Generalization and Theory-Building in Software Engineering Research

Magne Jørgensen, Dag Sjøberg
Simula Research Laboratory
{magne.jorgensen, dagsj}@simula.no

Abstract

The main purpose of this paper is to generate discussions which may improve how we conduct empirical software engineering studies. Our position is that statistical hypothesis testing plays a too large role in empirical software engineering studies. The problems of applying statistical hypothesis testing in empirical software engineering studies is illustrated by the finding: Only 3 out of the 47 studies in Journal of Empirical Software Engineering which applied statistical hypothesis testing, were able to base their statistical testing on well-defined populations and random samples from those populations. The frequent use of statistical hypothesis testing may also have had unwanted consequences on the study designs, e.g., it may have contributed to a too low focus on theory building. We outline several steps we believe are useful for a change in focus from "generalizing from a random sample to a larger population" to "generalizing across populations through theory-building".

1. Introduction

There are several methods of generalization in science and generalization based on statistical hypothesis testing, i.e., generalization from a randomly drawn sample to a larger population, is only one of them. In fact, it is only the last 50 years that generalization through hypothesis testing has become in common use. Before that, other types of generalization of scientific results were dominating. In [1, p269] it is reported that statistical hypothesis testing was practically nonexistent in psychology studies before 1940. However, already in 1955 more than 80% of the empirical articles in four leading psychology journals used statistical hypothesis testing. The proportion of empirical software engineering studies applying statistical hypothesis testing is high, as well. An examination of empirical studies[1] *(N=72)* in the 1996-2003 volumes of Journal of Empirical Software Engineering gave that about *65%* of the studies applied statistical hypothesis testing of some type, e.g., test of significance of difference in mean values of alternative models or test of significance of a variable included in a regression model. There is consequently empirical evidence supporting the claim that statistical hypothesis testing is a frequently used ingredient in empirical software engineering studies.

In this paper we claim that typical software engineering contexts may not enable meaningful use of statistical hypothesis testing and that the frequent use of statistical hypothesis testing may have had unwanted impacts on how software engineering researchers design their studies. We argue, amongst others, that the application of statistical hypothesis testing easily leads to formulation of non-optimal research questions and study designs.

The theory behind statistical hypothesis testing is quite complex and there is no unified interpretation of all important concepts. There are, for example, different opinions on how to interpret the basic concept of probability [2-4]. In addition, there are many deep philosophical discussions on important assumptions, e.g., the problem of induction [2, 3]. The authors of this paper are software engineering researchers, i.e., users of statistics, rather than researchers on statistics. This means that there may be inaccurate formulations, biases, and, even incorrect claims in this paper. We believe, nevertheless, that the main claims of this paper are correct and that the topics discussed are worth discussing.

The paper can be read as a criticism of most empirical software engineering studies, including studies conducted by the authors of this paper. To some extent it is meant to be exactly that. It is, however, important to notice that what we criticize is mainly one element of the studies, i.e., the use of statistical hypothesis testing without having a well-defined population, and not the usefulness

[1] We excluded case-studies of only one project or system from the set of empirical studies, because it is obvious that studies of single cases do not enable use of statistical hypothesis testing. There were, however, not many studies of single cases. Which studies to include and exclude in an examination of frequency of use of statistical hypothesis testing may be subject to discussions. The main point in this study is that statistical hypothesis testing is frequently applied, not whether this proportion is 65% or 50%.

of the studies themselves. We found, for example, in our review of papers in Journal of Empirical Software Engineering several empirical software engineering papers that, in addition to statistical hypothesis testing (based on generalization from sample to population), argued convincingly that their results could be generalized *across* the studied population/sample, e.g., from computer science students to software professionals. These arguments were based on generalization by similarity (representativeness) of their sample to samples of other populations, rather than on generalization from randomly drawn samples to larger populations.

