Empirical Software Research

An Interview with Dag Sjøberg, University of Oslo, Norway

by Walter Tichy

Editor's Introduction

Punched cards were already obsolete when I began my studies at the Technical University of Munich in 1971. Instead, we had the luxury of an interactive, line-oriented editor for typing our programs. Doug Engelbart had already invented the mouse, but the device was not yet available. With line editors, users had to identify lines by numbers and type in awkward substitution commands just to add missing semicolons. Though cumbersome by today's standards, it was obvious that line-oriented editors were far better than punched cards. Not long after, screen oriented editors such as Vi and Emacs appeared. Again, these editors were obvious improvements and everybody quickly made the switch. No detailed usability studies were needed. "Try it and you'll like it" was enough. (Brian Reid at CMU likened screen editors to handing out free cocaine in the schoolyard.) Switching from Assembler to Fortran, Algol, or Pascal also was a no-brainer. But in the late '70s, the acceptance of new technologies for building software seemed to slow down, even though more people were building software tools. Debates raged over whether Pascal was superior to C, without a clear winner. Object-oriented programming, invented back in the '60s with Simula, took decades to be widely adopted. Functional programming is languishing to this day. The debate about whether agile methods are better than plan-driven methods has not led to a consensus. Literally hundreds of software development technologies and programming languages have been invented, written about, and demoed over the years, only to be forgotten. What went wrong?

What happened was that it became less and less clear whether a new software technology was truly better than another. Software developers became skeptical. "Try it and you'll like it" didn't work anymore—there were too many variables involved and the improvements, if any, were too small or too rare to be discerned by the individual. Nobody in industry had the time to try out the many competing ways of writing better software.

The answer was to conduct controlled studies to check what would happen when using a given technology. Computer scientists began to apply empirical methods. A whole new area, called "empirical software engineering" or "empirical software research" sprang up, complete with workshops, conferences, and journals. One of the research questions explored early on was whether software inspections were effective at reducing defect density. Currently, agile methods are a hot topic among empiricists. This interview takes a snapshot of the status of empirical software research. Dag Sjøberg of the University of Oslo, who has run the largest and most realistic experiments in this area, shares his views with Ubiquity *readers.*

Walter Tichy
Associate Editor

Empirical Software Research

An Interview with Dag Sjøberg, University of Oslo, Norway

by Walter Tichy

Ubiquity: Empirical studies of software technology have been gaining in popularity. Why are such studies needed?

Dag Sjøberg: The choice of software technology in the IT industry has been driven too much by hype, fashion, and self-proclaimed gurus with vested interests. Fortunately, there is now somewhat of an agreement that empirical evidence collected in a systematic way should be part of the basis for important decisions—having data to back up claims should be part of any scientific or engineering discipline. I also believe that this is appealing to most practitioners.

Another reason for conducting empirical studies is to demonstrate that the questions regarding usefulness of various software technologies are often superficial. One often asks whether a technology works or whether method A is better than method B. These are meaningless questions. It's like asking whether a helicopter is better than a bike. The real question should be: "When is A better than B?" That is, for what kind of tasks, software, people (developers), processes, and other aspects of organizational and technological context is technology A better than B? To answer such a question, one must use empirical methods. Given that software development is a human activity carried out in organizations, we cannot use mathematical logic to deduce the answer. Carrying out empirical studies of software technology is about making our discipline more informed and rational.

Ubiquity: You and your coworkers have run an experiment with almost 300 professional programmers. What are the findings you came up with?

Sjøberg: Overall, we found that both the skill of the programmers and the complexity of the tasks greatly influence the effect of using a technology. In particular, we found that the question of whether pair programming is useful cannot be answered as such; it must be nuanced.

In the first part of the experiment we investigated the claim that a delegated control style in software is better than a centralized one. Delegation is supposed to be a good object-oriented principle and therefore better. It turned out that delegation was only better for highly skilled experts. Most developers are not in this category. For the majority, a software design with a centralized control style seems more maintainable than a design with a delegated control style, which is in contrast to what "gurus" in the area have claimed.

