What Works for Whom, Where, When, and Why? On the Role of Context in Empirical Software Engineering

Tore Dybå
Department of Informatics
University of Oslo and SINTEF
NO-7465 Trondheim, Norway
tore.dyba@sintef.no

Dag I.K. Sjøberg
Department of Informatics
University of Oslo
NO-0316 Oslo, Norway
dagsj@ifi.uio.no

Daniela S. Cruzes
Department of Computer and Information Science, NTNU
NO-7491 Trondheim, Norway
dcruzes@idi.ntnu.no

ABSTRACT
Context is a central concept in empirical software engineering. It is one of the distinctive features of the discipline and it is an indispensable part of software practice. It is likely responsible for one of the most challenging methodological and theoretical problems: study-to-study variation in research findings. Still, empirical software engineering research is mostly concerned with attempts to identify universal relationships that are independent of how work settings and other contexts interact with the processes important to software practice. The aim of this paper is to provide an overview of how context affects empirical research and how empirical software engineering research can be better 'contextualized' in order to provide a better understanding of what works for whom, where, when, and why. We exemplify the importance of context with examples from recent systematic reviews and offer recommendations on the way forward.

Categories and Subject Descriptors
D.2 [**Software Engineering**]

General Terms
Management, Measurement, Experimentation, Theory

Keywords
Evidence-Based Software Engineering, Generalization, Theory, Empirical Methods, Sociotechnical System

1. INTRODUCTION
What is best? Pair programming or solo programming? Test-first or test-last? A multitude of studies have been performed to answer these and other similar questions of the type: "Is <*technology x*> better than <*technology y*?" However, asking the general question of whether pairs outperform individuals in programming tasks, or whether test-driven development results in higher productivity, is meaningless. It is meaningless since these questions can be answered with "*Yes*" or "*No*" depending on the setting of the study. Still, posing research questions of this type, without considering contextual influences like the subjects of the study, the location, the time period, and the rational of the study, seems to prevail.

ESEM'12, September 19–20, 2012, Lund, Sweden.

However, we cannot expect a technology to be universally good or universally bad, only more (or less) appropriate in some circumstances and for some organizations [19]. The settings in which practice takes place are rarely, if ever, the same. For example, one software organization will have a different environment or be influenced by different environmental factors to that of another software organization. The size of the organization, types of customers, country or geographical location, the age of the organization, all impose different influences in unique ways. Additionally, the human factors, which form the organizational culture and make one setting different from another one, also influence the way software development is performed. We know that these issues are important for the successful uptake of research into practice and that there are interrelationships among organizational systems, structures, processes, technologies, settings, and cultures. However, the nature of these relationships is poorly understood.

This dependence of a potentially large number of relevant context variables in any study is an important reason for why empirical software engineering (SE) is so hard. Because of this, we cannot a priori assume that the results of a particular study apply outside the specific context in which it was run [6].

In an effort to bring context information in empirical research in SE more into consideration, Kitchenham et al. [32] suggested the following general guideline: "Be sure to specify as much of the industrial context as possible. In particular, clearly define the entities, attributes, and measures that are capturing the contextual information." However, as Whetten pointed out, it is not of much help to have a long context description if it is short on explanation [48].

Such explanation relies on understanding and interpretation of research evidence in light of the features and characteristics surrounding it. Contrary to empirical SE's treatment of context as a stable set of attributes of the world, the problem is that these surroundings themselves are selected and interpreted in different ways. There is an implicit parallel with linguistics here; that the meaning of a word is determined by the words and sentences that surround it. This raises the question about what a SE context is, how it is selected, and by whom.

The aim of this paper is to address this question and to provide an overview of how context affects empirical SE research and how this research can be better 'contextualized.' The remainder of the paper is organized as follows: Section 2 provides an overview of the concept of context and describes important dimensions and implications of context. Section 3 describes relationships between empirical studies and context, with examples from test-driven development and pair programming. Section 4 points to a potential way forward by suggesting how SE research can be better contextualized. Section 5 concludes.

2. WHAT IS CONTEXT?

The word contextus is of Latin origin and stands for weaving together or to make a connection [38]. Approaches to context and contextual dimensions range widely, reflecting different philosophical stances and practical orientations. In linguistics, for example, context refers to how readers can infer the meaning of a passage by referring to its intratextual clues; something that transcends the text itself [11]. In other words, trying to make sense of a single word in a sentence or of a sentence in a paragraph by looking only at the specific word or sentence and isolating them from the rest of the text in which they are used can be problematic, even if one knows technically their various linguistic meanings. For instance, "I am attached to you" has very different meanings to a person in love and to a hand-cuffed prisoner [33]. So, to take something 'out of context' leads to misunderstanding; there is no meaning without context. On the other hand, even if one is not familiar with the specific meaning(s) of a word or sentence, one can infer their correct meaning by situating them in the greater text and connecting them with the rest of the text.

In management research, context refers to the circumstances, conditions, situations, or environments that are external to a specific phenomenon and that enable or constrain it [47]. Mowday and Sutton see context as stimuli existing in the external environment [36], while Johns takes this a step further and understands context as situational opportunities and constraints that affect behavior [30]. Moreover, Johns distinguishes between substantive and methodological contexts [29], where substantive context stands for the context individuals or groups face while methodological context refers to detailed information about the research study.

In this paper we focus on substantive contexts for empirical SE, taking into account omnibus and discrete context dimensions as suggested by Johns [30]. Omnibus context refers to a broad perspective, drawing attention to who, what, when, where, and why [30], [49], while discrete context refers to specific contextual variables [30]. Thus, context can simultaneously be considered as a "lens" (omnibus context) and as a "variable" (discrete context) [25]. However, as most empirical SE research to date has studied discrete contexts, focusing on context as a set of variables, this paper emphasizes omnibus contexts, applying a context "lens."

2.1 Omnibus Context

According to Johns [30], research will benefit more from the careful consideration of context by paying more attention to designing and reporting studies along the lines of good journalistic practice in which a story describes the who, what, when, where, and why to the reader (see Fig. 1), thus putting recounted events in their proper context. This corresponds to the typical situation in empirical SE, in which we study how an actor applies technologies to perform certain activities on a software system [41].

