Teaching Evidence-Based Software Engineering to University Students

Magne Jørgensen[1,2], Tory Dybå[1,3], Barbara Kitchenham[4]
1) Simula Research Laboratory, Norway, 2) Hedmark University College, Rena, Norway, 3) SINTEF, Norway, 4) University of Keele, UK
magnej@simula.no, Tore.Dyba@sintef.no, ap_kitchenham@onetel.com

Abstract

Evidence-based software engineering (EBSE) describes a process of identifying, understanding and evaluating findings from research and practice-based experience. This process aims at improving software engineering decisions. For the last three years, EBSE has been taught to university students at Hedmark University College, Rena, Norway. The motivation for the EBSE-course is that it is essential for the students, as future practitioners, to learn how to base important software engineering decisions on the systematic and critical evaluation of the best available evidence. The main purpose of this paper is to inspire and support other universities in their work on developing their own EBSE-courses. For this purpose we report on how our course has been organized and what lessons have been learned. There are currently no studies available on the effects of teaching EBSE and, as far as we know, only we have gained practice-based experience. To acquire more knowledge about the costs and benefits of teaching EBSE we hope that other universities will develop their own EBSE-courses and report their experience.

1. Introduction

In [1, 2] we introduce "Evidence-Based Software Engineering" (EBSE). The aim of EBSE is to improve decision making related to software development and maintenance by collecting and evaluating the best evidence from research studies and practice-based experience. Without good skills in identifying, understanding and evaluating findings from research and relevant practice-based experience, important findings may be transferred to the software industry slowly, or not at all. However, efficient search, reliable interpretation and the proper use of relevant results may require a basic understanding of scientific method together with the adoption of an inquisitive and skeptical approach. Training in these skills may be necessary to derive the potential benefits of EBSE.

The main steps of EBSE are as follows:

1. Convert a relevant problem or need for information into an answerable question.
2. Search the literature for the best available evidence to answer the question.
3. Critically appraise the evidence for its validity, impact, and applicability.
4. Integrate the appraised evidence with practical experience and the client's values and circumstances to make decisions about practice.
5. Evaluate performance in comparison with previous performance and seek ways to improve it.

A more complete description of EBSE and its related activities can be found in [1, 2].

This paper reports the lessons learned from teaching EBSE to students at Hedmark University College in Rena, Norway in the period 2003-2005. The main purpose is to stimulate other universities to develop similar courses. Our EBSE course is, as far as we know, the earliest, and possibly the only, EBSE course taught in the world.

The paper is organized as follows:

- Section 2 motivates our teaching of EBSE to software engineering students.
- Section 3 outlines the structure of our EBSE course and gives example of course elements and lessons learned.
- Section 4 discusses the rationale for teaching EBSE.
- Section 5 concludes.

2. Motivation for teaching EBSE

The main motivation for the EBSE course is to provide future software professionals with the

knowledge, experience, attitude and skill to enable them to make better decisions. Our belief in the value of teaching EBSE to university students is based on, amongst other things, the following assumptions and beliefs:

- The software industry will benefit from moving in the direction of evidence-based software engineering. We argue for this claim in [1, 2].
- Software engineering students, in general, have insufficient knowledge and practice regarding the evaluation of arguments. In their professional lives, they are frequently required to arbitrate between the conflicting conclusions of different lines of argument. An EBSE course may support the critical and systematic evaluation of arguments.
- Students need to learn to collect information from all types of sources, e.g., from library data bases, textbooks, the internet and other people's experiences, and to assess critically the relevance and validity of this information. In our experience, few software engineering university courses address this. For example, the students we have taught EBSE had never had any teaching in how to examine scientific studies critically or how to systematically evaluate the arguments presented in course textbooks and computer magazines.
- Universities should have a stronger focus on how to acquire new knowledge and skill. Knowledge of, and skill in applying, particular technologies may soon be outdated. Skills in the formulation of meaningful questions, the identification of relevant information and the critical assessment of studies/arguments are, by contrast, of more long-lasting value.
- The software industry is full of "hype", e.g., small changes of old methods heavily marketed as methods that will revolutionize the productivity of the software industry. Teaching EBSE to university students may be an important means for software engineering to become a more mature discipline with more resistance towards "hype". In particular, we believe that EBSE may lead to more critical assessment of development methods and marketing ploys.