We then investigated the effect of pair programming. Here, too, the picture is more nuanced than what is often presented. We found that junior consultants made significantly more correct changes when they worked in pairs; the seniors did not. The intermediate pairs made more correct changes only on the more complex tasks. We found that for all developer categories, pair programming requires significantly more effort than individual programming. In our experiment, we paired developers at the same skill level. Elsewhere, it has been argued that pair programming with one senior and one junior is useful for teaching purposes. It may certainly be the case that juniors will learn from seniors this way, but there is no doubt that this way of teaching is very expensive.

We also tested the effect of personality, which has been the topic of many papers. In total, 160 developers were tested using the validated Big Five model. We cannot exclude the possibility that personality may influence long-term team performance, but in our experiment there were no strong indications that personality had any effect. Skill level, task complexity, and even country were stronger predictors of pair performance. So, the lesson learned is that industry and research should ignore personality and focus on skill level and task complexity instead.

Ubiquity: Employing 300 professionals for an empirical study is quite expensive. Under what circumstances is such expense justified?

Sjøberg: If the goal is to build useful knowledge, we cannot run experiments with students on toy systems. Can we justify *not* running proper experiments with industrial relevance? We spent about $340,000 US in total on direct costs for hiring those professionals from 27 companies in three countries. Is that expensive? Well, from the point of view that probably nobody else has paid that much to hire people for an experiment in computer science, it is expensive. But if you compare with the costs of running large experiments in, say, physics or medicine, it's a small amount. In any case, the cost is comparable to that of a post-doc over three years, including salary, university overhead, travel, and equipment. As a research leader, my policy has been to have about 25 percent of total budget to conduct empirical studies, at

the expenses of having more researchers in my group.

Given the importance and potential for great economic savings when building software, I'm surprised that there is not more investment in running comprehensive experiments to assess the trade-offs of various software technologies.

In our experiment, we wanted to study the moderating effect of different levels of system complexity and different categories of developers on the effect of pair programming. We ran a power analysis beforehand to see how many developers we needed in each group in order to detect effects if they in fact were present. So, to run this comprehensive experiment in a sound scientific way, we had to pay that much. But note that this experiment was also an investment. The data collected is still being analyzed in other studies and informs important decisions in yet other studies.

In any case, I believe that if we really want to run experiments in realistic settings that dig into the complexity caused by the many interacting variables that affect software development, we need to be better at seeking funding for such studies.

Ubiquity: Will you continue to run such large trials?

Sjøberg: Yes, I'm still in charge of a budget with a few hundred thousand dollars that I can spend on experiments. One area of research is agile development. In spite of an enormous interest in the software industry, there have been very few solid studies in the field. I've already started collaboration with a few companies. It's likely that the experimental unit this time will be teams instead of individuals or pairs.

We don't only run experiments. Recently, we have conducted a controlled, comparative case study. We hired four companies (instead of one) to develop the same system independently of each other. The cost of this trial was about $250,000 US. Even though the price for the four projects varied by a factor of seven, there was no clear relationship between price and quality of project and product. We are still working on the analysis of the effect of various context and process variables.

Ubiquity: Questions about using student subjects often come up. What is your opinion on this?

Sjøberg: I believe that we would make little progress if we continued with experiments with students of the researcher as subjects and tasks that are so small that they fit within the time frame of a single lecture. The value of such experiments for the software industry is questionable. We can of course run initial student experiments to help formulate hypotheses

and test experimental design, like they do with rats in medicine in earlier stages of a research program. But in later stages medical researchers need to test drugs on humans; we need to test on professionals.

I'll try to make my stand clearer. In general, one should sample subjects from the population to which one wants to generalize the results. Senior researchers in the community have claimed that even though the absolute performance level of students is lower than that of professionals, if technology A is better than technology B for a group of students, it will also be better for professionals. This is not necessarily correct, as we have demonstrated in several studies; the relative performance of a technology may depend on the skill level.

The point is that to make useful assessments or comparisons of software technologies, the participants in studies must be above a certain skill level, and different skills above this threshold may result in different relative results. When cars are tested, the test drivers aren't people without driving licenses who drive the car 20 meters in a parking lot. And even with licensed drivers, the results may be quite different if the test driver is, say, a journalist of a car magazine or a Formula 1 driver. We need to be much better at ensuring that the results of assessing a certain technology describe the technology relative to skill level. In an experiment, we therefore need to assess the participants' skills that are relevant to the phenomenon being studied.