'*What*' constitutes the substantive content of the research, the factors (variables, constructs, concepts) or treatments that logically should be considered as part of the explanation of the phenomena of interest. Although it might seem obvious, and maybe not strictly part of the context, it is not always clear what is actually studied. A typical problem is the descriptions of measures of the constructs being studied and the justification for variable coding. For example, if software quality is the phenomenon of interest, then it would require quite a bit of justification if only defects are being measured and how they are coded.

'*Who*' refers to the occupational and demographic context, and concerns both the direct research participants and those who surround them. It is important that the study clearly identifies the population about which one intends to make claims, and selects and describes subjects who are representative of that population [6]. Usually, there is an assumption that the target population is professional software developers. One should, however, be aware that this group may be very diverse [1]. A typical example of the 'who' in controlled SE experiments, is the student vs. professional [28] and the personality of the subjects [10]. However, it might not be enough to just state the occupational context or personality of a subject since individual differences in skill also affect the outcome of empirical studies. Within many different domains of expertise, with increased skill, the number of errors in performance decreases and the speed with which a task is executed increases [8]. The performance may differ significantly between various categories of professionals. Description of the 'who', and especially the skill, is important, thus, since the similarity of the subjects of a study to the people who will use the technology impacts the ease of the technology transfer.

'*Where*' a research study is conducted can have a noticeable impact on its results. It refers to the various locations in which software development happens. An important distinction is whether the phenomenon occurs in an artificial laboratory or in a more realistic industry setting [42]. For example, a challenge when configuring an experimental environment is to provide an infrastructure of supporting technology (processes, methods, tools, etc.) that resembles an industrial development environment. Because the logistics is simpler, a classroom is often used instead of a usual work place. Conducting an experiment on the usual work site with professional development tools also implies less control of the experiment than one would have in a classroom setting with pen and paper. Nevertheless, there are many challenges when conducting experiments with professionals in industry that might also be due to location effect such as economic conditions and organizational and national culture.

'*When*' refers to the time at which the research was conducted or research events occurred. For example, it includes when the data was collected and reflects the role of temporal factors in the research. Time affects the sociotechnical relationships that surround all aspects of software development, and is especially important for researchers who deal with software product life-cycles. Key contextual conditions of time are also related to whether the study is cross-sectional or longitudinal. Repeated calls for longer duration of experimental tasks [43] and more longitudinal research [40] in SE underscore the importance of the temporal dimension in contextual influences.

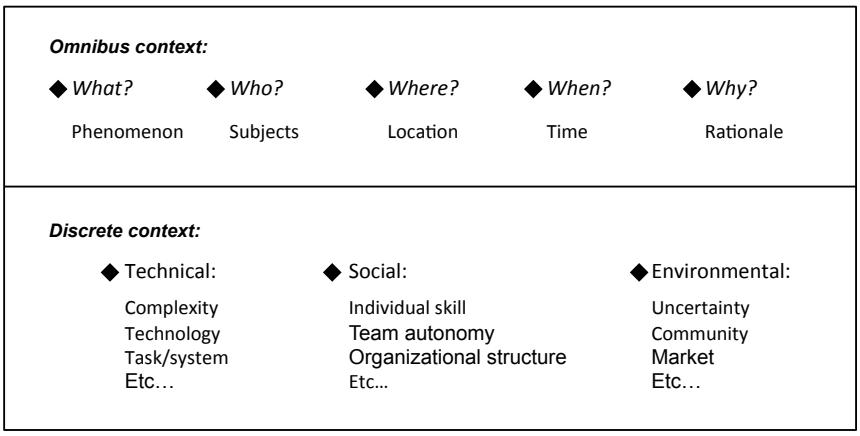

Figure 1. Important dimensions of SE context (adapted from [30]).

Time is also often a dependent variable in SE experiments, in which the goal is usually for subjects to solve the tasks with satisfactory quality in as short time as possible, as most software engineering jobs are subject of a relatively high pressure. However, if the time pressure on the participatory subjects is too high, then the task solution quality may be reduced to the point where it becomes meaningless to use the corresponding task times in subsequent analyses. A challenge is therefore to put a realistic time pressure on the subjects. How to best deal with this challenge depends to some extent on the size, duration, and location of the experiment. Methods for combining time and quality as task performance is currently being developed as a promising first step to measuring programming skill in both industry and research settings [8].

'*Why*' refers to the rationale for the conduct of the research or the collection of research data. Why data is collected can have a compelling contextual impact on organizational behavior and associated research. An experimental setting would, for example, ideally either reflect the subjects' organizational setting or would allow them to see some professional benefit from the experimental tasks, which would motivate them to put more effort and thought into the study. However, motivation can be a problem when subjects are asked to work on toy problems, are given unrealistic processes, or see some other disconnection between the study and their professional experience [6].

2.2 Discrete Context

Discrete context focuses on specific situational variables that directly influence behavior or moderate relationships between variables. Viewing software development from an open sociotechnical systems perspective [45], the lower portion of Figure 1 shows that the prominent dimensions of discrete context in SE research include technical, social, and environmental factors. Each of the variables within these dimensions may be treated as independent variables through selection or manipulation, or as moderator or mediator variables.

A moderator is a qualitative or quantitative variable that affects the direction and/or strength of the relations between an independent or predictor variable and a dependent or criterion variable [4]. Some moderator variables are categorical. Suppose, for example, that pair programming yielded high effect sizes on complex tasks but low effect sizes on simple tasks, with the opposite pattern emerging for solo programming. Task complexity is then a categorical moderator. Other moderators are continuous. An example would be if pair programming produced moderate effect sizes no matter how complex the task is, but solo programming produced low effect sizes for developers with few years' experience and high effect sizes for developers with many years.

Mediators reflect a mechanism through which the independent variable causes the mediator, which then causes the outcome. For example, suppose that self-management (the independent variable) causes higher motivation (the mediator), which then leads to increased software quality (the dependent variable). However, more complex and subtle relationships may exist. For instance, the same variable can be both a moderator and a mediator in the same model, and mediators can be nonlinear or non-recursive, see [4].

The elements of the three discrete context dimensions (Fig. 1) can be seen as mediating the omnibus context. For example, knowing someone's occupation often permits reasonable inferences about his or her tasks, social, and physical environment at work, which, in turn, can be used to predict behavior and attitudes [30].