Other studies and argumentation in support of the need for training in the evaluation of claims and evidence can be found in [3] (in particular, pages 383-392 are relevant).

It is not easy to study the degree to which these assumptions are true and the degree to which EBSE will have a positive impact on software practice. It is, for example, not obvious what to compare EBSE practice with, out of all the other decision-making practices. Further, we should obviously not expect a university course in EBSE to revolutionize the world of software development. However, if there are even a few occasions upon which the EBSE skills acquired prove to be of significant value in actual industry practice, it may be worth the effort of teaching EBSE. The rationale for teaching EBSE is discussed in Section 4.

3. Teaching EBSE to university students

3.1. Learning goal

The learning goal of our EBSE-course is formulated as: *"... to learn to practice evidence-based software engineering. This means the ability to identify, evaluate and apply valid and relevant research results and practice-related experience as the basis for judgments and decisions in software development."*

3.2. Participants

The participants of the EBSE course have, so far, been students at Hedmark University College ("høgskole"), Rena, Norway in 2003, 2004 and 2005. The course is mandatory for students following the systems development program at the university college. About 10-20 students have taken the EBSE course each year. These students are in their 3rd year (their final year at the university college) and they are about to complete their Bachelor degree. Many of them will start working as software professionals immediately after completing the EBSE course and some will continue as MSc students at other universities. The lecturer is the first author of this paper. The course elements have been based on input from all three authors.

3.3. Course structure

The course has two intensive teaching modules and a supervised project task. The work load is about 1/6th of the total load in the Winter/Spring semester, which represents an average work-load of 6-8 hours per

week. The supervised project task, to be completed individually, constitutes the examination. The project task consists of completing EBSE steps 1-4. The students typically start their project work immediately after the first teaching module. The timeline of the lectures and project deliveries are as follows:

Week 1: *Teaching module 1 (6 hours)*
Week 1-4: *Supervised individual project work.*
Week 5: *Delivery of first version of problem formulation.*
Week 5: *Teaching module 2 (12 hours)*
Week 5-11: *Supervised individual project work.*
Week 11: *Delivery of complete project report to be evaluated.*

The project delivery (the EBSE practice work) is evaluated as "passed" or "not passed". In order to pass, a project must fulfill the following criteria:

- An answerable software engineering question/problem must be formulated properly. This requires that terminology be explained and that the problem be specified precisely. The question/problem should be relevant for software practitioners. An example of a proper problem formulation is the following: "*What is the effect of X, for organizations/developers of type Y, in situations of type Z.*"
- An extensive search for relevant research results and practice-related experience must be conducted. In practice, we have required that, a) Available university library search facilities are used with appropriate search terminology, b) At least one expert on the topic is identified and contacted for information, and, c) At least two companies with relevant practice-based experience are contacted.
- The relevance and validity of the results, opinions, viewpoints received from the different sources must be evaluated properly.
- A cogent argument must be constructed that marshals the available evidence to support a conclusion. (The conclusion may be that it is not possible to form a definitive opinion, or that the evidence in favor of a particular decision or answer is weak.)

The students are stimulated to discuss with each other, but the project task is carried out individually, i.e., each student has his/her own problem formulation and project report. Our decision to have individual project work is based principally on practical considerations related to examination-based evaluation. Team work would, on the other hand, be more realistic.

3.4. Lectures

The main elements of the EBSE lectures taught are as follows:

- Motivation for EBSE (see Section 2).
- The steps of EBSE. This part of the lectures includes several examples of how to, and how not to, conduct the decision steps.
- Introduction to scientific method. We found it unrealistic to explain to the student all the steps and elements of scientific inquiry and had to make a selection. The course focuses on:
  - The general steps of scientific method. We found Wallace's cyclic model of science useful for this purpose. In particular, this model illustrates well the role of induction and deduction in theory building.
  - Strengths and weaknesses of the experimental and the observational method. The difference between correlation and cause-effect studies is emphasized.
  - Basic knowledge on how to evaluate scientific studies. We use the empirical study guidelines described in [4] as the starting point for this part of the lectures.
  - Common biases found in scientific engineering studies, e.g., the "Hawthorne effect", "question framing effects" and "theory-loaded observation".
  - Basic statistics. Here, the focus is not on teaching the students sophisticated statistical methods, but on teaching the strengths and weaknesses of commonly applied statistical techniques. In particular, we have found it useful to train the students in the identification of biased sampling

and biased allocation to treatments. This part includes several practical exercises and discussions, e.g., a discussion on whether a high number of observations can compensate for a biased sampling or treatment allocation method or not.

- How to evaluate practice-based evidence. This includes guidelines on how to evaluate the basis and relevance of experiences and opinions. In our experience, there are no large conceptual differences in how to evaluate scientific papers and how to evaluate practice-based experience, i.e., the guidelines are much the same. The types of evidence and the formality of the argumentation are, however, different and require practice to master properly.
- Argumentation theory. This part of the course is based mainly on the textbook "*Attacking faulty reasoning: A practical guide to fallacy-free arguments*" [5]. Examples of content:
    - Building the elements of an argument based on Toulmin's model [6], see Appendix 1.
    - Types of argument, e.g., arguments based on cause-effect, correlation, generalization, similarity, and authority.
    - Potentially manipulative elements in arguments, e.g., inappropriate generalizations, irrelevant arguments, circular argumentation, appeal to emotions, biased or imprecise use of terminology, use of humor, and reference to tradition.
    - How to construct a good argument, e.g., avoidance of premature formulation before the information is collected and analyzed, inclusion of relevant evidence only, clarification of the scope of the argument, clarification of important terminology, balanced analysis of evidence for and against, focused argumentation, and logical connection between evidence and conclusion.

3.5. Exercises on the evaluation of arguments

About 30% of the lectures consist of practical exercises in evaluating scientific studies and expert opinion articles that contain arguments pertaining to software engineering. Most of the study material is currently based on articles found in IEEE Software. We have found articles in IEEE Software to be very useful as material upon which to practice the critical evaluation of arguments, experience and opinions in software engineering. These articles are meant to be read and understood by educated practitioners. Examples of papers from IEEE Software used in the critical evaluation exercises are [7, 8].

Possibly, these exercises are the most important part of the EBSE course and the part regarding which we have received the best (informal) feedback from the students. For example, several students have, when learning how to assess arguments critically and read scientific and opinion-based papers efficiently, given feedback like: "*Why didn't we learn this in the beginning of our studies? This is really useful.*" This type of feedback suggests that teaching EBSE may have a positive impact on the students' performance in other courses at the university. The models for the evaluation of argument and checklist used by the students for the critical reading exercises and for the project work are described in Appendices 1 and 2.

3.6. Student projects

The completion of the project task demands quite a lot from the students. As opposed to previous courses they have had, they have to define the problem themselves, find relevant information themselves and to assess critically every piece of information identified. We have, not surprisingly, observed large variation in how well the principles of EBSE principles are applied. In particular, some students struggle with the difference between summarizing a study and critically assessing the relevance and validity of a study. Most students in our courses were, with supervision from the course instructor, able to do a good job on formulating problems, searching for information, assessing the information critically and using the information to come to a better understanding of the problem solution. The best project reports would, according to our assessment, provide good input to real-world software engineering decisions.

Examples of problem areas addressed by the students are the following:

- What are the benefits of using pair programming, as opposed to individual programming, and what are the conditions for deriving these benefits?
- What are the relations between project size and benefits from the use of XP?
- When are IT projects with a great deal of user contribution more successful than those with less user contribution?

The terminology used in the formulation of the problem, e.g., the meaning of "benefit" and "successful", and the scope of the assessment is (at least in the best project reports) defined and explained.