Ubiquity: The high cost for professional subjects may discourage empirical research. Almost all of the published empirical research fails to meet your standards. What should researchers do?

Sjøberg: Very few in our community seem to include expenses for carrying out experiments in their grant applications. The current attitude is that empirical studies should be inexpensive. I believe that applying for expenses for professionals or companies to take part in empirical studies should be as natural as applying for expensive laboratory hardware. I had problems when I started applying for funding for experiments, because I couldn't refer to other similar cases, but now we have paved the way and demonstrated that it is a sensible way to spend research money.

Alternatively, if one doesn't succeed in attracting such money, one option is to find out what software companies are interested in being investigated. For example, I know several companies that are interested in testing different agile techniques. Such companies may be willing to let their developers take part in an in-house experiment or case study as long as the research questions are relevant to them.

Note that it is possible to conduct case studies that also include some degree of control, for example, comparative case studies where one analyzes and compares data across cases. The reason that case studies seem to have a somewhat negative reputation in our field is, I believe, that they are in fact not case studies at all, or that they are poorly conducted. One often considers case study research as soft, because researchers don't follow systematic procedures. In software research, the term case study often denotes a description where the researchers themselves apply their own technology on a fictitious toy example.

To encourage higher standards of our empirical studies, I believe we also have to somewhat change the criteria for publication. For example, a careful description of the motivation and design of an empirical study should be sufficient as the sole contents of a conference or workshop paper.

Ubiquity: I personally find it difficult to motivate students to perform empirical research. Most students (and the people who hire them) get more excited about developing new technology. Do you have any advice on how to integrate empirical studies into technology development?

Sjøberg: I think this is a challenge for the whole community. It has to do with the perception of our discipline. The purpose of computing is eventually about solving practical problems. Developing software technology without applying it doesn't make sense. Still, we haven't yet made systematic, empirical evaluation an integral part as in other disciplines focused on application, as, for example, medicine. Testing the effect of a drug is as important as developing it.

I believe that empirical evaluation should become a natural part of our teaching curriculum. At present, there may be courses in empirical methods at the master's and Ph.D. levels for particularly interested students, but rarely at the bachelor's level. This year I took a risk: I introduced two lectures in empirical research methods in my first-year course in software engineering with 300 students. I was positively surprised. For example, the results from our experiments that show which technology is better depends on skill level created quite a lot of debate.

I also believe that if we manage to put more emphasis on evaluation, we may recruit a wider category of people. We need the hackers (in the positive meaning), but we also need people who are concerned about how technologies are used. For example, it is well known that women are generally more interested in the application of a technology than in the technology

for its own sake. More focus on evaluation may lead to the recruitment of more women.

Ubiquity: Is empirical software research similar to drug research?

Sjøberg: It can be compared to drug research in the sense that large, expensive controlled studies may be needed to answer the "when does it work well" question. However, there is a principal difference between pharmaceutical drugs and software development technologies. Drugs are potentially life threatening. That is why there are laws and regulations that mandate extensive testing and legal authorities who must approve a drug before it can enter the market. Support technologies are not dangerous to software developers. So it is unlikely that careful evaluation will become mandatory or that we will see approval bodies in our discipline.

As a consequence, the vendors of software development tools will hardly start running large-scale, scientific evaluations. They will find it more profitable to spend money on marketing than on testing. However, we may see an increase in collaboration between vendors and academic researchers. The researchers get useful data from which they can publish, and the vendors get useful feedback on the usability of their products.

Ubiquity: You suggest that we should integrate empirical evaluation into all software research, but wouldn't it be better to have researchers who primarily do empirical evaluations of other people's contributions?

Sjøberg: I think we need both. If we want all or most methods and tools to be informed by empirical results, they should be subject of preliminary evaluation by the technology developers themselves. These developers can then address weaknesses immediately before the technology is shared with a wider audience. Having an evaluation in mind may also help direct the initial development.

On the other hand, we know that published studies of our own technologies give much more positive results than studies run by others. The human bias is significant. So, at least for technologies that seem to have potential to become widely used, there is a need for objective, empirical evaluation by external researchers with sufficient competence, resources, and without vested interests.