The remaining part of this paper is organized as follows: Section 2 points at what we believe is one of the most important problems when applying statistical hypothesis testing in a software engineering context, i.e., the failure of establishing a well-defined population and consequently the failure to provide a random sampling process. Section 3 discusses two biases potentially resulting from the frequent use of statistical hypothesis testing in empirical software engineering: a) A bias towards formulation of fruitless (shallow) research problems to fit the format induced by statistical hypothesis testing. b) A lack of focus on the use of structured techniques for generalization across population. Section 4 provides a preliminary outline of how the authors believe a larger proportion of empirical software engineering research should be conducted to enable theory-building and generalization across populations. Section 5 concludes the paper.

2. Well-defined Populations?

The importance of a well-defined population from which the sample is drawn can hardly be over-emphasized when one applies statistical hypothesis testing. For example, in "Preliminary Guidelines for Empirical Research in Software Engineering" [5] it is claimed that: "*If you cannot define the population from which your subjects/objects are drawn, it is not possible to draw any inference from the results of your experiment"*[2]. A well-defined population is essential, because the statistical hypothesis testing methods are based on generalization from a sample to the population from which the sample was randomly drawn. When there is no well-defined population, then the measures, e.g., the p-values, do not have the intended probability-based meanings. In addition, if there is no well-defined population, it is impossible to design a procedure that ensures a random selection of subjects/objects. In other words, statistical hypothesis testing is of little use if we cannot define our population.

Unfortunately, empirical studies of software engineering may seldom have well-defined populations. It is not even clear what the elements of our population should be! A typical software engineering study may need to include the elements: Subjects (e.g., software developers), problems to be solved (e.g., development tasks), and, problem solving context (e.g., development tools). The importance of including subjects, problems and contexts means that we may have to define populations that include all these three elements. While this in principle may be possible, we have not seen a single study that has managed this other than for very narrow sub-populations, e.g., for the population of developers within a specific company and a specific development context. Now, some will argue, this is mainly a matter of lack of statistical skill and immaturity of our domain (we are a young research community compared with many other communities). We believe that this is only one reason for the problem. The main problem, we believe, is that the variation and dynamics of the software engineering domain are so large that the definition of a population, that meet the requirements set by statistical hypothesis testing, can be extremely difficult, and maybe not worth the effort. We, consequently, may have to apply other means than statistical hypothesis testing to generalize from experiments and other types of empirical software engineering studies.

Some of the problems concerning well-defined populations may be smaller in other domains with frequent use of statistical hypothesis testing, e.g., agriculture, quality assurance of industrial manufacturing processes, and, medicine. In these domains the problems to be solved and the problem solving context may have less variation and less impact on the effect of treatment compared with software engineering (we admit, however, not being experts in these fields). For example, in medicine it may sometimes be meaningful to define the population to be "the people in the world with disease X". For disease X there may be good reasons to believe that the nationality, profession and intelligence of the patient are of minor importance for the effect of a treatment. In some cases it may therefore be meaningful to claim that a random sample of patients from a particular hospital is a good substitute for a random sample of the total population of people with disease X. Similar simplifications regarding sample and population may frequently not be possible in software engineering studies.

We do, however, not make the claim that other disciplines have no problems with defining the population. On the contrary, a review of clinical cancer

[2] This claim should, as we understand it, not be interpreted outside the context of statistical hypothesis testing. Obviously, even a study without a well-defined population (but with a well-defined sample) may enable the researcher to infer about similar projects, e.g., based on argumentation by analogy or by theory.

studies in Norway [6] report that only 43 % of the studies "*presented inclusion and criteria which may give the reader an idea of the population intended to be studied*".

The statisticians Fisher and Neyman developed statistical hypothesis testing for agriculture (Fisher) and quality control in manufacturing processes (Neyman), see [4]. In other words, statistical hypothesis testing was in the beginning not intended to be a universal tool for all types of domains. According to [3, p. 226], Fisher meant that "*We have a large number of inductive techniques, and we should consider practical situations carefully to see which is appropriate*".

Medicine may have had a strong impact on how software engineering researchers use statistical hypothesis testing. An example supporting this claim, is that 7 out of the 9 main sources used as input to the "Preliminary guidelines for empirical research in software engineering" [5], were from medicine. We have argued that there may be important differences between medicine and software engineering regarding the possibilities to have well-defined populations from which we can draw random samples. Maybe empirical software engineering has been strongly impacted by the research practice in a discipline where statistical hypothesis testing has been successful, without being sufficiently aware of important differences between the domains.