Perhaps the most important challenge with many of the problem formulations has been to identify relevant scientific studies. We did not want the students to change from a problem relevant to industry to one of less industry relevance because of this challenge. Especially, in this regard, we did not want them to fit the problem to the evidence too much. This meant that many projects had to base their argumentation on critical assessments of expert opinions and practice-based experience alone. This challenge, we think, illustrates an important difference between EBSE and its origin (evidence-based medicine), i.e., there may typically be more relevant scientific studies pertaining to problems in evidence-based medicine. A possible consequence of this difference is that teaching EBSE should have less focus on evaluating scientific studies and more on evaluating practice-based evidence and argument. Our course has gradually changed its focus in accordance with that observation.

4. Should we teach EBSE?

Author names and affiliations are to be centered beneath the title and printed in As stated earlier, the main goal of this paper is to stimulate other universities to develop courses similar to our EBSE course, and to support them in their endeavor. We cannot claim that we have demonstrated that teaching EBSE has a significant positive effect on real-world software development work (though it is our hope that it does).

In accordance with our checklist for the evaluation of arguments, the reader of this paper should be skeptical about our opinions and findings. We were the first to describe the steps of evidence-based software engineering and have taught the first course on this topic, so we may have vested interests and our reported findings may be biased by those interests.

4.1. Empirical evidence

There are no scientific studies on the effects of EBSE. The empirical evidence regarding its effectiveness is constituted only by our own practice-based (possibly biased) experience of teaching it. There are, however, studies on teaching a similar topic, i.e., evidence-based medicine. As stated earlier, EBSE borrows the principles and steps from evidence-based medicine and the two disciplines have many similarities. An examination of the presentation material in several evidence-based medicine courses (see www.ebmny.org/teach.html for a sample of courses) suggests that there are important domain-specific differences, and differences in focus on the evaluation of scientific studies compared to practice-based experience. Hence, the wholesale transfer of results from teaching evidence-based medicine to teaching EBSE is of uncertain value. It is, on the other hand, perhaps the best empirical evidence that currently exists.

We began our search for information about the effects of teaching evidence-based medicine with a search for systematic reviews. The Cochrane database (www.cochrane.org) was developed for the purpose of helping people to make well-informed decisions by preparing, maintaining and promoting the accessibility of systematic reviews of the effects of healthcare interventions. Fortunately, this database includes a review of the effects of teaching evidence-based medicine [9]. In the abstract of the review it is stated that: "*There is evidence that critical appraisal teaching* [a subset of evidence-based medicine teaching] *has positive effects on participants' knowledge, but as only one study met the inclusion criteria the validity of drawing general conclusions about the effects of teaching critical appraisal is debatable. There are large gaps in the evidence as to whether teaching critical appraisal impacts on decision-making or patient outcomes. It is also unclear whether the size of benefit seen is large enough to be of practical significance, or whether this varies according to participant background or teaching method. The evidence supporting all outcomes is weakened by the generally poorly designed, executed and reported studies that we found.*"

The empirical evidence is, consequently, weak and uncertain, but nevertheless in favor of teaching EBSE.

4.2. Analytical argumentation

This section provides what we believe are the major arguments in favor and disfavor of teaching EBSE.

The claims (arguments) we make are mainly based on our own observations, knowledge and beliefs and their strengths should be interpreted accordingly. This section builds on the discussion in Section 2, where we motivate the EBSE-course.

Major arguments in favor of teaching EBSE:

- The critical evaluation of arguments is a useful skill for software engineering practitioners. This skill is frequently not sufficiently developed by practice alone, and has to be taught. The skill is currently not part of most programs of education on software engineering.
- The introduction of an EBSE course would replace another course also believed to have value. The question is, consequently, whether EBSE is more valuable than certain other courses currently taught. We believe that an important argument in favor of EBSE is that university courses should increase the focus on more robust knowledge. For example, while much of the knowledge acquired in courses teaching the use of particular technologies may soon be outdated, the ability to evaluate and construct arguments is of life-long value.

Major arguments against teaching EBSE:

- There are skills other than EBSE skills that may be more important for good software engineering decisions, e.g., good knowledge about the technology to be evaluated, good organization and domain knowledge, and, good ability to conduct empirical effect studies. Teaching EBSE skills instead of other potentially important decision-making skills may have a negative net effect on software engineering practice, even if the EBSE course has a positive effect.
- A change from the status quo to something else should require good evidence. The burden of evidence is on those who propose a change. Work needs to be done to assess the effectiveness of EBSE on industry practice. Current evidence in favor of EBSE is available only from comparison with the effectiveness of the teaching of evidence-based medicine and the authors' observations of the results of their own EBSE course.