Ubiquity: Bertrand Meyer, during his keynote at the Empirical Software Engineering Conference

in 2010, said the following: "The premise of empirical software engineering is that methods and tools should undergo objective assessment. Many empirical studies, however, fail to give conclusive evidence on the questions of fundamental interest in software engineering. If empirical software engineering cannot answer such questions, its benefit is debatable."

What is your reply?

Sjøberg: The premise is correct. I agree that there is too little focus on fundamental questions in empirical software engineering. However, the fundamental questions are complex, and individual studies cannot give conclusive evidence on the fundamental, complex questions. Only series of studies that complement and build on each other can give conclusive evidence. Single studies in cancer research are not conclusive, either. We need to remember that assessing software technology in a meaningful way is very difficult, because its usefulness typically depends on the tasks, systems, users, and organizational and technological contexts. It's unrealistic to expect single studies to give conclusive answers.

However, Meyer is right in that much of the empirical work could have given more conclusive results than it has. Better methodological and theory-building skills, increased industry collaboration, resources to run comprehensive studies, and replications and meta-analyses to investigate reproducibility are needed to make significant progress in this area. I miss larger research programs on software technology, where senior researchers from several research groups work together over time. At present, there are too many single professors with their Ph.D. students/post-docs who publish papers to keep up their publication rate. Given that software is present in almost all spheres of modern information society, I wonder why our discipline doesn't have research programs like the Human Genome Project or research centers like CERN.

Ubiquity: What role does theory play in software research?

Sjøberg: Theories are crucial to the progress of our field. In the interaction between humans and computers, a plethora of theories exist. Most of them originate from the social sciences and psychology, but quite a few are also proposed on the basis of results of empirical software studies. Papers that report software-relevant theories are well cited and the theories are used, but there is no doubt that if more researchers use and develop the same theories, they will progress faster.

The benefit of using theories is that your work does not stand in isolation; developing theories is a way of accumulating knowledge. Correlations don't lie, but the inferences drawn from them

may be dubious if results run contrary to expectations from a theoretical standpoint. Theories denote maturity within a field, and congruence with multiple theories operating at different levels of abstraction is important. Note that I don't believe in grand, universal theories in software research like what can be found in physics. I believe we have to develop middle-range theories, which involve some abstraction but are still closely linked to observations, including context-specific information.

Furthermore, we often conduct empirical studies to investigate the relative merits of technologies, that is, the utilitarian view. However, the ultimate goal is to acquire deep understanding, which is condensed into models and theories. Good models and theories should be able not only to explain what has been observed, but also predict phenomena that have yet to be observed. In my opinion, an applied discipline like computer science research should focus on those phenomena that are likely to be useful to humans. Still, we never know for sure what would be useful in future, so we should afford to have a (minor) proportion of research on phenomena for which we cannot see any immediate use.

Ubiquity: Can you give us examples of useful software theories?

Sjøberg: Examples of theories, also called models, are Human Information Processing (HIP) models, the Model of Personal Computer Utilization (MPCU), the Technology Acceptance Model (TAM), and the Theory of Planned Behavior (TPB). Although their origin is not within the field of software research, they have been successfully applied or been the basis for other theories or models proposed in our field. Examples of such theories, which have been further developed on the basis of empirical studies, are theories of program understanding and comprehension, a theory of coordination in software engineering, a model of inspection processes, a model of complexity of delegation, and economic models for the business value of a development project. To strongly profile our theories, I believe we should be better at naming them and giving them acronyms, like the ones that we adopt from other disciplines.

Ubiquity: Where do theories come from?

Sjøberg: Many theories in computing, including the most well known ones, are based on derivations from first principles in a mathematical framework. I often meet computer scientists whose attitude is that the "real" theories in computer science are the mathematical ones. I believe the reason is that much of the roots of computer science are in mathematics, but when the field of computing now has broadened to include many human aspects as an integral part,

mathematical theories can to a lesser extent be the starting point for theory development. A few older empirically-based models exist that originated from within the field of computing, for example, the Lehman-Belady growth models of the 1970s and the Belady-Denning locality models of the 1960s. However, our community should endorse to a greater extent work to develop and validate empirical models or theories. It's simply about using the scientific method to develop scientific theories.