We will not consider the details of all possible discrete context variables, as several lists of a large variety and diversity of such variables have already been identified and proposed elsewhere (e.g., [12], [37], [50]). Therefore, the elements of technical, social, and environmental context in Figure 1 are not meant to be exhaustive, but argued to be *important*. This importance is inferred from a combination of the fact that they are interrelated, that they operate at multiple levels of analysis, and that they appear as fundamental elements in several empirical SE studies.

Along the technical dimension, for example, there is often a gap between researcher expectations and empirical results because one fails to acknowledge that potentially more powerful technologies may be more complex to use, or may require new skills in order to use correctly. It is self-evidently true that a technology is better when it is easier *and* faster to use than when it is not. However, what if a "faster" technology comes at the price of added complexity, which makes the technology harder to use properly. Then the faster technology would require more training to be used successfully. Without training, the faster and more complex technology would be associated with a higher proportion of

incorrect uses, thereby making the faster technology appear worse than the existing alternative that is slow but easy to use correctly. More complex technologies frequently require more training than less complex technologies; at the same time, more complex technologies are adopted because they add to productivity [7].

Individual differences in skill are an important element along the social dimension that affects the outcome of empirical studies. When evaluating alternative processes, methods, or tools, skill levels may mediate the effect of using a specific alternative. For example, in an experiment on the effect of a centralized versus delegated control style, the purportedly most-skilled developers performed better using a delegated control style than with a centralized one, while the less-skilled developers performed better with the centralized style [1]. In another experiment, skill had a moderating effect on the benefits of pair programming [2].

Another important element along the social dimension is the complexity of team performance, which depends not only on the team's autonomy and competence in managing and executing its work but also on the organizational context surrounding it. For example, aspects of the organizational context such as reward systems, supervision, training, resources, and organizational structure can strongly affect team functioning. Likewise, relationships with the market and key stakeholders outside the team can influence task performance [35].

The environmental dimension refers to various characteristics outside the control of the organization that is important to its performance. These characteristics include the nature of the market, political climate, economic conditions, and the kind of technologies on which the organization depends. The environment of a particular software organization may range from stable to dynamic – from predictable to unpredictable. A study on process improvement strategies, for example, showed that there was no difference in the level of exploitation between small and large organizations regardless of the environment, but that there was a marked difference in the level of exploration. The results showed that small software organizations engaged in significantly more exploration in turbulent and uncertain environments than large software organization. Due to the increased complexity, increased convergence, and increased inertia of the large organizations, they are less likely to change in response to environmental changes than small organizations. They tend to generate the same response even when the stimuli had changed [15], [16].

The argument behind the usefulness of studying discrete context variables depend on the assumption that research findings are strongly a function of general empirical laws or processes. However, from a constructionist point of view, consistency of research results implies either the stability of the social constructions across the contexts in which the studies were conducted or an interpretive norm that leads to the perception of consistency [17], [27]. Similarly, inconsistency among research results might indicate an inconsistency among the interpretative norms of the research community. One of the most interesting implications of the constructionist perspective is that the perceived cumulativeness of scientific knowledge in SE is a function of the conventions of evidence and methodology in the research community.

3. CONTEXT AND EMPIRICAL STUDIES

It can often be problematic to transfer evidence generated from one context to another. Shull [39], for example, points to several recent studies showing that lessons learned from one project are simply not applicable to others. For example, a study of the prediction factors in the COCOMO model across 93 project datasets from one organization showed huge variations in the size of the effects those factors had on overall project effort. In some cases, the direction of the relationships changed from positive to negative depending on which projects were in the dataset being fit. Also, in another study on defect predictors for pairs of projects, only for four percent of the pairs did the factors that worked well for predicting defects in the first project also apply in the second [39].

In this section we will look further into three examples of the relationship between context and empirical studies of agile practices – two secondary studies and one primary study. The first one is a systematic review on test-driven development (TDD), the second is a meta-analysis of pair programming (PP), while the third is a large experiment on PP with professional developers. The systematic review and meta-analysis are typical examples of secondary studies which try to synthesize evidence from primary studies of the type: "Is *<technology x>* better than *<technology y*?" without considering contextual influences. Our primary study example, however, shows how important contextual elements influence the main effects of the experiment.

3.1 Test-Driven Development

Attempts to aggregate the available evidence on agile practices like test-driven development (TDD) have shown wide disparities in how well these practices work in different contexts. Although advocates claim that TDD enhances both product quality and programmer productivity, skeptics maintain that such an approach to programming would be both counterproductive and hard to learn [46].

However, it is the productivity dimension that engenders the most controversial discussion of TDD. Although many admit that adopting TDD may require a steep learning curve that may decrease the productivity initially, there is no consensus on the long-term effects. One line of argument expects productivity to increase with TDD; reasons include easy context switching from one simple task to another, improved external quality (i.e., there are few errors and errors can be detected quickly), improved internal quality (i.e., fixing errors is easier due to simpler design), and improved test quality (i.e., chances of introducing new errors is low due to automated tests). The opposite line argues that TDD incurs too much overhead and will negatively impact productivity because too much time and focus may be spent on authoring tests as opposed to adding new functionality [46].

A systematic review that aggregated the available evidence about the effectiveness of TDD found that TDD does not have a consistent effect on productivity. The evidence from controlled experiments suggests an improvement in productivity when TDD is used. However, the pilot studies provide mixed evidence, some in favor of and others against TDD. In the industrial studies, the evidence suggests that TDD yields worse productivity. Even when considering only the more rigorous studies, the evidence is equally split for and against a positive effect on productivity. Ten studies resulted in higher productivity for TDD than otherwise, nine studies led to worse productivity for TDD, while six additional studies found no significant effect on productivity at all [46].

Based on a more detailed investigation of the results, the authors could not recommended any specific context that would benefit from the use of TDD. The results do not suggest to which domains TDD is applicable, to which kinds of tasks within a domain, or to which projects sizes and complexities it is applicable. Furthermore, the studies do not make it clear whether TDD is an applicable practice for developing embedded systems or for developing highly decentralized systems where incremental testing may not be feasible.