5. Conclusion

The goal of this paper is to inspire universities to include a course in evidence-based software engineering (EBSE) in their software engineering education, and to support them in their endeavor. To that end, we describe elements of the course and lessons learned.

The scientific evidence in favor of teaching EBSE is weak (based on transfer of results from studies on teaching evidence-based medicine) and our practice-based experience is potentially biased (we may have vested interests in describing the results positively). There is consequently a need for more trials on teaching EBSE to university students before we can make confident claims about the cost/benefit. We hope that other university employees will be inspired by this paper and will report their experience and, preferably, their measurements of the effects. When more people have experience of teaching EBSE, forums for the exchange of course material and experience could be established, as in evidence-based medicine (see for example "Evidence-based medicine Resource center" at www.ebmny.org./teach.html).

Trials in industrial and more controlled contexts should be conducted, as well. Teaching EBSE to software practitioners may lead to a more immediate impact on real-world decisions and require only minor changes in the course material. Controlled experiments comparing real-world decisions of students or software professionals, with or without EBSE skills, would be useful for acquiring more, and more objective, evidence about the costs and benefits of teaching/learning EBSE.

References

1. Dybå, T., B. Kitchenham, and M. Jørgensen, *Evidence-Based Software Engineering for Practitioners.* IEEE Software, 2005. **22**(1): p. 58-65.
2. Kitchenham, B., T. Dybå, and M. Jørgensen. *Evidence-based Software Engineering*. in *International Conference on Software Engineering (ICSE)*. 2004. Edinburgh: p. 273-281.
3. Presley, M. and C.B. McCormick, *Advanced educational psychology*. 1995, New York: HarperCollins College Publishers.
4. Kitchenham, B., et al., *Preliminary Guidelines for Empirical Research in Software Engineering.* IEEE Transactions on Software Engineering, 2002. **8**(8): p. 721-734.
5. Damer, T.E., *Attacking faulty reasoning: A practical guide to fallacy-free arguments*. 2001: International Thomson Publishing.

6. Toulmin, S., *The Uses of Argument.* 1958, Cambridge: Cambridge University Press.
7. Beck, K., *Aim, Fire.* IEEE Software, 2001. **18**(5): p. 87-89.
8. Williams, L., et al., *Strengthening the Case for Pair Programming.* IEEE Software, 2000. **17**(4): p. 19-25.
9. Parkes, J., C. Hyde, and R. Milne, *Teaching critical appraisal skills in health care settings.* The Cochrane Database of Systematic Reviews, 2001(3).

APPENDIX 1: Toulmin's model of argumentation

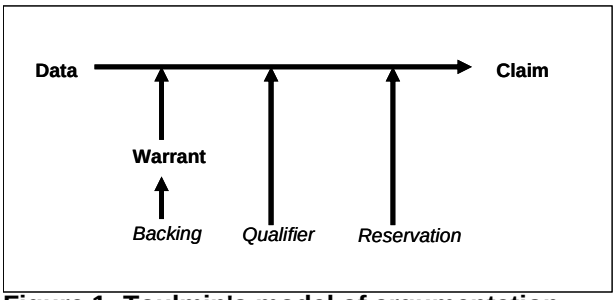

Figure 1: Toulmin's model of argumentation

Figure 1 graphically displays the elements of Toulmin's model. The primary elements of an argument, according to Toulmin's mode, are in **bold** letters, and the secondary elements in *italic*.

Toulmin's model of argumentation can be viewed as a layout of argument. This layout of argument, we believe, is useful for the students as a mental model when evaluating and/or constructing arguments. In particular, we find the model useful during the exercises on the evaluation of arguments. The checklist in Appendix 2 is based on Toulmin's model of argumentation.