To illustrate the problems of population in empirical software engineering, we further examined the 47 empirical studies in Journal of Empirical Software Engineering that applied statistical hypothesis testing. We found that only three of those studies did provide a well-defined population and conducted a random sample from that population[3]! This is not a satisfactory situation for empirical software engineering.

All these three studies achieved a well-defined population on the cost of narrowing the population quite a lot, i.e., narrowing the population to software companies within one country or to software companies of several countries one particular year. These studies show, nevertheless, that statistical hypothesis testing may be meaningful in some software engineering contexts.

Interestingly, many of the studies we examined described their sample reasonable well and were, when discussing the results, concerned about how *representative* their samples were, e.g., whether the task to be solved was representative of a set of realistic industrial tasks. There are similarities between "representativeness" and "generalization to a larger population by random sample", but these concepts do nevertheless require different analysis methods and argumentations. The need to discuss representativeness is no surprise, given that the researchers knew that they had no well-defined population, no random sample, and that the sample was too narrow to be of much interest of itself. In fact, we believe that representativeness-discussions should be more emphasized, but also more related to theory. We discuss this need for theories in more depth in Section 4.

According to [5] and several text-books on statistical hypothesis testing, e.g., [7] the lack of well-defined populations and random samples means that no generalization, based on statistical principles, to a larger population is possible. In other words, although statistical hypothesis testing is in frequent use in empirical software engineering studies, basic assumptions for meaningful use are met in only very few of these studies!

3. Biases Potentially Resulting From Statistical Hypothesis Testing

There may be unwanted study design biases caused by the frequent use of statistical hypothesis testing. In this section we discuss the following two:

- *Bias 1*: Use of statistical hypothesis testing may easily lead to formulation and testing of shallow, non-connected, co-variation based hypotheses instead of formulation and testing of cause-effect theories. A potential reason for this is that traditional (non-Bayesian) statistical hypothesis testing is developed for situations with little understanding of underlying processes and much data [3], i.e., it is not designed for the purpose of integrating previous knowledge in the statistical testing of hypothesis.
- *Bias 2*: Reliance on statistical hypothesis testing as the main generalization technique may easily lead to lack of focus on other types of generalization techniques, i.e., techniques used to generalize across populations. As described in Section 2, we observed that many researchers try to generalize across populations applying non-statistical techniques, e.g., argumentation based on representativeness. These researchers seemed to lack method support for doing this applying other techniques than informal argumentation.

These biases are not *necessary* consequences of statistical hypothesis testing, but we nevertheless believe that the frequent use of statistical hypothesis testing has contributed to them. This belief is mainly based on personal experience from our own empirical software engineering studies, e.g., how we had to change the hypotheses to fit the statistical hypothesis testing framework and the lack of stimuli to build theories.

[3] This evaluation may be to some extent subjective. However, in most studies there was no description at all of population or sampling process (only of the sample itself), i.e., it is hard to see that other researchers would reach substantially different evaluations.

Similar observations, regarding shallow hypotheses in code inspection studies, is described in [5].

The following example, based on several real empirical software engineering studies, among them some of our own, may illustrate the two biases described above:

Example: The overall goal of software cost estimation research should be to improve cost estimation practice. It is obvious from previous research, see for example [8], that there no single best estimation method independent of data set and context. This means that it is important to know which cost estimation methods work in which situations, i.e., we want a theory for selection of cost estimation methods.