3.2 Pair Programming

Pair programming is one of the best documented and most popular agile practices, and it has been the subject of a relatively large body of empirical research from an industrial perspective [18], [26]. Like the TDD example, common to most of the PP studies is the general question of whether pairs outperform individuals in programming tasks without consideration of context.

Claims as to both the benefits and the adversities of PP abound. Advocates of PP claim that it has many benefits over individual programming when applied to new code development or when used to maintain and enhance existing code. Stated benefits include higher-quality code, shorter development duration, happier programmers, improved teamwork, improved knowledge transfer, and enhanced learning (see [18]). There are also expectations with respect to the benefits and drawbacks of various kinds of pairing, e.g., expert–expert vs. novice–novice pairing. However, the finding across decades of small group research is that groups usually fall short of reasonable expectations to improved performance (see [26]).

Motivated by the diverse claims regarding PP a meta-analysis on the effectiveness of PP, which extended the analysis presented by Dybå et al. [18], was undertaken, taking into account between-study variance, subgroup differences, and publication bias [26]. The meta-analysis included 18 studies, which showed a small positive overall effect of PP on quality, a medium positive overall effect on duration, and a medium negative overall effect on effort. The meta-analysis suggests that PP is not uniformly beneficial or effective, that inter-study variance is high, and that publication bias might be an issue. It showed large and partly contradictory differences in overall effects, specifically with respect to duration and effort.

Also, differences in research design and methodological rigor across the studies, makes it difficult to compare the findings. For example, some studies used student homework assignments as experimental tasks, some used repeated measure designs in which the same subjects worked individually as well as in pairs, while others employed quasi-experimental designs for the assignment of subjects to treatments. The approach to partner selection also differed considerably, ranging from self-selection by pairs to matching pairs on abilities or experience levels. Yet another source of variation is the unit of analysis employed in these studies. For example, certain studies used the programming team as the unit of analysis, whereas others used a dyad [3].

These issues point to a need for untangling the moderating factors of the effect of PP. The meta-analysis also concluded that the question of whether PP is better than solo programming is not precise enough to be meaningful, since the answer depends on other factors, for example, the expertise of the programmers and on the complexity of the system and tasks to be solved.

Indeed, expertise and task complexity are perhaps the most central situation-independent predictors of SE performance [8]. Situation-dependent factors, on the other hand, include more dynamic factors such as motivation, team climate, organizational issues, etc. Theory predicts that experts perform better on complex tasks than do novices because experts' level of understanding corresponds to the deep structure of a complex task [22].

The collaborative nature of PP also influences what social mechanisms (e.g., social loafing, social laboring, social facilitation, social inhibition, and social compensation) are applicable. However, these social mechanisms also depend on a host of other factors. In a meta-analysis of social loafing (the phenomenon that individuals tend to expend less effort when working collectively than when working individually), Karau and Williams [31] identified several conditions in which such loafing is eliminated (e.g., by high group cohesion) and some in which the opposite phenomenon, social laboring, could be observed (i.e., greater effort on group tasks). Social laboring seems to occur when complex or highly involving tasks are performed, or when the group is considered important for its members, or if the prevailing values favor collectivism rather than individualism [9].

Group performance also depends on whether a task is additive, compensatory, disjunctive, or conjunctive [9]. For example, for conjunctive tasks, all group members must contribute to the solution, but for disjunctive tasks it suffices that one group member has the ability to complete the task. However, it is not obvious what sort of task PP is in this respect.

Figure 2 is good example of the relationship between context and empirical studies. It shows the results of a large experiment, with 295 professional developers, performed by Arisholm et al. [2] that found moderating effects of both task complexity and expertise. Overall, the results showed that the pairs had an 8% decrease in duration with a corresponding 84% increase in effort and a 7% increase in correctness. However, the main effects of PP were masked by the moderating effect of system complexity, in that simpler designs had shorter duration, while more complex designs had increased correctness.

When considering the moderating effect of programmer expertise, junior pairs had a small (5%) increase in duration and thus a large increase in effort (111%), and a 73% increase in correctness. Intermediate pairs had a 28% decrease in duration (43% increase in effort) and a negligible (4%) increase in correctness. Senior pairs had a 9% decrease in duration (83% increase in effort) and an 8% decrease in correctness. Thus, the juniors benefited from PP in terms of increased correctness, the intermediates in terms of decreased duration, while there were no overall benefits of PP for seniors.

When considering the combined moderating effect of system complexity and programmer expertise on PP, there appears to be an interaction effect: Among the different treatment combinations, junior pairs assigned to the complex design had a remarkable 149% increase on correctness compared with individuals. Furthermore, intermediates and seniors experienced an effect of PP on duration on the simpler design, with a 39% and 23% decrease, respectively. However, the cost of this shorter duration was a corresponding decrease in correct solutions by 29% and 13%, respectively.