Ubiquity: Usability studies are also empirical. What are the connections to empirical software research?

Sjøberg: The Human Computer Interaction community places much more emphasis on empirical studies than most other subdisciplines of computing, and the many researchers with background in psychology in this area often have a good grasp of empirical research methods.

Usability is a primary dimension of software quality, and I've therefore collaborated with several usability labs on large empirical studies. Ideally, there should be much more collaboration between software and usability researchers, both with respect to research methodology and evaluation of concrete software technologies.

Ubiquity: Google encourages small-scale experiments, supported by data. How does that fit in?

Sjøberg: The idea that software developers run their own small experiments to support suggestions for improvement fits nicely into the overall picture of creating a more evidence-based culture than an opinion-based one. However, there is a distinction between usability for end-users and software development. The "like" buttons and the like provide feedback about how well some software fits user needs. Empirical software research aims mainly to find good ways to build the software in the first place. End users are not in a position to comment on the efficacy of how the software was developed. Understanding and improving fundamental aspects of software development requires resource-demanding empirical methods.

Another issue is whether simple evaluation methods such as "like" buttons actually are good indicators of usability for end users. I'm not convinced that researchers within Human Computer Interaction would agree on that.

Ubiquity: Unified Modeling Language (UML) is a technology that is taught widely, but there

have been some experiments that cast doubt on whether UML is helpful. Can you comment?

Sjøberg: I think this is a good example of a question that needs refinement before it can be investigated and answered properly. Which phases of the development cycle—requirement elicitation, design, testing, maintenance? Which levels of programmer skill? How does quality of support tools influence usefulness? It may be the case that using certain diagrams in the initial phases helps stimulate the creativity needed to propose good designs. It may be the case that maintenance is not accelerated with the use of UML diagrams, as a couple of studies indicate.

However, the lack of a framework for formulating specific research questions leads to a lack of sharing of independent, dependent, and context variables among studies, as well as questionable quality of many of the studies. So, if we reformulate your question to consider the effect of using graphical modeling languages in general, not only UML, that would be an interesting and challenge topic for a larger research program in the community. Given that UML has become a de facto standard in industry, such a research program should focus on which position modeling should have in various kinds of software development. This may require comprehensive, "holistic" studies that also include aspects such as costs, resources, and skill level.

Ubiquity: Lots of effort is being spent on analyzing software repositories. What is your opinion on this type of research?

Sjøberg: Software repositories of good quality that include information about under which conditions the data was collected are potentially great sources for analysis. A large amount of data from various projects is collected automatically, as opposed to the expensive data collection in traditional empirical studies. For example, one has found that software metrics can predict failure proneness from one version of a project to the next, but that the same metrics can't be used to predict failure proneness across projects.

A fundamental problem with using repositories is that the analyses are done post-hoc, that is, after the projects have finished. One can thus not vary parameters of interest in a controlled way to study cause-effects.

Another problem is the large number of correlations that one can establish without having contextual information to investigate whether the correlations represent cause-effects or are spurious. In traditional empirical studies the researchers have the possibility to observe and interview people along the way. I see a lot of really poor studies based on repositories from open source projects. Correlations are presented without context and papers end up with

absurd conclusions. I feel that many researchers have started fishing for correlations in repositories after realizing how hard it is to do solid "traditional" empirical studies.

[Editor's Note: For a broader discussion of software repository analysis, see the November 2010 Ubiquity *interview with Andreas Zeller: "Mining Your Way to Software Reliability."]*

Ubiquity: What are some of the empirical results that every software developer should know?

Sjøberg: One principle that every software developer should follow is that "small is beautiful." Developers should therefore put some extra effort in making their systems small. Our own, controlled studies show that smaller design and code, given other factors constant, lead to more understandable and maintainable systems. Also a large number of other studies show that size correlates negatively with many quality attributes. Note that this does not imply that the smallest possible system is the best. An analogy is written text—it should be minimal, but not less than minimal, or as Einstein stated: "Everything should be made as simple as possible, but not simpler."

Ubiquity: This is an example to which most people would react with "I could have told you so." Are there more substantial or surprising insights that have been gained or confirmed with empirical studies?