A frequently applied hypothesis testing-based design of empirical studies with the goal of comparing estimation methods is similar to this:

1. Decide which cost estimation methods to compare, e.g., estimation methods *A* and *B*.
2. Decide on the hypotheses to test. Typically, a hypothesis is of the type "The mean estimation accuracy of *A* is better than that of *B*" or "There are no difference in mean estimation accuracy between *A* and *B*". (*Notice that on this stage we have already rewritten the goal of the research, i.e., instead of formulating research questions of great relevance, such as when are A better than B, we have adjusted our hypotheses to fit the usual mean or median value comparison format of statistical hypothesis testing.*)
3. Collect a set of projects with known actual effort and other important project variable values. (*Typically, this set of projects is far from randomly drawn and the total population of projects from which the sample is drawn is not defined.*)
4. Calibrate/tailor the estimation methods to the project context based on a training set, i.e., a sub-set of the data set collected in Step 3.
5. Compare the mean estimation accuracy of the estimation methods on the evaluation set of projects, i.e., the total set of projects subtracted the learning set.
6. Apply a statistical test, e.g., a t-test, to find the p-value potentially rejecting the hypothesis, i.e., calculate the probability that the sample value would be as large as the value actually observed if the hypothesis was true. *(Notice that there typically is no natural level of p-value that should lead to rejection of the hypothesis. This value is, in nearly all cases we have observed, arbitrarily set. Historically, significance levels of 0.05 and 0.1 have been applied because the statistical values related to them could be found in tables, and applying other values would require time-consuming calculations. Significance levels of 0.05 and 0.1 are nevertheless still in common use when testing statistical hypotheses. In some situations, e.g., situations where we have to choose between two alternatives and have no other information than the collected data, it is our opinion that we should accept p-values much higher than 0.1 sufficient to reject a hypothesis of no difference between the two alternatives.)*

Assume that we found that the observed data is highly unlikely given the correctness of the hypothesis "The mean estimation accuracy of *A* is better than *B*", with a p-value of 0.03. Based on this we therefore reject the hypothesis.

What does this rejection of hypothesis and the p-value mean? Are we able to apply the rejected hypothesis result to some relevant population? Unfortunately, this is not the case. The lack of a well-defined population from which we randomly selected the projects means that we cannot use statistical methods to generalize to relevant populations, e.g., we cannot use the statistical hypothesis testing as a means to generalize to future projects in other companies. The lack of a definition of a larger population means that we can only make statistical claims about the sample we studied. To make claims about the studied sample, however, we do not need sophisticated inference techniques, such as the statistical t-test of difference in mean values. We could simply look at the mean or median accuracy values of *A* and *B* together with the variance.

4. Are There Better Approaches?

As always, it is easier to see problems with existing empirical studies than to devise better approaches. The topic of design for empirical research is a very complex topic with many practical and principal challenges. In spite of this complexity, we will try to outline a general approach for empirical studies on software engineering where statistical hypothesis testing is only one out of many techniques that may be used.

The approach we propose is based on the following beliefs:

- Statistical hypothesis testing is mainly designed for situations were we do not understand much about what is happening, and, were we mainly are interested in co-variation, i.e., it is a research method mainly developed to see "if something work" rather than "why something works". A support of this belief is provided in [3].
- Generalization across populations, e.g., to future software projects and contexts, is what we typically need in the domain of software engineering.
- Generalization from one software engineering study context to other contexts, i.e., across populations, happens only through theory. It is the interplay

between theory and study design that determines our ability to generalize, i.e., there is no best study design independent of the aspect of theory we want to test or improve.

- A theory based on cause-effect is in most cases a better theory than a theory based on co-variation. (*Philosophically, one may argue that it is not meaningful to talk about cause-effect relationships, only different levels of co-variation. However, for practical purposes most people may agree that there is a meaningful difference and that we should strive for cause-effect theories for better predictions and understanding of behavior in other contexts than the one examined in a particular software engineering study.*)
- There is nearly always some relevant knowledge available from the previous software engineering research or from research in other domains that enables meaningful theories to be developed. There are consequently few good excuses for conducting purely exploratory studies and isolated studies applying statistical hypothesis testing. Most studies should start with a thorough review of relevant knowledge. (*The only good reason we can think of to conduct more exploratory studies and isolated hypothesis testing is for educational purposes, e.g., Ph.D. students learning how to conduct experiments.*)
- A theory should have a defined scope and try to predict and understand behavior in relevant situations.
- A theory should reflect best knowledge, i.e., even relationships based on weak evidence may be included if there are reasons to believe that the relationship is more likely than other relationships. The strength of the theory, or its components, should be described. A weak theory may frequently be better than no theory.
- Testing theories means to create predictions applying the theory and then examine the validity of these predictions through empirical studies. A theory that escapes many falisification trials increases its validity. Tests of theories may be conducted applying experiments with non-random sampling, observational studies applying statistical hypothesis testing with randomly drawn samples, or, any other meaningful study design. Falsification of a theory may lead to modification of the theory or to stronger belief in alternative theories.