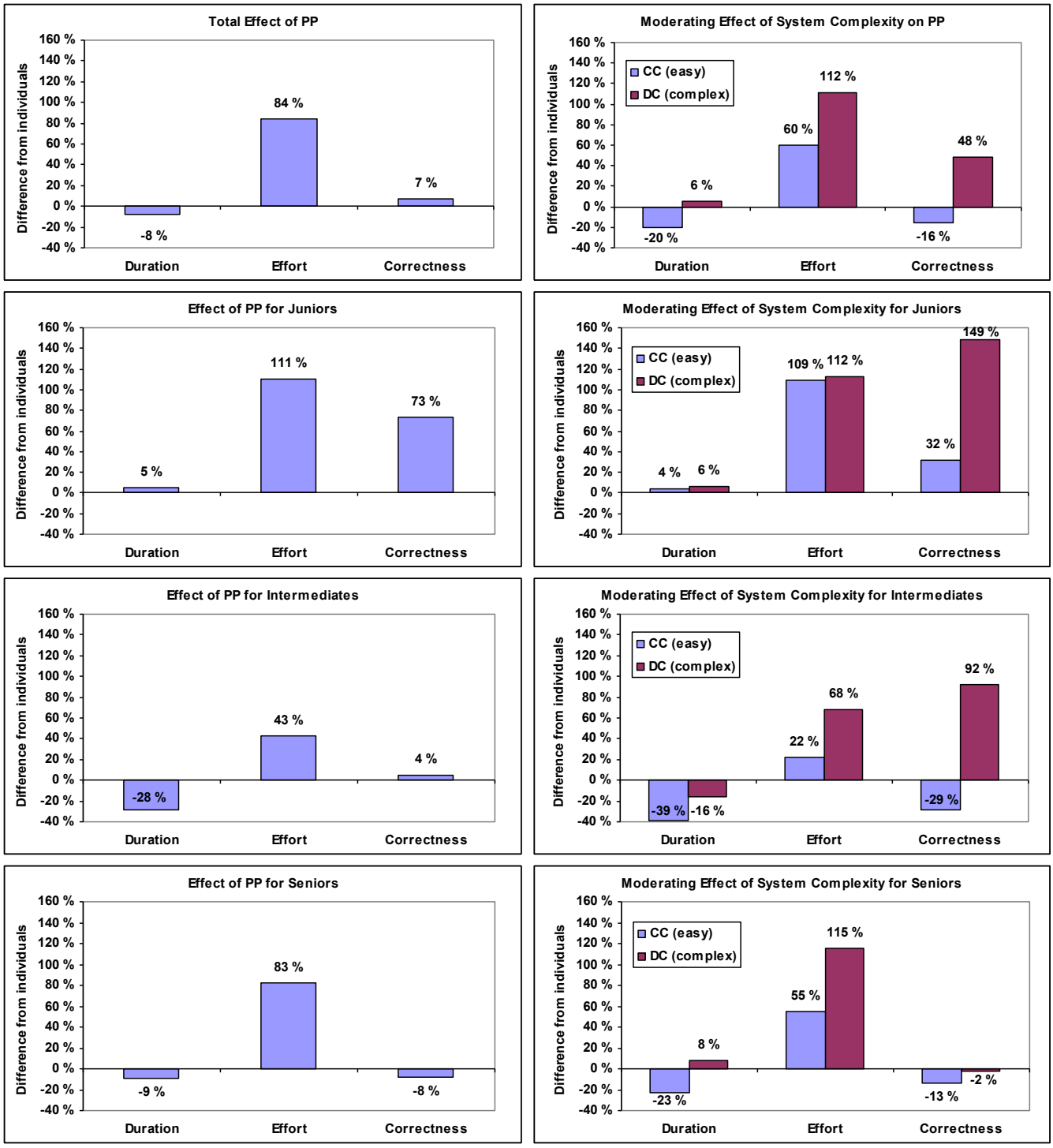

Figure 2. The moderating effects of programmer expertise (left column) and system complexity (right column) on the relation of pair programming on duration, effort, and correctness (Arisholm et al., 2007).

Hence, based on this experiment, the overall answer to the question of whether PP is "better" than solo programming is a clear "No". However, the more detailed examination of the evidence suggests that PP is faster than solo programming when programming task complexity is low and yields code solutions of higher quality when task complexity is high.

By cooperating, programmers may complete tasks and attain goals that would be difficult or impossible if they worked individually.

Junior pair programmers, for example, seem able to achieve approximately the same level of correctness in about the same amount of time (duration) as senior individuals. However, the higher quality for complex tasks comes at a price of a considerably higher effort (cost), while the reduced completion time for the simpler tasks comes at a price of a noticeably lower quality. These relationships give rise to a few evidence-based guidelines for the use of PP for professional software developers [18]: If you do not know the seniority or skill levels of your programmers, but do have a feeling for task complexity, then employ pair programming either when task complexity is low and time is of the essence, or when task complexity is high and correctness is important.

4. THE WAY FORWARD: CONTEXTUALIZING SE RESEARCH

The examples in the previous section clearly show that an obvious problem with current empirical SE research is in the stage of asking questions. The initial question posed, "Is <*technology x*> better than <*technology y*?", e.g., "Is pair programming better than solo programming?" is meaningless. SE covers a highly diversified set of sociotechnical tasks, procedures, and systems based upon more or less well-defined theoretical formulations. Narrowing the question down to "Does pair programming lead to higher quality than solo programming" is no more meaningful than the general question, since the range of tasks and systems remain as diversified as within SE in general. Furthermore, this question fails to take into account the skill of the developers who may contribute to quality.

So, what is the appropriate question to be asked of empirical SE research? In all its complexity, the question towards which empirical SE research should ultimately be directed is the following:

> *What* technology is most effective for *whom,* performing *that* specific activity, on *that* kind of system, under *which* set of circumstances?

This question resembles the goal part of the goal-question-metric (GQM) paradigm, which is a systematic approach for setting project goals tailored to the specific needs of an organization and defining them in an operational and tractable way [5]. Relating it to the omnibus context in Figure 1, and the example of PP in Section 3.2, we find:

> What technology (PP vs. solo programming), is most effective (in terms of improvement in correctness), for whom (highly skilled developers), performing that specific activity (change tasks), on that kind of system (complex system designed with a delegated control style), under which set of circumstances (in the subjects' normal work environment for a full-day experiment)?

Posing the question in this manner, it becomes obvious that in order for knowledge to meaningfully accumulate across separate studies and provide a solid empirical foundation for subsequent research, it will be necessary for every empirical investigation to adequately describe, measure, or control a potentially large number of discrete context variables. However, as we will demonstrate, applying an experimental logic to this problem based on ready-made lists of discrete context factors is not a viable option.

The experimental logic, which underpins most empirical SE research, is to identify the 'dependent variable' that captures the 'outcome' or 'effect' that needs to be explained, and the 'independent variables' that have impact on, or explain variation in the dependent variable. A change in the independent variable (X) is said to 'bring about' change in the dependent variable (Y). The goal of an experiment is thus to isolate the critical causal condition, or treatment, (X_1) by experimental manipulation, with other potential influences (X_2, X_3, X_4, etc.) being held in control. The influence of X_1 on Y can thus be observed and measured directly. Another strategy for achieving 'control' is by statistical means; by observing and including variables in *post hoc* analyses.

The quasi-experimental and analytical modelling traditions in empirical SE research build on these strategies in which variables do the explanatory work and causal complexity is managed via the progressive addition of subsets of variables (like in the PP example). The prevailing view on context in empirical SE has the same variable oriented logic.

