341–359, ISSN 1573-2916.
- [104] D. Szucs, J. Ioannidis, Empirical assessment of published effect sizes and power in the recent cognitive neuroscience and psychology literature, *PLoS Biology* 15 (3) (2017) e2000797.
- [105] C. Tantithamthavorn, S. McIntosh, A. Hassan, K. Matsumoto, Automated parameter optimization of classification techniques for defect prediction models, in: *IEEE/ACM 38th International Conference on Software Engineering (ICSE 2016)*, IEEE, 321–332, 2016.

- [106] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, K. Matsumoto, The Impact of Automated Parameter Optimization on Defect Prediction Models, *IEEE Transactions on Software Engineering* .
- [107] C. Theisen, M. Dunaiski, L. Williams, W. Visser, Writing good software engineering research papers: revisited, in: *IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, IEEE, 402–402, 2017.
- [108] D. Van Aken, A. Pavlo, G. Gordon, B. Zhang, Automatic database management system tuning through large-scale machine learning, in: *Proceedings of the 2017 ACM International Conference on Management of Data*, ACM, 1009–1024, 2017.
- [109] P. Whigham, C. Owen, S. Macdonell, A Baseline Model for Software Effort Estimation, *ACM Transactions on Software Engineering and Methodology* 24 (3).
- [110] R. Wilcox, *Introduction to Robust Estimation and Hypothesis Testing*, Academic Press, 3rd edn., 2012.
- [111] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, A. Wesslén, *Experimentation in software engineering*, Springer Science & Business Media, 2nd edn., 2012.
- [112] Q. Zhang, H. Li, MOEA/D: A multiobjective evolutionary algorithm based on decomposition, *IEEE Transactions on Evolutionary Computation* 11 (6) (2007) 712–731.
- [113] D. Zimmerman, Invalidation of parametric and nonparametric statistical tests by concurrent violation of two assumptions, *The Journal of Experimental Education* 67 (1) (1998) 55–68.
- [114] T. Zimmermann, P. Weisgerber, S. Diehl, A. Zeller, Mining Version Histories to Guide Software Changes, in: *Proceedings of the 26th International Conference on Software Engineering, ICSE '04*, IEEE Computer Society, Washington, DC, USA, ISBN 0-7695-2163-0, 563–572, URL <http://dl.acm.org/citation.cfm?id=998675.999460>, 2004.