Based on these beliefs we propose the following general phases for empirical software engineering studies:

1. *Collect and synthesize relevant studies:* In our experience, most software engineering studies are based on few, if any, references to studies made outside the software community. We believe, however, that we should accept and take the consequences of that software engineering is a multi-disciplinary field with strong links to, e.g., psychology, social science, project management, organizational theory, and, construction engineering. The claim that we need more exploratory studies before we establish theories are, we believe, very much based on an ignorance of the usefulness of results achieved in other, strongly related, disciplines. Skills in "review" and "research synthesis" are important to be able to this step properly. In our opinion, there are much too few review papers in software engineering trying to summarize relevant research, including that from other disciplines.
2. *Establish a theory*: In many cases there are no established theories relevant for answering important software engineering research questions. In this case, we should try to establish best knowledge relationships based on all available studies in our field and, not least, in related fields, as outlined in Step 1. Without a much stronger focus on this theory-development step, we probably will continue to produce isolated, exploratory studies with limited ability to aggregate knowledge.
3. *Test and/or refine the theory*: Empirical software engineering studies should test and refine the theory. The tests should be able to falsify relevant or uncertain parts of the theory, or to improve the theory. The tests may be based on statistical hypothesis testing, but also on other testing techniques. All good argumentations should be accepted, not only those based on generalization from sample to population.
4. *When needed, change or replace the theory*: If a test falsifies the theory, we may have to change or replace it.

There is nothing revolutionary with this approach, i.e., it is well within the framework of the hypothetical-deductive method of science. These steps are not meant to be "silver bullets" to better research results. As with statistical hypothesis testing, there are many ways of misusing the approach. Instead of a, in many ways, mechanical process imposing rigid structures on the researchers (statistical hypothesis testing), we now have a process requiring maybe even more reflection and skill regarding study design and argumentation. If there is no good quality assurance of the studies and argumentations, we may easily fall into the trap of the research method of "anything goes", i.e., all study designs and generalization techniques are equally good. In addition, the building of theories easily lead into "schools", i.e., some research communities believes in one theory (and do their best to support it) and other communities in other theories (and do their best to validate their theories).

The following example of research study, which is based on the same research question as in Section 2, illustrates what the proposed theory-based research study approach may mean in practice.

Example: As in Section 2, we want to know which cost estimation methods that work in which situations, i.e., we want a theory for selection of cost estimation methods. We now start with the collection of relevant studies and the development of a theory.

Assume that we want to compare expert estimation (A) with linear regression based estimation models (B), i.e., we want to know when we can expect expert estimation to provide more accurate estimates than do linear regression based estimation models. In several domains, particularly in psychology, there have been numerous such studies, see [9] for an overview. Based on these studies we formulate an element of a theory of selection of estimation methods (only outlined here), e.g.:

T1: Expert estimates are on average more accurate than linear regression-based estimates when the variance in project types is low.

A good theory should, as stated earlier, preferably include cause-effect relationship, and, have a well-defined scope. T1, which is only an element of a theory, is therefore not optimal (although it nevertheless may represent the best knowledge about the relationship).

The wish for better understanding of the scope and validity of T1 leads us to design an experiment. In this experiment we try to distinguish between two different types of project type variance, i.e., project type variance which is a) included, and, b) not included in the training set used to derive the regression models. We would expect the expert estimators to be more flexible about estimating the cost of new types of projects, i.e., projects different from those in the learning set of projects. We therefore propose that:

Expert estimators are on average better than linear regression-based models in situations with high variance of project types, i.e., in situations where there are new types of projects to be estimated that are different from the one in the regression models' and experts' learning sets.