However, if we were to evaluate only a small selection of discrete context variables that presumably would influence the main effects of a study using this logic, we would quickly find it difficult because of combinatorial complexity. As an example, Petersen and Wohlin [37] suggested a checklist for context documentation consisting of six facets: product, processes, practices and techniques, people, organization, and market, and a set of context elements describing each facet. However, Menzies et al. [33] criticize Petersen and Wohlin since "they offer no way to learn new contextualizations or … no experimental confirmation that their contexts are the 'right' contexts" (p. 350), claiming that context is interesting only if it results in different and better treatments.

In total, Petersen and Wohlin suggested 21 context elements [37]. Even with simplified assumptions and a very conservative estimate for the number of legal values per context element, we get more than 4 billion combinations! Making matters even worse, Clarke and O'Connor [12] proposed a reference framework of situational factors for software development processes consisting of 44 factors and 170 sub-factors. Assuming only two legal values for each sub-factor this results in a minimum of:

$$2^{170} = 1.5\text{x}10^{51} \text{ combinations of context factors.}$$

As a comparison, there are:

$$1.33\text{x}10^{50} \text{ atoms in the world}^{1},$$

which shows the absurdity of the variable oriented logic. Consequently, we need to shift focus away from a checklist-based approach to context in favor of a more dynamic view of software practice. Instead of viewing context as a set of discrete variables that statically surround parts of practice, we argue that context and practice stand in a mutually reflexive relationship to each other, with software practice, and the interpretive work it generates, shaping context as much as context shapes the practice.

On the one hand the traditional variables of empirical phenomena have to be supplemented by sociotechnical attributes and patterns that are intrinsic to the activity of software practice. On the other, the characteristics of research evidence as an interactive phenomenon, challenges the traditional notions of empirical SE

[1] http://education.jlab.org/qa/mathatom_05.html

research, suggesting a view of the relationship between evidence and context as a process that emerges and changes through time and space.

Given that in any study, there are an infinite number of contextual factors and combinations to consider, the decision as to which parameters along which to contextualize should be no different from the decision regarding which variables to control. Both decisions should be grounded in theory relevant to the phenomenon under study, or as Menzies et al. [33] formulated it: "*Rather than focus on generalities (that may be irrelevant to any particular project), empirical SE should focus more on context-specific principles.*"

Therefore, we do not expect a single, precise, technical definition of context in SE. The term means quite different things within alternative research paradigms, and even within particular traditions seems to be defined more by situated practice, by use of the concept to work with particular analytic problems, than by formal definition [24]. Like Goodwin and Duranti [24], we do not see the lack of a single formal definition, or even general agreement about what is meant by context, as a situation that necessarily requires a remedy. We clearly dispute any attempt at providing a general framework or checklists of specific factors intended at describing the context of local, situated practice.

Instead, we encourage SE researchers to take a broad, omnibus, perspective to context in their studies and to actively take part in explaining how phenomena works, for whom, where, when, and why. Context is shaped by the specific activities being performed. It is crucial, therefore, to acknowledge that any definition of context can only be done in relation to a specific practice situation [17].

However, simply naming an organization, describing a site in detail, or doing a longitudinal study does not constitute a contextual contribution. Rather, these means of fostering context have to be used in a way that adds explanatory value to a study. So, if we are to move beyond simple assertions that the context is important, we need to articulate more clearly how contextual influences operate. Perhaps the best question to ask oneself is this:

> Does the inclusion of this information explain the constraints on, or the opportunities for, the phenomenon I am studying?

There are several ways to explore and exploit contextual impact in empirical SE research. Menzies et al. [33], for example, suggest that, rather than seeking general principles that apply to many projects, empirical SE should focus on ways to find the best local lessons for groups of related projects. Following Johns [30], we mention a few ways to explore and exploit contextual impact that are related to research design, measurement, analysis, and reporting. However, to succeed with such contextualization, a prerequisite is theoretical grounding and familiarity with the research site(s).

- *Perform cross-level comparative research* that explicitly demonstrate how higher-level situational factors such as environmental uncertainty and market conditions affect lower-level factors such as individual behavior and team autonomy.
- *Perform longitudinal research* that studies processes and examines how behavior unfolds over time or how software teams and organizations configure themselves to deal with recurrent problems.
- *Study critical events* that can punctuate context and make possible research and theory that form part of a larger whole, such as Moe et al.'s [35] study of the introduction of self-managing teams.
- *Collect qualitative data* that illuminate context effects and interactions that might affect behavior in a studied setting, or that can aid in making inferences about the situation.
- *Measure multiple dependent variables* that can uncover situational context when used in conjunction with one another or explain the gap in meaning, such as Dybå et al.'s [20] multiple measurement of software methodology usage.
- *Use analytic strategies* that are sensitive to the distributional properties of data, rather than simply exploring means, and contextual control variables that can explain interactions with main effects, as shown in the pair programming example.
- *Report contextual information* that has theoretical bearing on the study's results or that might be useful to others (e.g., meta-analysts) in the future [13], [14]. A good place to begin is to ensure that the elements of omnibus context are addressed in adequate detail: what was studied, who was studied, where were they studied, when were they studied, and why were they studied?

The last point is especially important to enable the identification of recurring themes or common contextual factors across studies. Systematic reviews conducted with respect to the determination of why study results differ (as they are likely to do), and the evaluation of the potentially contrasting insights from empirical studies will generally be more helpful than those that focus on identifying average effects [13]. Seemingly unpatterned and disagreeing findings from quantitative studies may have underlying consistency when omnibus context is taken into account. Qualitative data may also be useful in capturing developers' subjective evaluations of organizational- or project-level interventions and outcomes. In addition, qualitative findings can be used to develop theories and to identify relevant variables to be evaluated in future quantitative studies.

However, the presence of contextual variables does not mean they will shape software practice or be of theoretical interest. The context must act on, be noticed by, and be construed as important by individuals and groups before it can influence practice. Discovering that contextual variables are present but do not appear to be influential is often as important from a research perspective as confirming their power [36].

Maybe the most critical issue in contextualizing empirical SE research is our willingness, as researchers, to immerse ourselves in the context. Empirical studies in leading SE conferences and journals are often based on laboratory studies using students as subjects. About 90% of the subjects who take part in these experiments are students [44]. The applicability of most experimental results to an industrial setting may, therefore, be questioned.