Sjøberg: You're right. In fact, this was more a message to researchers in the area than to practitioners. There are too many researchers who use obscure software metrics to indicate quality when they would have been better off using simple size measures. But focusing on practice, empirical studies have given many eminently useful results. Let me give three examples: Inspections are consistently shown to reduce errors, and two reviewers are sufficient. Well-applied design patterns will generally improve maintenance activities, and such patterns also improve communication among beginners. Static type-checking across module interfaces is effective in reducing the number of defects.

Ubiquity: What are the empirical studies that you are working on or planning?

Sjøberg: One study aims to quantify the trade-offs of time-boxed (Scrum) and flow-based (Kanban) development styles. Several software companies take part and are happy to change or modify their processes to test the effect on quality, productivity, and lead-time. We've already collected detailed information about 10,000 work items.

Another study investigates the effect of code smells on the maintainability of object-oriented software. We have four functionally equivalent Java systems, but otherwise they are different, for example, with respect to the number and density of so-called code smells. We hired six developers from Poland and the Czech Republic who carried out the same set of maintenance tasks on two of the systems each over a period of six weeks. The preliminary analysis indicates that code smells in general have almost no effect on maintenance effort. Much more important for maintenance effort is size. Note that the systems were developed by senior professionals and thus had reasonable quality.

In yet another study, 65 professional developers from eight countries solved 17 Java programming tasks over two days in an ongoing evaluation of an instrument for measuring skill, developed by a Ph.D. student of mine. The instrument started as a tool to help conduct our large-scale experiments and is currently being packaged as an industrial-strength tool to be used both for job recruitment and calibration in empirical studies.

Ubiquity: What are the most pressing questions in software research that should be explored empirically?

Sjøberg: There are many core areas that still need more investigations like how to design software to make it maintainable, but I would like to stress one general aspect: To be more relevant to industry, empirical evaluations of various software technologies should be accompanied by cost-benefit analyses. We often end up with studies that state that it's better to develop more models, design documents, other documentation, etc. and to carry out more inspections, testing, pair programming, etc. However, we rarely see the costs of doing this extra work. Like any engineering discipline, we have to take the costs into account, too.

Ubiquity: Fine, but more specifically, what are the really important open questions in software research that should be tested with the rigor and cost you applied?

Sjøberg: One pressing issue is identifying the level of planning that is optimal to carry out at the beginning of a software project for given categories of projects and contexts. Some elements of the process should be defined beforehand. The question is how much planning and to what detail, so that the planning effort is helpful but at the same time does not hamper a process that is sufficiently agile.

Another issue is that the many new methods, tool frameworks, and languages are usually developed by, and geared for, highly skilled individuals. The "average programmer" may not be

able to handle these tools very well. So a challenge is to develop technology that benefits the majority of programmers. We then need to conduct large-scale usability studies with that category of programmers as subjects to improve their productivity.

Ubiquity: What are your recommendations for individual software developers regarding empirical studies? Should they conduct their own?

Sjøberg: Individual developers should certainly be "reflective practitioners" with respect to their work situation and processes. They should write down their thoughts and give them as input to their team, project, or organization on how to improve. But I'm not sure if it is realistic that individual developers run their own empirical studies. However, at the organizational level, I would certainly recommend conducting studies to provide evidence for decision making. However, conducting such studies requires someone with knowledge of empirical methods in the organization, or collaboration with motivated researchers with interest in the same fundamental problems. Practitioners and researchers who identify challenges, take actions, and learn together is the core of *action research*, of which I would like to see more in our discipline.

In any case, conducting empirical studies requires some effort, but I believe it is a good investment instead of deciding on the basis of hype, fashion, or otherwise unsupported claims.

Ubiquity: Where can readers find out more about results in this area?

Sjøberg: I recommend the journals *IEEE Transactions on Software Engineering*, *IEEE Software*, *Empirical Software Engineering*, *Information and Software Technology*, and *Journal of Software and Systems*, and the Empirical Software Engineering and Measurement (ESEM) conference.

About the Author

Walter Tichy has been a professor of computer science at Karlsruhe Institute of Technology (formerly University Karlsruhe), Germany, since 1986. His major interests are software engineering and parallel computing. You can read more about him at www.ipd.uka.de/Tichy.

DOI: 10.1145/ 1998372.1998374