We recommend that the student start with the identification of the **claims** or conclusions made by the authors. These are normally found in the conclusion section of the papers or in the abstract, but may be found other places as well. Poor papers may, in fact, have no explicit claims at all. The students are requested to evaluate the claim, e.g., whether the claim is circular or vague. We also ask the students to identify the **qualifiers**, i.e., statements about the strength of the claim, and the **reservations**, i.e., statements about the limitations of the claim. These are important when later evaluating the relevance of the evidence and the connection between evidence and claim. For example, a claim that is qualified with "this weakly indicates a cause-effect relationship" should be evaluated differently from the claim "there is a cause-effect relationship."

Then, we ask the student to look for the **data**, i.e., the evidence supporting the claim. In particular, we ask them to evaluate the relevance of the evidence. We frequently find that the students are surprised by how little relevant evidence a lengthy software engineering paper contains.

Finally, we ask the students to look for the **warrant**, i.e., the supporting connection between the data and the claim. This is frequently the most difficult part of the evaluation of the argumentation, where the critical appraisal ability and analytical skill of the students is most important. The students are requested to evaluate the degree to which the relevant data supports the claim. To support this evaluation step, the students are taught guidelines for how empirical studies should be conducted [4] and general guidelines on ethical issues pertaining to such studies when conducted on human subjects. The warrants may have a **backing**, i.e., an argument that supports a connection of confirmation or deduction between the data and the claim. When it is not obvious that the connection between data and claim is valid (or invalid), we ask the students to search for elements that the authors use to support it (the backing). This may, for example, consist of analytical argumentation or evidence supporting the specific interpretation of data conducted by the authors.

We have experienced that familiarity with the model just outlined does more than change how the students evaluate argumentations. It also seems to lead to a more efficient and critical reading of software engineering texts that may be useful for the student in other contexts. For example, we have observed that most of them replace the mechanical reading of papers from the first to the last page, to a more information seeking, flexible reading strategy. If the information about potential vested interests and the claims are found in at the last page of a paper (as it is in most IEEE Software paper), it is natural to start the reading there. In addition, more knowledge about the elements of argumentation may have enabled the students to improve the quality of their own arguments.

APPENDIX 2: Checklist for the evaluation of argumentation

1. Be a skeptic!
2. Remember that it is the argument that you are supposed to evaluate, not how much you agree with the claims.
3. Start with the identification of the main claims.
4. Assess the relevance of the claims for your purpose.
5. Before you read the paper, assess whether or not it is likely that the authors have vested interests in the claims. If yes, how might this affect the results? What is the background and scope of the previous experience of the author? Is it likely that this biases the search for evidence and the conclusion?
6. Read the paper with the purpose of identifying evidence that supports the claims. Skip the irrelevant parts the first time you read the paper.
7. Evaluate the relevance and validity of the evidence. Assess whether it is opinion-based, example-based, based on a systematic review of scientific studies, etc. Is the evidence credible?
8. Evaluate the connection between the evidence and the claim. Is the claim a possible, likely, or, necessary consequence?
9. Check the use of measures and statistical methods. In particular, assess randomness in selection of subjects and allocation of treatment when statistical hypothesis testing is used. If not random, assess the effect of the non-randomness.
10. Search for manipulating elements, e.g., text that is not relevant for the argument, or loaded use of terminology used to create sympathy or antipathy. If large parts of the text are not relevant, evaluate the intended function of that part. Be aware of rhetorical elements.
11. Assess the degree to which the norms of ethical argument are broken (these norms are part of the course material).
12. Assess whether the inclusion of evidence is one-sided or gives a wrong picture.
13. Assess whether weaknesses of the study are properly discussed. If not discussed at all, why not?
14. Try to identify missing evidence or missing counter-arguments. Be aware of your tendency to evaluate only what is present and forget what is not included.
15. Be particularly careful with the evaluation of the argumentation if you are sympathetic to the conclusion. Our defense against "theory-loaded evaluation" and "wishful thinking" is poor and must be trained. Put in extra effort to find errors if you feel disposed to accept the conclusion in situations with weak or contradictory evidence.
16. Do not dismiss an argument as having no value, if it has shortcomings. There are very few bullet-proof arguments and we frequently have to select between weak and even weaker arguments in software engineering contexts. A weak argument is frequently better than no argument at all.