When researchers move into the field, it is often to administer questionnaires to anonymous respondents who return them by mail, or by performing online surveys of organizational members or various online software communities. Thus, software practice is often studied without going near the organization and without talking to any of its members. Doing research so remote from the industrial context has costs, both in terms of the depth of

understanding researchers can achieve and with respect to the inspiration that leads to new, relevant areas of inquiry.

In general, the more similar the research setting of a study is to the context in which the results will be applied, the more directly relevant the study is perceived. Fenton et al. [23], for example stated that "*evaluative research must involve realistic projects with realistic subjects, and it must be done with sufficient rigor to ensure that any benefits identified are clearly derived from the concept in question. This type of research is time-consuming and expensive and, admittedly, difficult to employ in all software-engineering research. It is not surprising that little of it is being done*."

In this situation, the most realistic research setting is found in action research studies, because the context of the study is the same as the context in which the results will be applied for a given organization, apart from the presence of the researcher(s). The context of industry-based case studies is also generally very similar to the setting of application, although researchers *may* study phenomena that might not be regarded as very relevant by the studied organization. Hence, more (high-quality) action research and case studies should be conducted. The increasing number of qualitative studies appearing in our leading journals may suggest a positive trend in this direction [21].

5. CONCLUSION

The aim of this paper was to provide an overview of how context affects empirical research and how empirical SE research can be better 'contextualized' in order to provide a better understanding of what works for whom, where, when, and why.

Empirical SE is concerned with different technologies, actors, activities, and systems, and therefore contexts, which demand higher levels of contextualization for accuracy in empirical generalization. Progress in this area is unlikely, however, if research is conducted with students in academic settings, through online surveys, or through short visits to companies during which questionnaires are distributed to convenience samples.

Contextualization requires immersion and a focus on relevant phenomena, which means that SE researchers need to invest considerable time within the practice they wish to understand. Action research and case studies applying qualitative and ethnographic methods are examples of approaches that will aid this immersion. Immersion will also enable us to move the discipline into a more useful direction that will counter the common criticism that much empirical SE research is irrelevant for software organizations and their members.

Empirical SE research only becomes comprehensible when one takes into account the larger sociotechnical frameworks within which it is embedded. It is all about context, interpretation, and evaluation. However, what counts as context will depend on the substantive problem under scrutiny; it cannot be captured by generalized lists of discrete variables. So, if we are to move beyond simple assertions that the context is important, we need to articulate more clearly how contextual influences operate.

6. REFERENCES

[1] Arisholm, E. and Sjøberg, D.I.K. (2004) Evaluating the Effect of a Delegated versus Centralized Control Style on the Maintainability of Object-Oriented Software, *IEEE Transactions on Software Engineering*, 30(8): 521-534.

[2] Arisholm, E., Gallis, H.E., Dybå, T. and Sjøberg, D.I.K. (2007) Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise, *IEEE Transactions on Software Engineering*, 33(2): 65-86.

[3] Balijepally, V., Mahapatra, R., Nerur, S., Price, K.H. (2009) Are Two Heads Better Than One For Software Development? The Productivity Paradox of Pair Programming, *MIS Quarterly*, 33(1): 91-118.

[4] Baron, R.M. and Kenny, D.A. (1993) The moderator–mediator variable distinction in social psychological research: Conceptual, strategic, and statistical considerations, *Journal of Personality and Social Psychology*, 51(6): 1173-1182.

[5] Basili, V.R. and Rombach, D. (1988) The TAME Project: Towards Improvement-Oriented Software Environments, *IEEE Transactions on Software Engineering*, 14(6): 758-773.

[6] Basili, V.R., Shull, F., and Lanubile, F. (1999) Building Knowledge through Families of Experiments, *IEEE Transactions on Software Engineering*, 25(4): 456-473.

[7] Bergersen, B.R. and Sjøberg, D.I.K. (2012) Evaluating Methods and Technologies in Software Engineering with Respect to Developers' Skill Level, *Accepted to EASE'2012*.

[8] Bergersen, B.R., Hannay, J.E., Sjøberg, D.I.K., Dybå, T., and Karahasanović (2011) Inferring skill from tests of programming performance: Combining time and quality, *Proc. ESEM'2011*, IEEE Computer Society, pp. 305-314.

[9] Brown, R. (2000) *Group Processes: Dynamics within and between Groups*, Second Ed., Blackwell.

[10] Capretz, L.F. and Ahmed, F. (2010) Making Sense of Software Development and Personality Types, *IT Professional*, 12(1): 6-14.

[11] Chin, E. (1994) Redefining "context" in research on writing, *Written Communication*, 11(4), 445-482.

[12] Clarke, P. and O'Connor, R.V. (2012) The situational factors that affect the software development process: Towards a comprehensive reference framework, *Information and Software Technology*, 54(5): 433-447.

[13] Cruzes, D.S. and Dybå, T. (2011) Research Synthesis in Software Engineering: A Tertiary Study, *Information and Software Technology*, 53(5): 440-455.

[14] Cruzes, D.S. and Dybå, T. (2011) Recommended Steps for Thematic Synthesis in Software Engineering, *Proc. ESEM'2011*, IEEE Computer Society, pp. 275-284.

[15] Dybå, T. (2000) Improvisation in Small Software Organizations, *IEEE Software*, 17(5): 82-87.

[16] Dybå, T. (2003) Factors of Software Process Improvement Success in Small and Large Organizations: An Empirical Study in the Scandinavian Context, *Proc. ESEC/FSE'2003*, ACM press, pp. 148-157.

[17] Dybå, T. (2003) A Dynamic Model of Software Engineering Knowledge Creation, in A. Aurum et al. (Eds.) *Managing Software Engineering Knowledge*, Springer, pp. 95-117.

[18] Dybå, T., Arisholm, E., Sjøberg, D., Hannay, J., and Shull, F. (2007) Are Two Heads Better than One? On the Effectiveness of Pair-Programming, *IEEE Software*, 24(6): 12-15.

[19] Dybå, T., Kitchenham, B.A., and Jørgensen, M. (2005) Evidence-based Software Engineering for Practitioners, *IEEE Software*, 22(1): 58-65.