To test this proposal we conduct a series of experiment where the experts and the linear models learn from the same sets of project data and try to estimate the cost of projects different from those in the learning set. Instead of trying to sample from a large population of subjects, models, projects and contexts, which probably would not be practically possible, we now design a series of experiments that test the proposed relationship in different contexts with similarity to important industrial contexts and with the purpose of understanding when projects different from those in the learning set make differences between how well "formulas" and "experts" are performing.

Assume that we, amongst others, find evidence suggesting that most experts were clearly better than most linear regression models in predicting project costs when the projects were unusual regarding productivity (size/effort). (The scope of this finding should, of course, be described, e.g., for what type of models and what type of experts is this relationship valid). We now may choose to replace/refine the theory T1 with T1-new:

T1-new: Expert estimates are on average more accurate than linear regression-based estimates when the variance in project types is low, or, when the variance is high and the productivity of the projects is unusual regarding productivity (size/effort).

Whether we chose to refine the theory or not, will in this case not depend on an arbitrarily set significance level with no link to results in previous studies. Instead the change depends on our ability to provide a robust argumentation, including use of results from other studies, and maybe the resolving of previous contradictory observations. Our main concern is now whether T1-new reflects best knowledge better than T1 or any alternative theory.

An interesting consequence of this approach is that the discussions about use of artificial experiments vs. realistic observational studies, and, student participants vs. software professionals can be reduced to a question of *fit* between the test of a theory and the chosen study design. It may, for example, be that an implication from a theory more easily can be tested using an artificial environment and homogenous participants, i.e., pen-and-paper based programming and computer science students. It is the theory-related argumentation that counts, not the type of study design. This said, we strongly believe that there is a need for more realism in empirical software engineering studies today. There are, regardless of type of generalization argumentation, frequently difficult to generalize from inexperienced students solving very small programming tasks to relevant industrial programming contexts.

5. Conclusions

This paper describes an attempt to formulate how we should conduct empirical software engineering studies. An important purpose of the paper is to get responses on the meaningfulness and practicality of our suggestions. What we suggest is, in short, that there is a need for more focus on research questions derived from theory and generalization across populations, as opposed to isolated hypotheses and generalizations from sample to population through statistical hypothesis testing. Our suggestions are, amongst others, based on the observation that basic assumptions of population and random sampling are violated in many empirical software engineering studies, and, that the focus on statistical hypothesis testing seems

to hinder the aggregation of empirical knowledge. In cases where statistical hypothesis testing is appropriate, e.g., as means to test implications from a theory, empirical software engineering studies should base the tests on well-defined populations and randomly drawn samples.

Acknowledgement: Thanks to Vigdis By Kampenes, Tore Dybå, Erik Arisholm and other researchers at Simula Research Laboratory for interesting discussions on the topic of generalizations.

References

1. Gigerenzer, G., *Adaptive thinking: Rationality in the real world*. 2000, Oxford: Oxford University Press.
2. Hacking, I., *The emergence of probability*. 1975, Cambridge: Cambridge University Press.
3. Hacking, I., *An introduction to probability and inductive logic*. 2001, Cambridge: Cambridge University Press.
4. Salsburg, D., *The lady tasting tea: How statistics revolutionized the twentieth century*. 2001, New York: First Owl.
5. Kitchenham, B., et al., *Preliminary guidelines for empirical research in software engineering.* IEEE Transactions on Engineering Management, 2002. **28**(8): p. 721-734.
6. Skovlund, E., *A critical review of papers from clinical cancer research.* Acta Oncologica, 1998. **37**(4): p. 339-346.
7. Wonnacott, T.H. and R.J. Wonnacott, *Introductory Statistics*. 1990, New York: John Wiley & Sons.
8. Shepperd, M. and G. Kadoda, *Comparing software prediction techniques using simulation.* IEEE Transactions on Software Engineering, 2001. **27**(11): p. 1014-1022.
9. Jørgensen, M., *A Review of Studies on Expert Estimation of Software Development Effort.* Journal of Systems and Software, 2004. **70**(1-2): p. 37-60.