[20] Dybå, T., Moe, N.B., and Arisholm, E. (2005) Measuring Software Methodology Usage: Challenges of Conceptualization and Operationalization, *Proc. ISESE'2005*, pp. 447-457

[21] Dybå, T., Prikladnicki, R., Rönkkö, K., Seaman, C., and Sillito, J. (2011) Qualitative Research in Software Engineering, *Empirical Software Engineering*, 16(4): 425-429.

[22] Ericsson, K.A. and Charness, N. (1994) Expert Performance: Its Structure and Acquisition, *American Psychologist*, 49(8): 725-747.

[23] Fenton, N., Pfleeger, S.L. and Glass, R.L. (1994) Science and Substance: A Challenge to Software Engineers, *IEEE Software,* 11(4): 86-95.

[24] Goodwin, C. and Duranti, A. (1992) *Rethinking context: Language as an interactive phenomenon*, Cambridge Univ. Press.

[25] Griffin, M. (2007) Specifying organizational contexts: Systematic links between contexts and processes in organizational behavior, *Journal of Organizational Behavior*, 28: 859-863.

[26] Hannay, J., Dybå, T., Arisholm, E., and Sjøberg, D. (2009) The Effectiveness of Pair Programming: A Meta-Analysis, *Information and Software Technology*, 51(7): 1110-1122.

[27] Hedges, L.V. (1987) How Hard Is Hard Science, How Soft Is Soft Science? The Empirical Cumulativeness of Research, *American Psychologist*, 42(2): 443-455.

[28] Höst, M., Regnell, B., and Wohlin, C. (2000) Using Students as Subjects: A Comparative Study of Students and Professionals in Lead-Time Impact Assessment, *Empirical Software Engineering*, 5(3): 201-214.

[29] Johns, G. (1991) Substantive and methodological constraints on behavior and attitudes in organizational research. *Organizational Behavior and Human Decision Processes,* 49: 80-104.

[30] Johns, G. (2006) The Essential Impact of Context on Organizational Behavior, *Academy of Management Review*, 31(2): 386-408.

[31] Karau, S.J. and Williams, K.D. (1993) Social loafing: a meta-analytic review and theoretical integration, *Journal of Personality and Social Psychology*, 65(4): 681-706.

[32] Kitchenham, B.A., Pfleeger, S.L., Pickard, L.M., Jones, P.W., Hoaglin, D.C., El Emam, K. and Rosenberg, J. (2002) Preliminary Guidelines for Empirical Research in Software Engineering, *IEEE Transactions on Software Engineering*, 28(8): 721-734.

[33] Menzies, T., Butcher, A., Marcus, A., Zimmermann, T., and Cok, D. (2011) Local vs. Global Models for Effort Estimation and Defect Prediction, *Proc. ASE'2011*, pp. 343-351.

[34] Michailova, S. (2011) Contextualizing in International Business research: Why do we need more of it and how can we be better at it? *Scandinavian Journal of Management*, 27: 129-139.

[35] Moe, N.B., Dingsøyr, T. and Dybå, T. (2009) Overcoming Barriers to Self-management in Software Teams, *IEEE Software*, 26(6): 20-26.

[36] Mowday, R. and Sutton, R. (1993) Organizational behavior: Linking individuals and groups to organizational contexts, *Annual Review of Psychology*, 44: 195-229.

[37] Petersen K. and Wohlin, C. (2009) Context in Industrial Software Engineering Research, *Proc. ESEM'2009*, pp. 401-404.

[38] Rousseau, D.M. and Fried, Y. (2001) Location, location, location: contextualizing organizational research, *Journal of Organizational Behavior*, 22: 1-13.

[39] Shull, F. (2012) I Believe! *IEEE Software*, 29(1): 4-7.

[40] Sjøberg, D, Dybå, T., and Jørgensen, M. (2007) The Future of Empirical Methods in Software Engineering Research, *Proc. FOSE'2007*, pp. 358–378.

[41] Sjøberg, D., Dybå, T., Anda, B., and Hannay, J. (2008) Building Theories in Software Engineering, in F. Shull, J. Singer, and D. Sjøberg (Eds.) *Advanced Topics in Empirical Software Engineering*, Springer, pp. 312-336.

[42] Sjøberg, D.I.K., Anda, B., Arisholm, E., Dybå, T., Jørgensen, M., Karahasanovic, A., Koren, E.F. and Vokac M. (2002) Conducting Realistic Experiments in Software Engineering, *Proc. ISESE'2002*, pp. 17-26.

[43] Sjøberg, D.I.K., Anda, B., Arisholm, E., Dybå, T., Jørgensen, M., Karahasanovic, A., and Vokac M. (2003) Challenges and Recommendation when Increasing the Realism of Controlled Software Engineering Experiments, in R. Conradi & A.I. Wang (Eds.) *Empirical Methods and Studies in Software Engineering - Experiences from ESERNET*, Springer, LNCS 2765, pp. 24-38.

[44] Sjøberg, D.I.K., Hannay, J.E., Hansen, O., Kampenes, V.B., Karahasanović, A., Liborg, N.-K. and Rekdal, A.C. (2005) A Survey of Controlled Experiments in Software Engineering, *IEEE Transactions on Software Engineering*, 31(9): 733-753.

[45] Trist, E. (1981) The Evolution of Socio-Technical Systems: A Conceptual Framework and an Action Research Program, *Occasional papers No. 2,* Ontario Quality of Working Life Center.

[46] Turhan, B., Layman, L., Diep, M., Shull, F. and Erdogmus, H. (2010) How Effective is Test Driven Development?, in G.Wilson & A. Orham (Eds.), *Making Software: What Really Works, and Why We Believe It*, O'Reilly Press, pp. 207-219.

[47] Welter, F. (2010) Contextualizing Entrepreneurship: Conceptual Challenges and Ways Forward, *Entrepreneurship Theory and Practice*, 35(1): 165-184.

[48] Whetten (2009) An Examination of the Interface between Context and Theory Applied to the Study of Chinese Organizations, *Management and Organization Review*, 5(1): 29-55.

[49] Whetten, D.A. (1989) What Constitutes a Theoretical Contribution, *Academy of Management Review,* 14(4): 490-495.

[50] Xu, P. and Ramesh, B. (2007) Software process tailoring: an empirical investigation, *Journal of Management Information Systems*, 24(2): 293-328.